Bayerische Julius-Maximilians-Universität
Würzburg
Institut für Informatik
Lehrstuhl für Verteilte Systeme
Prof. Dr.-Ing. P. Tran-Gia

Modeling and Optimization of Cluster Tools in Semiconductor Manufacturing

Mathias Dümmler

Würzburger Beiträge zur
Leistungsbewertung Verteilter Systeme

Bericht 01/2004

Würzburger Beiträge zur Leistungsbewertung Verteilter Systeme

Herausgeber

Prof. Dr.-Ing. P. Tran-Gia
Universität Würzburg
Institut für Informatik
Lehrstuhl für Verteilte Systeme
Am Hubland
D-97074 Würzburg

Tel.: +49-931-888-6630
Fax: +49-931-888-6632
email: trangia@informatik.uni-wuerzburg.de

Satz

Reproduktionsfähige Vorlage vom Autor.
Gesetzt in LaTeX Palatino 9pt.

ISSN 1432-8801

Modeling and Optimization of Cluster Tools in Semiconductor Manufacturing

Dissertation zur Erlangung des
naturwissenschaftlichen Doktorgrades
der Bayerischen Julius–Maximilians–Universität Würzburg

vorgelegt von

Mathias Dümmler

aus
Kitzingen

Würzburg 2004

Eingereicht am: 21.11.2003
bei der Fakultät für Mathematik und Informatik
1. Gutachter: Prof. Dr.-Ing. P. Tran-Gia
2. Gutachter: Prof. Dr. John W. Fowler
Tag der mündlichen Prüfung: 09.02.2004

Danksagung

Ich möchte meinem Doktorvater und Betreuer dieser Arbeit, Herrn Prof. Dr.-Ing. Phuoc Tran–Gia herzlich dafür danken, dass ich diese Arbeit an seinem Lehrstuhl in einer Atmosphäre wissenschaftlicher Freiheit anfertigen durfte. Durch die fachlichen Diskussionen, die Möglichkeit zum Austausch mit nationalen und internationalen Kollegen auf Konferenzen, die Kontakte zu Industrie und Forschung, sowie die hervorragende Ausstattung und das kollegiale Klima an seinem Lehrstuhl waren hervorragende Arbeitsbedingungen geschaffen.

My deepest gratitude and appreciation go to Prof. Dr. John Fowler, who acted as a reviewer of this thesis and provided me with valuable comments on my research.

An Dr. Alu Schömig geht mein Dank dafür, dass er mich während meiner Zeit als Student am Lehrstuhl für Informatik III in die wunderbare Welt der Modellierung von Fertigungssystemen eingeführt hat. Dr. Oliver Rose danke ich für die gute Zusammenarbeit in der Arbeitsgruppe „Leistungsbewertung von Fertigungssystemen“sowie in einer Reihe von wissenschaftlichen und industriellen Projekten, weiterhin natürlich für die zahlreichen gemeinsam verbrachten Dienstreisen.

Zu herzlichem Dank bin ich auch den Studierenden verpflichtet, die mich als wissenschaftlichen Mitarbeiter unterstützt haben. Dazu gehören vor allem die Diplomanden Markus Bohr und Matthias Schmid, die in ihren Diplomarbeiten wesentliche Grundlagen für die vorliegende Arbeit geschaffen haben. Christian Köhler, Heiko Niedermayer und Andreas Reifert sei an dieser Stelle ebenfalls gedankt.

Meinen ehemaligen Kollegen am Lehrstuhl für Verteilte Systeme, Dr. Notker Gerlich, Klaus Heck, Michael Menth, Dr. Michael Ritter, Dirk Staehle, Dr. Kurt Tutschku, Dr. Norbert Vicari, Patricia Wilcox und natürlich Frau Alt möchte ich für die gute Zusammenarbeit danken. Dr. Kenji Leibnitz gilt besonderer Dank für die kritische Durchsicht des Entwurfs dieser Arbeit.

Eine Arbeit wie diese wäre ohne vielfältige Kontakte und Gespräche mit Kollegen im In– und Ausland nicht möglich gewesen. Viele Ideen zu dieser Arbeit sind aus Diskussionen mit Mitarbeitern der Infineon Technologies AG entstanden. Herzlichen Dank daher an Jörg Domaschke, Dr. Hermann Gold und Hans–Jürgen Wimberger. Furthermore, collaborations and discussions with Dr. Frank Chance and Dr. Jennifer Robinson have contributed to my work as a researcher.

Meinen Eltern schulde ich tiefen Dank dafür, dass sie mich während der Studien– und Promotionszeit stets unterstützt haben. Patricia Buschmann danke ich für ihre Liebe und Aufmunterung während der letzten Monate meiner Promotionsarbeit.

Contents

1 Introduction

After the dramatic worldwide economical downturn in the year 2001, a recovery of the semiconductor industry was expected following some positive press releases of the major players in the beginning of the second quarter of 2002. However, at least for the providers of semiconductor manufacturing equipment, the year 2002 brought a ”double dip”, i.e., another downturn in revenue. After a decrease of 41 percent in revenue in 2001 to USD 28 billion, manufacturers of semiconductor equipment reported sales to decline another 31 percent to a total of USD 19.7 billion of new chip manufacturing, testing, and assembly equipment in 2002, cf. (Tracy 2003). The reduced spending of the semiconductor manufacturers is implied, among other things, by the severe conditions of the overall economy, geopolitical uncertainties, and local incidents with global implications, like the outbreak of SARS in parts of Asia.

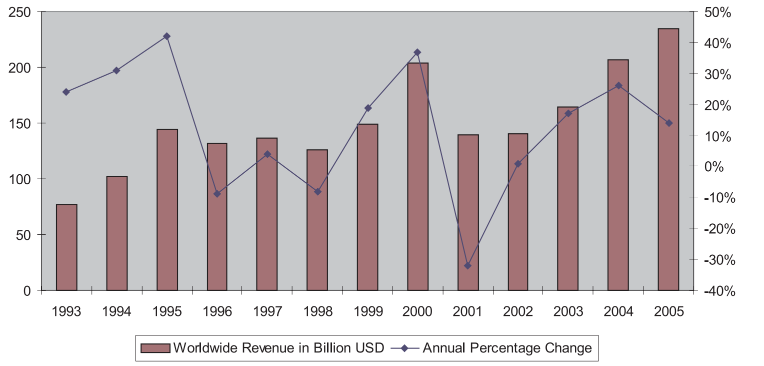


Figure 1.1: *Worldwide Semiconductor Market (Sources: WSTS for Historical Data until 2002, InStat for Forecast)*

Current forecasts are slightly more optimistic, predicting an increase of 8 percent in revenue in the worldwide semiconductor market in 2003 to USD 168 billion in total, cf. (Gartner, Inc. 2003), and improved quarterly results for the equipment suppliers, cf. for example (Applied Materials, Inc. 2003). Figure 1.1 shows a chart of the worldwide annual revenue in the semiconductor market.

One of the few constants in semiconductor business is the ongoing effort to reduce costs and to increase productivity at the same time. Strong growth in the industry is not expected until the year 2004, so investments in new manufacturing equipment have been reduced to a minimum in the past. Engineers try to get the most out of the equipment by planning the resources more carefully, optimizing the operations, and putting more intelligence into the control of the manufacturing process. Improving operational processes has been identified to be among the most promising opportunities to reduce costs in semiconductor fabrication facilities, cf. (Schömig and Fowler 2000).

Producing semiconductors requires a very cost–intensive and sophisticated manufacturing environment. With the size of the structures built on a semiconductor chip decreasing, the production costs are increasing at the same pace. Semiconductor production in a modern fab requires several hundreds of machines, with prices reaching several millions of USD per machine. On a single silicon disc, called *wafer*, up to several hundreds of chips are located. The small structural sizes of several micrometers make the wafers very sensitive to even smallest particles polluting the wafer surface. A single dust particle can render a wafer unusable.

Having the above as a background, this monograph focuses on the performance modeling and optimization of a particular class of equipment in semiconductor manufacturing, the so called cluster tools. Cluster tools have been identified as a high–potential cost saver, cf. (Singer 1995) and (Bader et al. 1990). By integrating a series of processing steps, transportation, and control into a single cluster tool, the risk of contamination is reduced, leading to an improved yield. Additionally, the floor space required in the semiconductor fab and the need for human intervention are reduced, and transportation is simplified.

Since cluster tools are complex and expensive machines, it is essential to have a thorough understanding of the performance characteristics of these tools. Hence, in this monograph it is shown which approaches for modeling the performance of cluster tools are available and how these performance models can be used in the planning and control of a semiconductor fab. Furthermore, the aim of this monograph is to show areas where the operation of cluster tools can be improved by optimizing the procedures within a cluster tool, as well as possibilities for a better integration of the cluster tools into the control of the overall wafer fabrication.

The actual physical and chemical processes taking place in a cluster tool are not considered in detail in this monograph. For the purpose of this study, it is sufficient to represent these processes by process duration and potential pre– and post–processing timing constraints. Other process parameters like temperature and vacuum levels are not considered.

This monograph is organized as follows. Chapter 2 provides an introduction into the manufacturing of semiconductors. A summary of the manufacturing process is presented, and the special role of cluster tools, which are in the focus of this monograph, is explained. Chapter 3 gives an overview of approaches for performance modeling of cluster tools. These approaches can be divided into analytical and simulative approaches. In Chapter 4, methods for scheduling the processes within a single cluster tool are presented. Whereas Chapter 4 is dedicated to single cluster tools in isolation, Chapter 5 contains studies with the goal of optimizing the operation of pools of cluster tools by intelligently distributing the work load among the cluster tools. Chapter 6 summarizes the results presented in this monograph and gives directions for future research.

2 Semiconductor Manufacturing and Cluster Tools

In this chapter, the steps required for transforming silicon, the raw material of wafer manufacturing, into integrated circuits are described. It is explained how cluster tools are applied during this manufacturing process, and the benefits of using cluster tools as compared to other tool types are specified.

2.1 Overview of Chip Manufacturing

Integrated circuits manufactured in today's chip manufacturing facilities (or short: *fabs*) consist of millions of transistors, resistors, and capacitors on a single silicon chip of only a few square centimeters in size. The manufacturing sequence necessary to produce these highly miniaturized integrated circuits consists of five basic steps: wafer preparation, wafer fabrication, probe, assembly, and test, cf. (Schömig and Fowler 2000).

Since the focus of this monograph will be on the application of cluster tools in the second step, the wafer fabrication, the other steps will be described only briefly. *Wafer preparation* consists of the processes required to produce raw wafers out of the semiconductor raw material, that in most of the cases is silicon. *Wafers* are thin, usually round slices of a semiconductor material. The diameters of a wafer in mass production usually range from six to twelve inch (150 to 300 mm), smaller diameters down to one inch are possible as well. A single wafer can contain up to several hundreds of integrated circuits. In order to make the raw wafers ready for processing in the fabrication step, additional treatment like polishing and edge grinding is required.

During *probe*, taking place immediately after the fabrication step, the individual circuits or *dice* on a wafer pass a series of tests and dice failing a test are marked, e.g., by red ink dots.

The individual dice are now separated by sawing the wafer using a diamond saw and are sorted. The good dice are provided with a package, and the chip is connected to the inner leads of the package. This step is called *assembly*. After sealing the package and final *testing*, the chip is ready for shipping. The four steps transferring the wafer into the final chip are displayed in Figure 2.1.

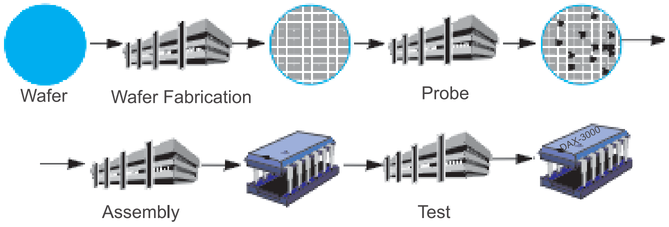


Figure 2.1: *Production Steps (Schömig and Fowler 2000)*

2.2 Wafer Fabrication

As already indicated, the focus in this monograph is on the application of cluster tools in the second step of the chip manufacturing process, the wafer fabrication process. This step is often termed as the *front end* part of semiconductor manufacturing, as opposed to the *back end*, consisting of testing and packaging.

2.2.1 Processing Steps

Independently of the variations of the semiconductors produced in today's fabs, such as the chip structure or the basic material, the wafer fabrication process generally consists of repeatedly applying four basic sub–steps:

1. Layering,

2. Patterning,

3. Doping,

4. Heat treatments.

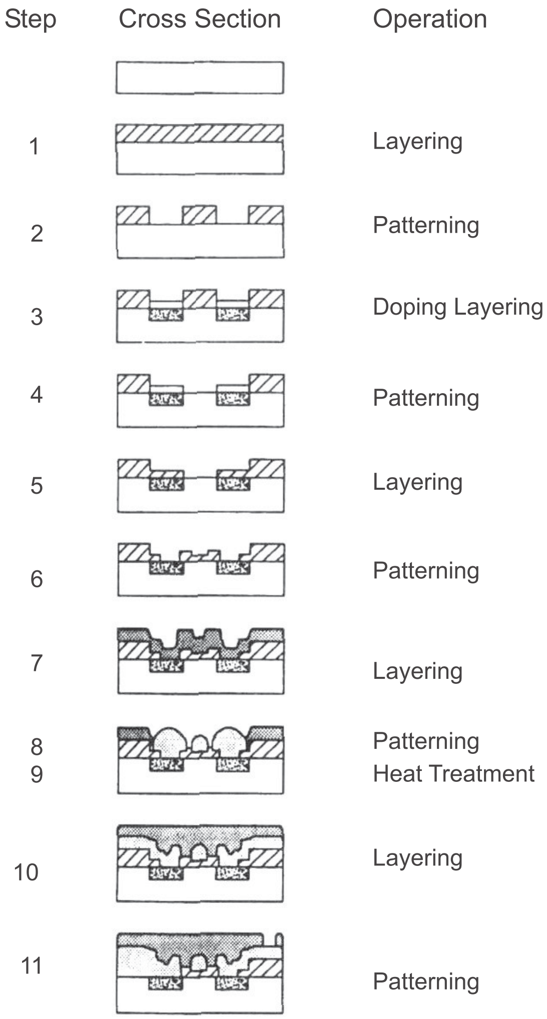


Figure 2.2: *Formation of Metal Gate MOS Transistor (Zant 1996)*

This sequence of chemical and physical processing steps is performed repeatedly to build the required chip structure on the silicon wafer. Figure 2.2 displays the steps required to build a metal gate Metal Oxide Semiconductor (MOS) transistor. The individual steps will now be described briefly.

Layering

During a layering step, thin layers of an insulating, semi–conducting or conducting material are added to the wafer surface (see steps 1, 5, 7, and 10 in Figure 2.2). The layers are added using *oxidation* or *deposition*. Oxidation is used to form a dielectric silicon dioxide layer on the semiconductor, for example to protect the surface from contamination. This operation is often performed in tube furnaces.

Deposition can be performed in the form of *chemical vapor deposition* (CVD), *evaporation*, and *sputtering*, in order to form various other layers of semiconductors, conductors or dielectrics.

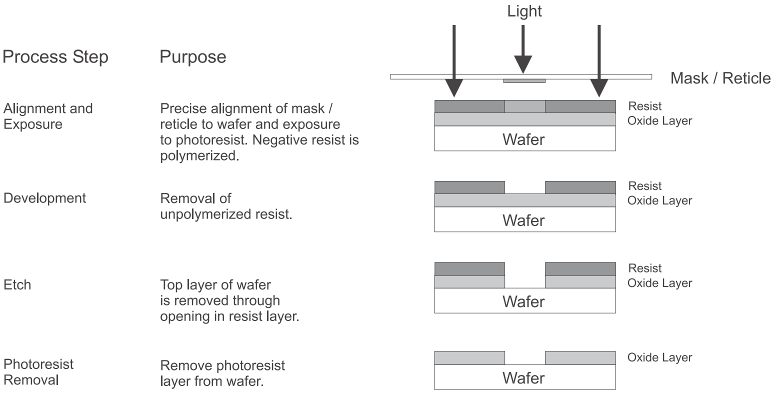
Evaporation is the oldest deposition method used in semiconductor manufacturing. It takes place in an evacuated chamber by heating the metal to be deposited to a temperature that allows atoms or molecules to escape from the solid evaporation source.

Sputtering, or *physical vapor deposition* (PVD) is also performed in vacuum chambers. Argon gas is ionized and accelerated towards the target material that is to be deposited. The energy of the argon atoms is enough to separate atoms or molecules from the target. The particles form a cloud within the chamber, and some of these particles will land on the wafer surface.

Chemical vapor deposition (CVD) is the most common deposition technique. CVD is performed in chambers with atmospheric pressure (APCVD) as well as at lower pressure (LPCVD). In the CVD chamber, the wafers are heated and a vapor containing the atoms or molecules required on the wafer surface is introduced into the chamber. After the required thickness of the layer is reached, the vapor is extracted again from the chamber.

Patterning

In the patterning steps (see steps 2, 4, 6, 8, and 11 in Figure 2.2), parts of the layer created during the previous layering step are removed to form the required geometrical structures, e.g., holes or islands, on the wafer surface. The patterning processes are usually termed *photolithography* or *masking* and are comparable to photography, however on a microscopic scale.



Process Step	Purpose
Alignment and Exposure	Precise alignment of mask / reticle to wafer and exposure to photoresist. Negative resist is polymerized.
Development	Removal of unpolymerized resist.
Etch	Top layer of wafer is removed through opening in resist layer.
Photoresist Removal	Remove photoresist layer from wafer.

Figure 2.3: *Pattern Transfer During Photolithography Step (Zant 1996)*

During photolithography, the desired horizontal structures are transferred from a photomask to the surface layer. This is done by first forming a light–sensitive layer of *photoresist* on the wafer surface. The surface is then exposed to light through the photomask (see step ”Alignment and Exposure”in Figure 2.3). The photoresist areas of the surface not exposed to light can be removed with chemical solvents whereas the portions of the photoresist that are exposed to light change their chemical conditions and become resistant to these chemicals (see step ”Development”in Figure 2.3).

Etchants now remove those parts of the wafer’s surface that are not protected by the photoresist (see step ”Etch”in Figure 2.3). The final step is to remove the photoresist from the wafer surface (see step ”Photoresist Removal”in Figure 2.3).

Doping

To give the wafer surface the desired electronic properties, either thermal diffusion or ion implantation is performed during the doping step. The goal is to create conductive regions and so called N–P junctions, i.e., separations between regions with a surplus of electrons (N–type) and regions with a surplus of holes (P–type).

In so–called diffusion tubes, the wafer, for example a P–type wafer, is exposed to a concentration of dopant atoms, e.g., N–type dopants. Those atoms diffuse into the holes in the wafer surface generated during the photolithography step, creating N–type islands within the P–type region.

Since structure sizes in integrated circuits become smaller and smaller, there is a limit on the applicability of the diffusion technique. Ion implantation facilitates doping of wafers with smaller feature sizes. With this technique, dopant atoms are shot on the wafer surface with high energy, enter beneath the surface and rest there, creating islands of the required conductivity.

Heat Treatment

The heat treatment step consists of heating the wafer to temperatures of about 500 to 1000 degrees Celsius. This treatment is necessary to repair disruptions of the crystal structure caused for example by ion implantation and is performed either by thermal techniques or using infrared radiation.

2.2.2 Manufacturing Environment

If the wafer is contaminated during a single processing step, e.g., due to a microscopic dust particle, a single die or the whole wafer may be rendered useless. With structure sizes on the wafers being $0.3\mu m$ or less in size, a human hair with a diameter of $100\mu m$ covers more than a hundred transistors. As a rule of thumb, particles that are 10 times smaller than the smallest structure on the wafer can cause defects on the wafer.

Four major types of contamination can be distinguished, cf. (Zant 1996):

- Particles
  Particle contamination can be caused by a human hair, a dust particle, a smoke particle, a fingerprint, etc.

- Metallic ions
  Most chemicals present in wafer fabs contain atoms of metals in ionic form, so–called *mobile ionic contaminants* (MICs). In semiconducting materials, these ions are highly mobile, causing the conductivity of the semiconductor to be modified in an undesired way.

- Chemicals
  Trace chemicals present in chemicals or water used during the wafer fabrication process can cause unwanted chemical effects on the wafer's surface.

- Bacteria
  Bacteria can contribute both to particle contamination and metallic ion contamination, if they are located on the wafer. They are introduced into the manufacturing environment, e.g., by exhaled air.

Due to the sensitivity of the wafer to particulate and molecular contamination, wafer fabrication takes place in a *clean room environment*. Access to the clean room is strictly controlled, only special clothing is allowed, and the air is constantly circulated and filtered. With these precautions, it is possible to achieve conditions where only a small number of particles remains in a cubic foot of air. Clean rooms are classified according to Table 2.1. As an example, for producing 64 Megabit chips, a clean room of class 0.1 is required.

Table 2.1: *Classification of Clean Rooms (Zant 1996)*

Class	Particle Size in μm	Number of Particles per ft^3 of Air
100,000	0.5 - 4	500 - 100,000
10,000	0.5 - 4	60 - 10,000
1,000	0.5 - 4	6 - 1,000
100	0.2 - 0.5	100 - 700
10	0.1 - 0.5	10 - 120
1	0.1 - 0.5	1 - 12
0.1	< 0.1	< 1

Two other important approaches for reducing the risk of contamination are creating *mini–environments* and increasing the amount of automation. In a mini–environment, i.e., a part of the fab physically separated from the surrounding, for example by metal walls, the environmental conditions can be more easily controlled and measured. Using automation, the amount of human intervention during the manufacturing process can be reduced. One way of implementing both approaches is the use of cluster tools, as will be explained later in this chapter.

Besides minimizing the contamination of the air, it is important to control the quality of the applied chemicals, the air temperature, and the humidity.

The measures required to maintain a clean room environment of the desired class make a wafer fab very expensive to maintain. The larger the floor space of the fab, the higher the cost for circulating and filtering the air. Therefore, one measure to reduce the cost of operating a fab is to reduce floor space, which again can be implemented by integrating several processing steps into a single machine.

2.2.3 Transportation

Since the processing steps presented in the previous sections require different machines and equipment, the wafers have to be transported between the individual steps. Usually, a set of 25 wafers is transported in a carrier or cassette. This set is called a *lot*. Depending on the number of layers and the layout of the fab, up to several hundreds of transportation activities of a wafer occur during the whole fabrication process. Figure 2.4 shows the layout of a wafer fab where the machines are grouped in different rooms according to the process step they perform (e.g., diffusion tools are placed in rooms ”Diffusion 1”, ”Diffusion 2”and ”Diffusion 3”). The arrows indicate the paths a wafer has to follow in the case of a very simple integrated circuit fabrication with only a small amount of layers.

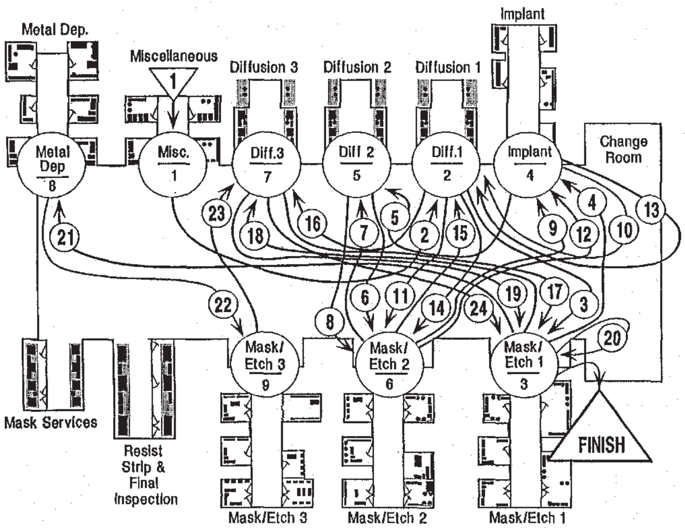


Figure 2.4: *Transportation of a Wafer During Processing (Atherton et al. 1990)*

In modern fabs, most of the transportation is performed by automated material handling systems. But even in the most recent fabs, the human operator still plays an important role, as the term *semiconductor manufacturing* (Latin ”manus”= ”hand”) implies. For example, several inspection activities can only be performed by humans, and some transportation between consecutive processing steps is done by manually carrying the wafers from machine A to machine B.

2.3 Cluster Tools

Cluster tools have gained significant importance in the fabrication of semiconductor chips during the last decade. Especially in modern 300mm fabs, i.e., facilities producing wafers with 300mm in diameter, they are an integral part of the production process. Besides their application in semiconductor manufacturing, cluster tools play an increasingly important role in the production of flat panel displays.

According to the SEMI E21–96 standard, a cluster tool is an ”... integrated, environmentally isolated manufacturing system consisting of process, transport, and cassette modules mechanically linked together”(SEMI 1996).

In a cluster tool, several processing steps required to produce a semiconductor chip are integrated into a single piece of equipment. The driving force behind that integration is the need to guarantee the performance of a process solution at a low cost–of–ownership, cf. (Singer 1993).

Cluster tools cover almost all types of processes in a wafer fab. Current applications include photolithography, etching, chemical and physical vapor deposition, cleaning, thermal processing, and photoresist strip.

As an example, Figure 2.5 shows a cluster tool configured to perform the processes required during a deposition step. Load Lock A is used to load a lot of wafers into the cluster tool. An aligner integrated into the load lock brings each wafer into a defined orientation before it is inserted into the first process chamber. The first process step is a cleaning step in chamber ”Clean / Degas”, followed by, if required, an etching step taking place in the chamber labeled with ”Soft Sputter”. Then, the actual deposition takes place in one of the three chambers ”CVD”, ”PVD (hot)”, or ”PVD (cold)”, according to the wafers’ recipe. The final process is *rapid thermal processing* (RTP), where the wafer is heated to a target temperature for a short period of time. The wafer is then transferred into Load Lock B where it cools down to environmental temperature.

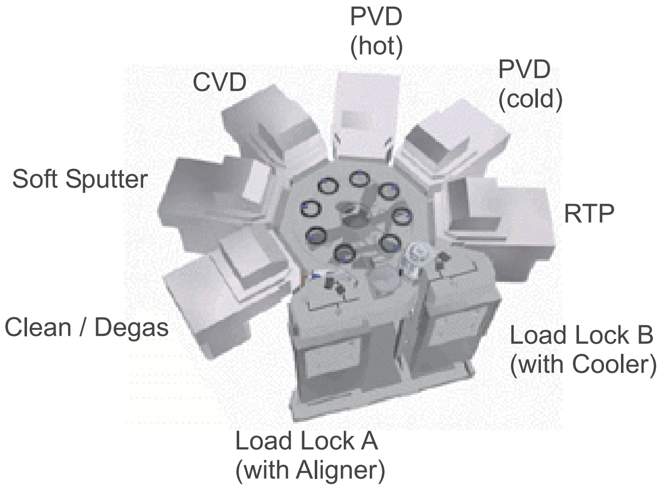


Figure 2.5: *Deposition Cluster Tool (Brooks Automation 2002)*

Since the early applications of cluster tools, a lot of effort has been invested in the standardization of the mechanical interfaces, cf. (SEMI 2002a) and (SEMI 2002b), and the communication between the individual components of a cluster tool, cf. (SEMI 2002c). This has led to the possibility of a *best–of–breed* approach when configuring a cluster tool for a specific application: it is possible to compose a cluster tool of components from different vendors and to choose the process module that provides the best performance for the required operation, cf. (Bader et al. 1990) and (Huntley 1990).

Although the communication among the cluster components follows non–proprietary standards, a lot of the know–how of the vendors, especially in the area of scheduling the cluster tool operations, is still proprietary and kept secret. Thus, optimization and adaptation of the logical control of a cluster tool is often impractical for the user or can only be achieved in close cooperation with the hardware vendor.

2.3.1 Benefits of Cluster Tools

Using cluster tools in the manufacturing process has several advantages as compared to non–integrated tools, cf. (Singer 1995). Basically, by combining sensitive processing steps and ensuring increased control of the processing conditions within the cluster tool, yield and process performance is significantly improved.

Since the processing steps and the respective process chambers are encapsulated in a ”mini–environment”within the wafer fab, it is easier to control the physical conditions in the cluster tool. Therefore, the probability of contamination is reduced and some of the causes for defects can be eliminated, which is of special importance in multi–step procedures requiring different chemical and physical processing steps.

By integrating metrology equipment, e.g., microscopes, into a cluster tool, it is possible to repeat specific processing steps if the measurement indicates that the process result is not within specification. This kind of loop can then be performed without human intervention and without removing the wafer from the cluster tool.

As indicated earlier, providing the clean room environment necessary to produce integrated circuits requires cost–intensive precautions. Through the integration of several processing steps together with the transportation equipment into a single piece of equipment, the expensive clean room space required to perform a sequence of processing steps can be significantly reduced and the productivity per square foot of fab space is increased.

A cluster tool can be regarded as a ”mini–fab”within the fab, having its own transportation system and controller. Whereas in a non–integrated environment, different machines have to compete for shared resources like operators and transportation systems, in a cluster tool the resources and the sequence of process steps is regulated by the cluster tool controller. Thus, it is possible to reduce the transportation and waiting times between processing steps to a minimum and to guarantee a certain maximum time for performing a sequence of steps.

From economical point of view, an important property of this kind of equipment is that the modularity allows to use a cluster tool to produce a series of generations of end–products. If a new generation of products requires modified processing equipment, it is possible to exchange only the respective process chambers while still using the mainframe, transportation, and remaining process chambers of the cluster tool. When using cluster tools with redundant process chambers, operations can still be continued if one of the redundant chambers fails.

Finally, the need for human intervention is reduced to a minimum. If at all, an operator is usually only required to load a lot of wafers into the cluster tool's load lock.

2.3.2 Cluster Tool Components

In the following paragraphs, the individual components of a cluster tool are presented, their functions are defined, and the interfaces to other components are described.

Mainframe

The mainframe (sometimes also called *transport module*) is the central component of the cluster tool. It contains the transport mechanisms that move the wafers from the load locks to the individual process chambers. Figures 2.6 and 2.7 display two types of mainframes.

Basically, we can distinguish between mainframes with *linear* and *radial* layout. The layout defines the topology of the transport mechanisms. Two simple layouts can be seen in Figure 2.8.

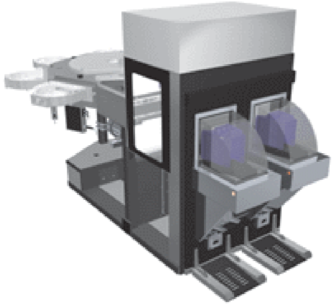


Figure 2.6: *Mainframe Type 1 (Brooks Automation 2002)*

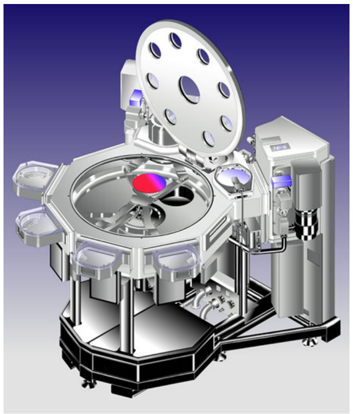


Figure 2.7: *Mainframe Type 2 (Brooks Automation 2002)*

Typically, cluster tools are built around one mainframe, but more complex types with two or even more mainframes are available as well, like the one in Figure 2.9.

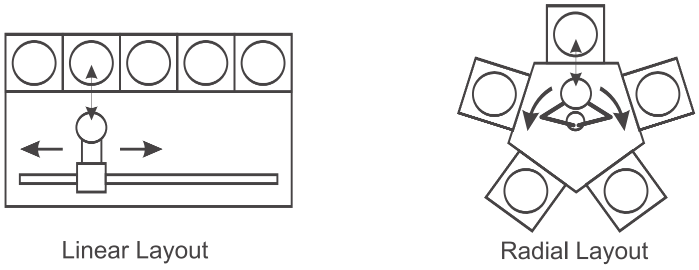


Figure 2.8: *Simple Types of Cluster Tools*

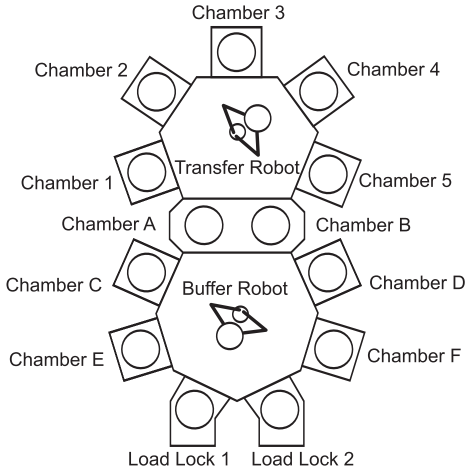


Figure 2.9: *Cluster Tool with Two Mainframes*

Process Chambers

The *process chambers* or *process modules* responsible for performing the actual material processing are attached to the mainframe.

The different functions that can be implemented in a process chamber are chemical and physical processes, inspection, and pre– or post–processing. The processes for which process chambers are available include CVD, PVD, etch, sputtering, photolithography, metrology, but also steps that prepare the wafer for processing like preheating, aligning, cool down, and degas. It is also possible to use a process chamber as a ”parking”station for wafers if they need to wait for a resource to become available.

Commonly, only single wafers are processed in a process chamber. But for some processes, there exist chambers that can perform *batch processing*, i.e., they can process more than one wafer in parallel. Also available are *index modules* that perform a sequence of process steps. Figure 2.10 shows the mentioned chamber types attached to a mainframe.

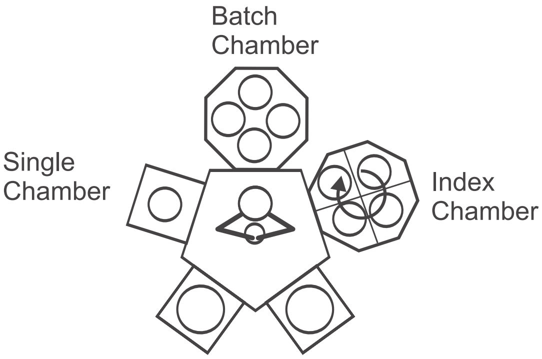


Figure 2.10: *Chamber Types*

Depending on the kind of process performed in a process chamber, the chamber is also responsible for generating the environmental conditions. Some processes, for example, require a vacuum (like etching and CVD), others, like photolithography, can be performed under non–vacuum conditions. If the process environment differs from the conditions in the mainframe, the chamber will also adapt the chamber conditions if a wafers is loaded into or from the chamber.

Figure 2.11 displays the individual phases that occur during the processing of a wafer in a process chamber, together with examples for the respective times.

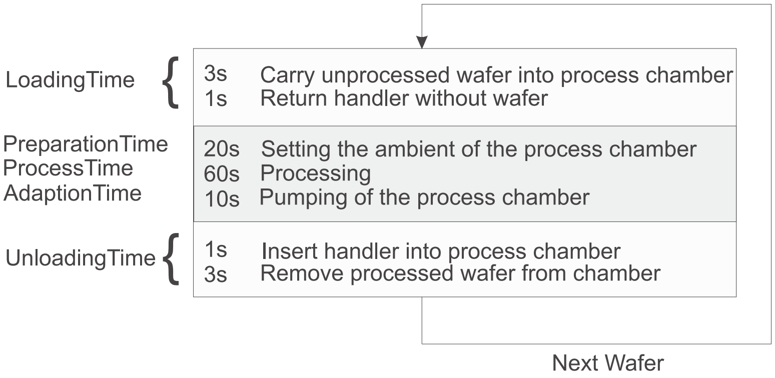

LoadingTime
3s Carry unprocessed wafer into process chamber
1s Return handler without wafer
PreparationTime
ProcessTime
AdaptionTime
20s Setting the ambient of the process chamber
60s Processing
10s Pumping of the process chamber
UnloadingTime
1s Insert handler into process chamber
3s Remove processed wafer from chamber
Next Wafer

Figure 2.11: *Process Steps and Times in a Process Chamber (Kawamura et al. 1998)*

Load Locks

The *load locks* (also known as *product storage* or *cassette module*) are the interfaces of the cluster tool to the manufacturing floor. A cluster tool has at least one load lock.

A lot of wafers that has to be processed in a cluster tool is placed into the load lock by an operator or by the material handling system. The load lock is then isolated from the outside environment by closing the external door and the required conditions (for example, vacuum) are generated; in some cases, the wafers are taken from the cassette and are placed into slots within the load lock, cf. (Paré et al. 2002). Then, the internal door to the mainframe is opened and the transport system can start picking wafers from the load lock and transferring them into the process chambers.

In some cluster tool types, there exists one load lock where wafers are taken from and loaded into the cluster tool and another one where the finished wafers are placed. In any case, the load lock has to be adjusted to the outside conditions, e.g., by venting the load lock, before the lot can be taken from the load lock.

The time required to adapt the load lock environment to the conditions within the mainframe is called *pump time*. The time it takes to adjust the load lock to the conditions outside of the cluster tool is called *vent time*.

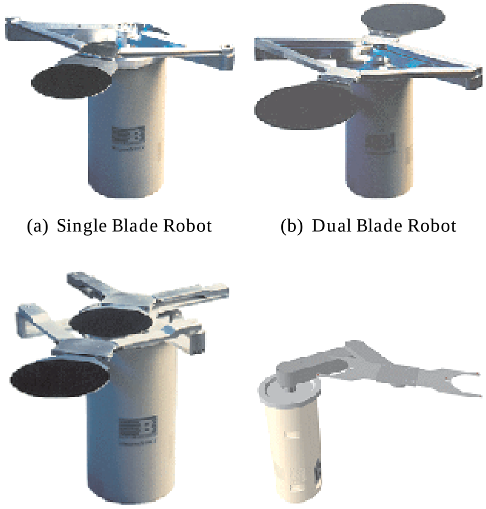
Handlers

The usual transport mechanism responsible for moving the wafers within the cluster tool to the individual process stations is a robot, called *handler*. At least one handler is located in a mainframe, but configurations of cluster tools with more than one robot per mainframe do also exist. The robot is used to move the wafers from the load lock to the process chambers, between process chambers, and back to the load lock.

As already mentioned in the paragraph on mainframes, there exist *radial* and *linear* handler topologies, cf. Figure 2.8. Furthermore, handlers are distinguished by the number of wafers they can carry at a time. Figure 2.12(a) shows a handler of the type *single–blade* (or *single–effector*), that can move only one wafer at a time.

Figures 2.12(b) and 2.12(c) display handlers of the *dual–blade* type. They are able to perform a *switch operation*: One blade can be used to transport a wafer to a chamber where another wafer is waiting to be picked up after a process step. The second blade is used to take the waiting wafer from the chamber. The first blade is then immediately used to place the new wafer into the chamber. Using this switch operation, the idle time of the chamber can be reduced compared to a single–blade robot. Whereas the first three handler types presented can move along two axes (X and Y axis), the handler in Figure 2.12(d) can also move along the Z axis.

There exist further, more specialized types of wafers, like dual–blade handlers that have a left and right arm that can be extended in parallel.



(a) Single Blade Robot (b) Dual Blade Robot

(c) Leap Frog Robot (d) Robot Arm

Figure 2.12: *Robot Types (Brooks Automation 2002)*

In this way, two chambers can be loaded at the same time, cf. (Paré et al. 2002).

Depending on the way the wafer is placed on the blade, the move time of the handler can change if the handler is loaded. For example, there exist blade types where that place the wafer on the blade only loosely. Therefore, the robot can not move as fast when loaded, because the wafer could fall from the blade during circular motions.

Cluster Tool Controller

The *cluster tool controller* is the centralized module responsible for the control of all activities within the cluster tool.

Depending on the cluster tool architecture, the cluster tool controller can be a program run on a processor integrated into the cluster tool or a software installed on an external PC or workstation, connected to the cluster tool via TCP / IP.[1] The way the cluster tool controller communicates with the individual modules is standardized, cf. (SEMI 2002c).

The tasks performed by the cluster tool controller include:

- Coordination and scheduling of the material flow from module to module (see Chapter 4 for more details on the scheduling task).
- Providing process modules with recipe data required for the material processing.
- Control of the hand–off of wafers between chambers and load locks.
- Coordination of the processes required for adapting the load locks and process chambers to the environmental or processing conditions.
- Alarm functions for the operator, if a wafer falls off the handler, if a deadlock occurs, etc.
- Acquiring data, generating and displaying statistics (e.g., average cycle time, down times, etc.).
- Animation of the cluster tool activities, since usually the handler movements in the cluster tool and the progress in processing chambers are not visible from the outside.

Recipes

A *recipe* is a set of instructions, describing the sequence and parameters of the process steps that a wafer has to follow in a cluster tool. Each lot or wafer contains a label that contains a reference to the recipe for this

[1] Transmission Control Protocol / Internet Protocol, the two most common protocols used in commercial and private communication networks, cf. (Tanenbaum 2002)

specific lot or wafer. The recipes are stored in the cluster tool controller or in an external database, where they can be retrieved from upon request.

Among the data stored in a recipe are the kind of chemicals required for processing and process parameters like process duration, temperature, chemical or gas level, vacuum levels, voltage, energy levels, etc. Usually this data is stored in two levels: The top level is the *cluster recipe* with information which chambers have to be used. For each chamber, a *chamber recipe* on the second level contains the process parameters for this specific chamber.

2.3.3 Cluster Tool Configuration and Modes of Operation

The basic configurations of a cluster tool are *serial* and *parallel* configuration. In a serial cluster tool, all process chambers perform a different process step, whereas in a parallel cluster tool all process chambers are identical. More common are *hybrid* configurations, where some of the process chambers are duplicated (usually those with the longest process times).

Two basic operation modes exist for cluster tools: *single mode* and *parallel mode*.[2] In single mode, a cluster tool processes the wafers of only one lot at a time. If all wafers are finished processing, the lot can be taken from the load lock, a new lot is loaded and the cluster tool starts processing the wafers from the new lot. If the cluster tool is equipped with only one load lock, this is the only mode of operation possible.

If more than one load lock is attached to the cluster tool, parallel operation is also possible. In this mode, while the cluster tool is processing wafers of lot A placed in load lock A, another lot B can be loaded into load lock B and after the pump time, the wafers of lot B can be processed in parallel with the wafers of lot A. In this way, the cluster tool can still process lots during the pump time necessary to generate the required conditions

[2] It might be confusing that the term *parallel* is used in the configuration context as well as in the context of operations modes, but unfortunately in the literature both uses can be found.

in the load lock. This ability to begin processing one lot while simultaneously processing the previous one is also called *cascading*.

In parallel mode, the impact of processing wafers of different recipes on cluster tool performance can depend on the time difference between the moment when the individual lots become available for processing, cf. (Niedermayer 2002). A special case of parallel mode is called *sync mode*, where the start of processing wafers of the lots in the load locks is synchronized and, therefore, effects of varying start delays are avoided.

Cluster tools that process wafers in a pre–defined sequence, independently of the recipe mix being processed in parallel, are called *fixed sequence cluster tools*. If, however, the cluster tool control system can spontaneously decide on how to allocate the available resources, depending on the current state of the resources and the current requests, the cluster tool is called a *flexible sequence cluster tool*. The problem of finding the optimal sequence in such tools is addressed in Chapter 4.

3 Performance Modeling of Cluster Tools

Since cluster tools integrate a series of processing steps into one machine and are able to process several wafers in parallel with wafers sharing resources, they represent a very complex and sophisticated class of machines. Even minor changes to the cluster tool configuration (e.g., of the processing times or the scheduling strategies) may have significant effects on the cluster tool performance.

The reasons for creating an analytic or simulation model of cluster tools are manifold. These models facilitate the performance assessment, allowing, for example, to compare existing cluster tool designs with alternatives, to anticipate the impact of changes in process times on cluster tool performance, to evaluate scheduling techniques, etc.

Important performance characteristics considered in this context are the number of wafers or lots the cluster tool can process per time unit, called the *throughput*, and the cycle time. The *cycle time* of a lot is the time between the moment when the lot is placed into one of the cluster tool's load locks until it is removed from the load lock, with all wafers in the lot being processed. The cycle time of a wafer is defined as the time between the moment when the wafer is picked from the load lock by a robot until the moment when it is finished processing in the cluster tool and returned to the load lock.

Using a performance model of a cluster tool as a decision support tool before purchasing new equipment can protect the buyer from false investments. For example, comparing different alternatives can help in deciding which is the best cluster tool configuration to buy. Often cluster tool vendors offer such models to assist the customer in this decision. In most of the cases, however, it is not possible to compare the performance of tools from different vendors using such models, because the modeling assumptions and methods differ significantly.

Even though the scheduling and controlling techniques applied in modern cluster tool types are very sophisticated, the appearance of deadlocks is still a frequent problem. Usually, a deadlock can only be resolved by stopping the cluster tool operations, opening the cluster tool, and manually removing one or more of the wafers that caused the deadlock. This often leads to defective wafers, since running processes are affected, and the availability of the cluster tool is reduced. Therefore, another application for models of cluster tools is the development and testing of new scheduling techniques in order to reduce the probability of deadlocks.

In principal, two approaches to modeling cluster tools can be distinguished: *analytical models* and *simulation models*, cf. Figure 3.1.

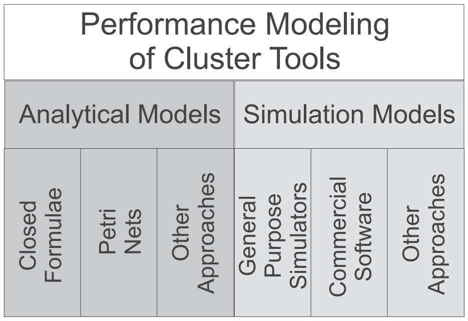


Figure 3.1: *Cluster Tool Performance Modeling Approaches*

Analytical models include models using closed formulae, formal models such as *Petri Nets*, and other mathematical approaches. The simulation models presented in this chapter are divided into models created using general purpose simulators such as AutoMod, cf. (Brooks Automation 2002), cluster–tool–specific simulation environments and other, less prevalent approaches, e.g., meta–modeling.

Publications on modeling cluster tools mostly focus on the presentation of a single modeling approach, without comparing it to alternatives. To the author's knowledge, no comprehensive overview of the literature is available yet. In this chapter, we give a review of these publications, categorize the publications cited, and give a short evaluation of the presented methods.

In addition, the approaches are compared in different aspects, such as the effort necessary to create and maintain the models, their ability to predict essential performance measures, and their usability in semiconductor manufacturing planning and control. As an example for a simulation approach to the modeling of cluster tools, the simulation model used to perform the studies presented in the following two chapters is introduced and discussed.

3.1 Literature Review

3.1.1 Analytical Models

Closed Formulae

Closed Formulae approaches have the appealing property of being able to quickly produce the desired performance metrics, because the formulae can easily be entered in a simple program or a spread sheet. However, this approach has limitations concerning the range and complexity of cluster tool types that can be modeled and the flexibility of the resulting models in terms of changes in configuration, etc. Nevertheless, in a variety of applications in daily practice, closed formulae models can provide a very good estimate of the desired performance metrics with little effort.

In (Perkinson et al. 1994) an analytical approach for computing the time required to process a lot of wafers is presented. The relationship between process time of the chambers, transport time of the robots and maximum throughput of the cluster tool is analyzed.

In this approach, a distinction is made between *transport–bound* and *process–bound* schedules. In a transport–bound schedule, the handler is always busy and, therefore, determines the cycle time of a wafer. On the other hand, in a process–bound schedule, the throughput of the cluster tool is impacted by both the transport times of the handler and the process times of the process chambers.

The authors’ analytic method is restricted to rather simple models of cluster tools. For example, process times are supposed to be identical in all chambers. Furthermore, the approach can only be applied to cluster tools in sequential mode with one single–blade robot. In detail, the cluster tool investigated has the following properties:

- N_C process chambers,
- a single load lock with load time T_{load} and unload time T_{unload},
- a single handler with constant and equal move time T_{move} between all chambers and the load lock,
- constant and equal process time T_{proc} for all chambers,
- equal recipe for all N_W wafers in a lot,
- each wafer has to be processed in each of the chambers.

For $N_C = 3$, the cluster tool is depicted in Figure 3.2.

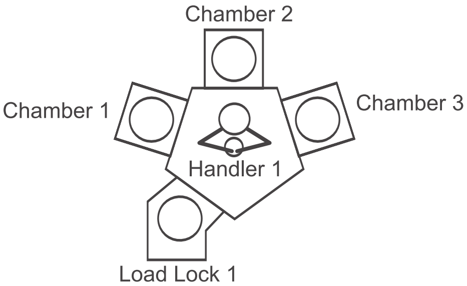


Figure 3.2: *Cluster Tool Investigated by Perkinson et al. (1994)*

The authors derive the following expression for the total cycle time T_L of a lot of wafers:

$$\begin{aligned} T_L &= N_W \cdot T_{FP} + T_{load} + T_{trans} + \\ &\quad T_{unload} - (N_C - 1) \cdot T_{FP} \, , \end{aligned} \tag{3.1}$$

where T_{FP} is the *fundamental period* of the cluster tool, i.e., the time between subsequent arrivals of completed wafers at the load lock. It can be expressed as

$$T_{FP} \quad = \quad \begin{cases} 2T_{move} \cdot (N_C + 1)\,, & \text{if} \quad \frac{T_{proc}}{T_{move}} \leq 2(N_C - 1)\,, \\ T_{proc} + 4T_{move}\,, & \text{else}\,. \end{cases} \tag{3.2}$$

To derive T_{trans}, we define

$$Z \quad = \quad \min\left(N_C - 1, \text{int}\left(\frac{T_{proc}/T_{move} + 2}{2}\right)\right)\,. \tag{3.3}$$

For $Z \neq N_C - 1$, we obtain

$$T_{trans} \quad = \quad 2Z(T_{proc} + 3T_{move}) - 2T_{move}(Z+1)^2 - T_{move}\,, \tag{3.4}$$

whereas for $Z = N_C - 1$

$$T_{trans} \quad = \quad 2(N_C - 1)(T_{proc} + 4T_{move}) - T_{move}(2N_C + 1)\,. \tag{3.5}$$

This basic model is extended in (Perkinson and Gyurcsik 1996) by incorporating redundant chambers, revisiting of chambers and using load locks as buffers for wafers. The resulting model is still a closed formula that facilitates easy application in performance computations.

Venkatesh et al. (1997) use the same distinction between transport–bound and process–bound schedules to compute the steady state throughput of a cluster tool with a dual–blade robot (see Section 4.1.2 for an explanation of steady state and transient state). It is assumed that the wafers have to be processed sequentially in the N_C process chambers. Process time in chamber $i, i = 1, \ldots, N_C$ is p_i. All wafers have the same recipe and there are no redundant chambers. The throughput of the cluster tool during the transient phase is not considered in the paper. As an application, the authors compare the steady–state throughput of a single–blade robot to that of a dual–blade robot using the derived formulae.

In (Wood et al. 1994) the throughput time T_{TP} of a cluster tool with a single handler is derived from the fixed throughput time T_{fix} that is independent of the lot size, an incremental throughput time T_{inc} for every wafer processed, the number of wafers N_W, and a correction term c as follows:

$$T_{TP} \quad = \quad T_{fix} + N_W T_{inc} + c\,. \tag{3.6}$$

The correction term c has to be introduced since for some tool configurations throughput time is not a linear function of lot size. Eqn. (3.6) implies that the throughput rate of a cluster tool of the investigated type can never exceed $1/T_{inc}$.

Like Perkinson et al. (1994), the authors distinguish cluster tools operated in process–bound and in transport–bound mode. The formulae of the throughput time and throughput rate are derived for cluster tools in serial as well as in parallel configuration. Processing times in the individual chambers can differ in serial configuration in their model, only the processing time of the *bottleneck* chamber, i.e., the chamber with the longest processing time needs to be known.

For the throughput rate r_{TP}, the authors derive the following formula:

$$r_{TP} = \min\left(\frac{N_L l}{T_{fix} + N_W T_{inc}}, \frac{1}{T_{inc}}\right)\ , \tag{3.7}$$

where N_L is the number of load locks of the cluster tool.

As applications, the authors show a performance prediction of a cluster tool based on their formulae as well as a cost comparison of alternative loading methods.

Similarly, Wood (1996) uses these formulae to implement a cost model and to compare serial and parallel configuration of simple cluster tools. In addition, the corresponding formulae are derived for a system consisting of n identical cluster tools with n chambers each, in serial as well as in parallel configuration.

This study is extended in (Lopez and Wood 1996) and (Lopez and Wood 1998), where again serial and parallel configurations are compared and optimal lot sizes and lot release policies are determined for systems consisting of n identical cluster tools with n chambers each. Furthermore, in (Lopez and Wood 1996) these systems are examined under the effect of scheduled maintenance.

Another way to derive the throughput of a cluster tool in closed form is to perform a *bottleneck analysis* of the tool by identifying the module of the cluster tool that limits tool performance. If the bottleneck module has been identified, it is sufficient to assess the throughput of this module; the other components of the cluster tool can be neglected. 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. In this case, the bottleneck module can change depending on the current recipe mix and the timing of the cluster tool processes. Therefore, identifying the bottleneck is very difficult or not possible at all.

Different approaches have been presented for assessing the *Overall Equipment Effectiveness* (OEE) of cluster tools, or, to be more precise, to adapt the SEMATECH definitions of OEE to cluster tools, cf. (Ames et al. 1995). OEE is defined as follows (the variable names are adopted from the definition found in the SEMATECH standard):

$$\begin{aligned} OEE &= (\text{Availability}) \cdot (\text{Rate Efficiency}) \cdot \\ &\quad (\text{Operational Efficiency}) \cdot (\text{Rate of Quality})\,, \end{aligned} \tag{3.8}$$

where the individual factors are defined as

$$
\begin{aligned}
(\text{Availability}) &= \frac{(\text{Total Time}) - (\text{Downtime})}{(\text{Total Time})}\,, &\quad (3.9)\\
(\text{Rate Efficiency}) &= \frac{(\text{Ideal Cycle Time})}{(\text{Actual Cycle Time})}\,, &\quad (3.10)\\
(\text{Oper. Efficiency}) &= \frac{(\text{Total Productive State Time})}{(\text{Equipment Operational Uptime})}\,, &\quad (3.11)\\
(\text{Rate of Quality}) &= 1 - \frac{(\text{Rejected Wafers})}{(\text{Total Wafers Processed})}\,. &\quad (3.12)
\end{aligned}
$$

OEE is applied, for example, in the planning process as a measure of how many parts a certain machine can actually produce per time unit, taking into account all factors that degrade the machine's theoretical throughput. Therefore, it is essential that the OEE value reflects the actual effectiveness of the machine as precisely as possible. If the calculated OEE of the machine is higher than the actual effectiveness, the production plan might lead to a utilization of the machine that is higher than expected, causing large waiting times of the parts to be processed on that machine. On the other hand, if the OEE value predicts a productivity lower than what the machine actually can achieve, the machine might be under–utilized, and productivity is partly lost.

As already mentioned, cluster tools can be configured in three ways: Serial configuration, with each chamber performing a different processing step; parallel configuration, where the cluster tool consists of identical process chambers, each performing the same processing step; and hybrid configuration, similar to serial, however, some of the processing chambers are duplicated so that the respective processing step can be performed in parallel in more than one chamber.

The type of configuration has significant influence on the availability of the cluster tool in case of a chamber downtime. For a serial cluster tool, an outage of a single chamber makes the whole cluster tool unavailable, whereas in parallel configuration, the cluster tool can continue operation if one chamber fails, but with lower throughput. In case of a hybrid cluster tool, the tool can still be used if the downtime occurs at a redundant chamber. Otherwise, the tool becomes unavailable. Matters become even more complicated if a cluster tool is able to process wafers of different recipes. Then, it might be that a chamber downtime affects the availability of the cluster tool for one specific recipe, while for another recipe the chamber outage has no effect. Similarly, in the case of cluster tools with multiple robots per mainframe a downtime of one of the robots does not make the cluster tool unusable.

It is obvious, that it is not simple to define the availability of a cluster tool. This problem has been approached by several authors. The *weighted configuration matrix* method presented by Wang and Christian (1998) and Dolman et al. (1999) allows assigning weights to the different fault scenarios of a cluster tool. For each scenario, the weight of the scenario determines the impact of the equipment failure on cluster tool throughput. As an example, Table 3.1 shows one possible instantiation of a weighted configuration matrix for a cluster tool with one robot, two load locks ("LLA"and "LLB") and two identical process chambers ("ChA"and "ChB").

Table 3.1: *Weighted Configuration Matrix (Wang and Christian 1998)*

Scenario	LLA	LLB	ChA	ChB	Weight
1	0	0	X	X	0
2	X	X	0	0	0
3	0	1	0	1	0.55
4	0	1	1	0	0.55
5	0	1	1	1	0.6
6	1	0	0	1	0.55
7	1	0	1	0	0.55
8	1	0	1	1	0.6
9	1	1	0	1	0.7
10	1	1	1	0	0.7
11	1	1	1	1	1

In columns two to five of this table, an entry of "0"means that the respective module is down, whereas an entry of ”1”means that the module is up. An ”X”means that the respective module can either be up or down, because it does not have an impact on the weight of the respective scenario. The weight determines how the throughput is affected in a certain fault scenario. For example, if one load lock is down (Scenario 5), the throughput is degraded by 40 percent.

In (Dhudshia and Clyde 1996), the authors emphasize the necessity of achieving a common standard for defining OEE, reliability and Cost of Ownership for cluster tools, but no attempts for defining these metrics are made.

Busing and Leachman (1998) map the problem of defining OEE for flexible–sequence cluster tools on the problem of defining the theoretical processing time. They identify five factors that affect the operations sequences: cluster configuration, cluster sequencing logic, recipe mix, alternative resources, and lot sequencing effects. Since these factors make modeling the performance of cluster tools very complex, the authors propose two approaches for deriving OEE metrics for cluster tools: one approach is to limit the point of view to a ”virtual machine”consisting of only a single process chamber and the associated transport module and to derive the theoretical processing time for this virtual machine.

The other approach is to use restrictive assumptions to come up with an acceptable approximation of the theoretical processing time of the whole cluster tool. To this end, they present four strategies: actual operations sequence, optimal operations sequence based on actual lot arrival times, optimal operations sequence disregarding lot arrival times, and average of chamber theoretical processing times. The authors suggest using the latter approach, i.e., to compute the theoretical processing time for each cluster tool chamber by applying the ”virtual machine”approach and using the average of these times as an approximation of the cluster tool theoretical processing time.

Furthermore, the authors present definitions of productivity metrics like *uptime* and *productive time* for individual chambers and complete cluster tools.

Petri Nets

Performance analysis of complex cluster tools, consisting, e.g., of multiple robots, is not straightforward and cannot be accomplished using simple closed formulae approaches. 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, *Petri nets*, as a more powerful modeling technique, have been applied to the performance modeling of cluster tools.

A Petri net is a graphical and mathematical modeling tool that can be applied to model systems with concurrency. Petri nets were introduced by Carl Adam Petri in the beginning of the 1960's as the first theory for discrete parallel systems. They are a generalization of automata theory. Petri nets can on the one hand be used as a graphical tool to display concurrent systems. On the other hand, the mathematical theory related to Petri nets allows to compute state equations and probabilities and to derive important properties for the system under consideration. For example, they can be used to examine whether deadlocks can occur in a concurrent system, cf. Chapter 4.

For an introduction to Petri nets and a list of publications on this topic see (Petri Nets Steering Committee 2003). In order to get an overview of the application of Petri nets in the modeling of manufacturing systems see, for example, (Leventopoulos 1994).

One of the first attempts to model cluster tools using timed Petri nets is reported in (Srinivasan 1998). In this article, the author shows in detail the construction of a Petri net model of a specific cluster tool configuration. As examples, Petri nets are presented for a cluster tool with two chambers and a single–blade handler in serial configuration as well as for a tool with three chambers and a dual–blade robot in parallel configuration. However, the presented approach is applicable to various other cluster tool configurations as well. Using state cycle and reachability analysis, the transient phase and steady state cycles are derived from the Petri net. The author finally derives the throughput for the mentioned cluster tool configurations.

Shin and Lee (1999) also use timed Petri nets to model simple single–robot cluster tools and to compute the cycle time for these tools. They restrict their approach to the analysis of one–wafer cycles. These are cyclic sequences during which one wafer enters the cluster tool and one finished wafer exits the tool. The results are formulae for the cluster tool's cycle time in the process–bound and in the transport–bound configuration, similar to the closed formulae derived by Perkinson et al. (1994). The authors also present a predictive control method that increases the performance of a cluster tool, namely by placing the robot in front of the chamber that finishes processing next, and compare this method to a non–predictive scheduling method using their presented Petri net model.

In his first article in a series of publications on cluster tool modeling using timed Petri nets, Zuberek (2000) models a simple cluster tool with multiple chambers, a single robot and one load lock operating in serial configuration. The Petri net is used to derive the steady–state as well as the initial and final transient behavior of the tool and to compute the duration of these phases in symbolic form. This basic model is extended in later papers to assess the cycle time of cluster tools with chamber revisiting, cf. (Zuberek 2001a), multiple robots, cf. (Zuberek 2001c), and dual–blade robots as well as multiple load locks respectively chambers, cf. (Zuberek 2001b).

Other Analytical Approaches

Besides closed formulae and Petri nets, other formal and mathematical methods can be used to model cluster tools. Niedermayer (2002) presents several approaches to predict lot cycle times. For cluster tools in single mode, linear and non–linear approximations are compared that derive the cycle time for different lot sizes if the cycle time for a given recipe with a given lot size is known. Different predictors are developed and compared that estimate the cycle time of lots for cluster tools in parallel and batch mode, as well as the effects of varying start delays that occur when processing two lots in parallel. The approaches include simulation, Neural Networks, and linear and non–linear formulae.

Shin et al. (2000) use *finite state machines* (FSM) to model the individual components of a cluster tool and specify the interaction between the FSMs. This model is used to implement and test a real–time embedded cluster tool scheduler for complex cluster tools; performance characteristics are not derived using this model.

3.1.2 Simulation Models

According to Shannon (1975), digital computer simulation is the process of designing a model of a real system and conducting experiments with this model on a digital computer for a specific purpose of experimentation. In this monograph, the focus is on *discrete–event simulation models*. They describe the system under investigation in terms of logical relationships and state changes of the elements at certain points in time. For a deeper introduction into simulation methodology, see for example (Law and Kelton 1991).

General Purpose Simulators

One of the common approaches of simulating cluster tools is to use a general purpose simulation environment, i.e., a software that allows to model a wide variety of technical systems and that is not focused on a specific field of application. Such environments usually provide the user with a tool set of generic components that allow to compose a model of a technical system.

Pierce and Drevna (1992) present a generic simulation model of cluster tools that was developed by SEMATECH, cf. (SEMATECH, Inc. 2002), using the general purpose simulation language *SIMAN IV*.[1] The robot movement calculations were implemented in FORTRAN. The simulation, equipped with an optional real–time animation of the cluster tool model, can be used for capacity, cost, and performance prediction. Operators performing tasks like transporting lots and system maintenance as well as

[1] SIMAN is the simulation language that is the basis of Rockwell Software's Arena family of simulation software products, cf. (Rockwell Software Inc. 2002) for details.

system failures are modeled. Process chambers can be of the type single wafer, batch, or index.

Another animated cluster tool model is presented by LeBaron and Pool (1994). The authors use *AutoMod* (cf. (Brooks Automation 2002) for details on AutoMod) to build a model of a two–robot cluster tool similar to the type depicted in Figure 2.9. Model parameters are entered using spreadsheets. Some of these parameters are explained in the publication, and the model output like tool throughput and module utilization is presented.

In (Mauer and Schelasin 1993) and (Mauer and Schelasin 1994), the general purpose simulation software *ProModelPC* (ProModel Solutions 2002) has been used to build simulation models of a plasma etch tool with radial handler movement, a photolithography tool with a central multi–axis robot, and a wet bench with linear handler movement. Though the structure of these tools differs significantly in terms of tool layout and handler movements, the authors report that the respective models are capable of simulating these tools with appropriate accuracy.

Another cluster tool simulation model based on the ProModel factory simulation software is presented in (Hendrickson 1997). The model is applicable for tools with one to eight process chambers, one or two load locks and a central handler. The algorithms used to schedule the robot movements are based on the transport module control software developed by Brooks Automation. The model is applied in several studies to predict throughput under different tool configurations, e.g., for a comparison of a single–blade and a dual–blade robot and to study the effect of adding process chambers to the tool.

Unfortunately, none of the publications mentioned in this section presents details on the implementation of the simulation model, especially no information on the scheduling algorithms is given.

Cluster Tool Simulation Software

In terms of commercial software for simulation of cluster tools mentioned in scientific publications, *ToolSim* (Brooks Automation 2002) seems to be the most popular one. ToolSim and its predecessor *ClusterSim* are simulation software packages for the modeling of cluster tool equipment, with a focus on capacity analysis of cluster tools. ToolSim facilitates 3–D animation of the simulation model. The simulation models already available for ToolSim include deposition tools, PVD and CVD tools, lithography equipment, steppers, etc. ToolSim is based on AutoSimulation's general purpose simulation software *AutoMod*.

The data defining the cluster tool configuration and simulation parameters are entered via a number of input files. The cluster tool model can be composed of pre–defined module templates like processing chambers and load locks.

LeBaron and Hendrickson (2000) use ClusterSim for a comparison of the tool performance when using the cluster tool scheduler provided with the simulation model and a real cluster tool scheduler. They show that the rule–based simulation scheduler, using either a push or a pull rule, is not capable of completely reproducing the behavior of the real scheduler, that applies a branch–and–bound search algorithm to find the sequence of robot moves that yields the highest tool utilization. Therefore, they propose integrating the real scheduler's program logic in the simulation software (cf. Chapter 4 of this monograph for further approaches for sequencing the robot moves within a cluster tool).

Paré et al. (2002) apply ToolSim to analyze different configurations of a dielectric deposition tool. The options considered are side–by–side or stacked configuration of the load locks as well as different batch capacities of the load locks. The goal of the study is to identify the configuration that provides the highest throughput.

Different tools in the thin film area were modeled by Aybar and Potti (2002). The pre–defined simulation models shipped with ToolSim were used to study the impact of changing the frequency of cleaning cycles and the effects of different chamber configurations, to compare single and parallel operation mode, and to identify the bottleneck process at a sputter tool.

Other Simulation–based Approaches

Unfortunately, some of the authors of publications on cluster tool simulation do not explicitly mention the simulation environment they used to build their cluster tool model. For example, Atherton et al. (1990) perform case studies to evaluate the impact of process recipes, sequence of operations, etc., using a ”detailed simulation model”of a multi–process tool. They focus on the waiting time of wafers as a performance characteristic. However, they do not give any hint whether the model was built using a general purpose simulation language or was implemented otherwise.

Similarly, in (Atherton et al. 1990), potential throughput advantages of cluster tools and the problem of shifting bottlenecks are discussed without presenting any details on how the simulation model used in this study had been created.

Wood and Saraswat (1991) compare a cluster–based wafer fab to one without cluster tools. They perform *Monte Carlo* simulations of the models of these two fabs to predict cost and throughput time performance. Their results indicate that cluster–based fabs have a significant throughput advantage at a relatively small additional cost per wafer as compared to non–cluster–tool fabs.

This study is repeated in (Wood 1997), now based on fab data collected in a survey of equipment vendors, chip manufacturers, and relevant literature. The simulation of the resulting fab models of a cluster–based fab and an equivalent fab without cluster tools leads to the conclusion that a cycle time reduction of up to 50 percent can be accomplished by employing cluster tools without a significant increase in cost per wafer.

The cluster tool model developed by Koehler et al. (1999) is used to predict cycle times of tools applied in the production of thin film heads used in hard drive storages. Though not explicitly mentioned in the paper, it can be assumed that the Arena simulations software was used to build the model. The simulation results for different tool configurations are used in a probabilistic throughput model to predict throughput of multiple cluster tools where the individual tools are subject to random downtimes.

Another paper on simulation of cluster tools for comparing design alternatives, presented by Poolsup and Deshpande (2000), presents results for different studies without elaborating on the details of modeling the cluster tools. Among the alternatives considered are single–arm vs. dual–arm robot, different capacities of the cooling station, and different priorities of the robot activities.

In (Pampel et al. 2000), the authors present a study that aimed at improving the productivity of a workcenter of dry–etch cluster tools. In particular, the authors compared workcenter configurations and cluster tool operation modes with respect to wafer throughput. Furthermore, the impact on throughput of hardware and recipe changes at a single cluster tool is considered. For a range of process times, different handler configurations, wafer alignment procedures, and dispatching rules are compared. The authors emphasize that slight deviations of the individual process times can have significant impact on wafer throughput. Therefore, they collected input data for the simulation model from existing equipment with an accuracy of one second. The respective throughput results were verified by on–tool tests. The authors mention some of the problems of the existing equipment, like the lack of a look ahead strategy of the cluster tool controller and the need for extensive automation of data transmission from equipment to simulation software.

Lemmen et al. (1999) use a dynamic simulation model of the cluster tool type applied in the deposition area of a wafer fab. This model is used to compare different scheduling strategies for the deposition area concerning the impact on throughput and cycle time of that area. The authors show that applying area–level scheduling rules can significantly improve the fab performance. A similar problem is addressed in Chapter 5 of this monograph.

Quite similar to discrete–event simulation is the approach of event–graph simulation. Nehme and Pierce (1994) use this approach for the modeling of cluster tools. They apply a *resident–entity* model instead of the more common *transient–entity* model, i.e., the events in the model are associated with the resident entities like chambers or robots and not with the wafers. This approach has proven to lead to very short execution times of the simulation model, however, the resident–entity models are not able to cover some of the properties that can be modeled using transient–entity simulation. The authors give a detailed overview on how the simulator and the underlying objects and methods were implemented in C++.

Schruben (1999) addresses the problem of detecting and avoiding deadlocks when simulating cluster tools. The presented simulation model is based on event graphs and provides animation of the robot activities. He suggests to use simulation to prove that a specific schedule for the cluster tool operation is deadlock–free.

The simulation model presented by Schruben is called *Cluster Tool Performance Simulator* (CTPS) and has been developed at Cornell University and later at University of California, Berkeley. The model data is entered using a spread sheet, containing five areas:

- tool configuration, containing the number of process modules, number of load locks, pump and vent times as well as the type of the process modules. Three classes of modules can be used: Single chamber modules (only one wafer is processed at a time), batch chamber modules (batches of $n > 1$ wafers are processed in parallel), and index modules (an index module consists of different chambers in which a wafer undergoes a series of different processing steps);

- robot movement, where the move times of the handler between chambers and load locks are specified using a from–to matrix;

- product description, associating a process flow with each wafer in the model. A lot can contain wafers with different flows;

- process description, containing the required process module and the associated processing time for each process flow ;

- simulation run control, specifying simulation run length, dispatching rules, and output files.

The simulator provides a schematic animation of the simulation run, cumulative flow plots of the cycle times and wafers entering and exiting the cluster tool, and Gantt charts of the simulated entities.

Event–graph simulation is also used by Pederson and Trout (2002). They use the SIGMA simulation environment (Schruben and Schruben 2000) to create a cluster tool model and compare this modeling approach to a spreadsheet model based on bottleneck analysis.

Chandrasekaran (1999) presents an approach that relates the total processing time of a lot of wafers to process parameters like temperature, pressure and processing time. The goal is to predict how changing the physical or chemical processes taking place in the cluster tool's processing chambers affect the performance of the complete cluster tool. To this end, they use a *response surface model* (RSM) of a CVD Tungsten deposition process, relating the deposition rate to the process parameters reactor pressure and temperature. Using the deposition rate, the process time is derived for a specific deposition thickness. The process time is then used as input for two models applied to derive the cycle time of a complete lot: an analytic network model and a simulation model.

The network model represents the sequence of wafer moves in the cluster tool as a directed graph. Using this graph, the lot cycle time can be derived using a simple algorithm. This analytic model can be applied if the sequence of handler moves is known in advance. If this sequence is not known, a simulation model can be used. The simulation model applied by the author is based on the Cluster Tool Performance Simulator of Schruben (1999). The same approach is presented in (Herrmann et al. 1999) and (Herrmann et al. 2000).

This integration of process parameters in the simulation model allows one to assess the impact of changes in process parameters on the cluster tool performance. Using such a model, it is easier to communicate process related topics between process engineers and operations personnel. It has to be annotated that integration of the process parameters can be achieved with all the presented models, because the computation of processing times for a given set of process parameters is relatively simple.

Ruppert et al. (2000) develop *regression spline meta–models* of cluster tools, using output from a simulation model. With such meta–models, the output of the simulation can be predicted rapidly for different parameter settings. Since the component functions of the regression spline models can be investigated to analyze the factors influencing cluster tool throughput, these models also give further insight into the cluster tool characteristics.

The disadvantage of this approach and of similar methods, like Neural Networks, is that after changing the configuration of the cluster tool under investigation, the respective model has to be adapted to reflect the changed configuration. Furthermore, if no cluster tool is available to generate the training data for the model, it is necessary to create a simulation model to receive performance data of the changed cluster tool configuration. Finally, with the speed of processors increasing, the run time of simulation models is decreasing and it is questionable whether the effort of generating a meta–model is worth the relatively small gain in computing time.

3.1.3 Comparison of Analytical and Simulation Approaches

The two groups of modeling paradigms, analytical and simulation models, will now be summarized and compared. The presented analytical methods can be divided into two groups: on the one hand, there are simple closed–formulae approaches that are easy to understand and easy to implement. Due to their simplicity, they are applicable to model simple cluster tools only and can not be used to model tools with sophisticated layout or complicated internal logistics.

On the other hand, there are more complex analytic approaches, like Petri nets, that are able to model virtually all kinds of cluster tools. The more complex the cluster tool, the more complicated it becomes to maintain the associated model, since the modeling methodology is more complex and harder to comprehend.

Therefore, for analytical approaches, a trade–off has to be made for the application that the model is needed for. From a pragmatic point of view it would be better to have one methodology that is easy to implement and to understand and at the same time allows for greater flexibility concerning the kind of models that can be created.

Simulation offers great flexibility concerning the degree of details put into the model. The model builder can start with a simple and coarse model of the tool and add details according to the requirements. Changes of the cluster tool configuration are much easier to implement in simulation models compared to analytical models.

An important advantage of the simulation approach is that the scheduling algorithms used in the cluster tool can easily be implemented if they are known, because most of the simulation environments allow the user to extend the model using a programming language.

Generating the different performance measures that might be of interest for the model builder is relatively easy using simulation, since either the simulation environment offers statistical functions for the model elements applied, or the model builder has access to the required data using a programming interface. Therefore, it is possible to include additional performance measures into the simulation model even after the original model is generated. Most analytic approaches are built to compute a specific performance measure, and therefore adding another performance measure might not be possible.

Once a model of a single cluster tool is created, it can be re–used to model pools of several cluster tools. Using analytical models however, in most of the cases it is not possible to use an existing model of a single cluster tool and build models of cluster tool pools because the analytical approaches have only limited flexibility concerning the complexity of the systems that can be modeled.

In some cases, equipment vendors provide simulation models of their products. This enables the equipment buyer to use an accurate model, since the vendor has all knowledge about the cluster tool and its scheduling algorithms necessary to build a detailed model.

The usability of the two approaches in a semiconductor manufacturing environment depends on the specific purpose of the model. If only rough cut performance data is needed, then closed formulae or simple spreadsheets might be the better choice because they can be applied and communicated easily. Furthermore, an analytical approach may help to gain insight *why* the performance measure of interest is influenced by the model parameters. On the other hand, if a series of exact performance data of a complex cluster tool is needed, analytical models do usually not provide the same flexibility and modeling power as simulation.

In general, the simulative approach seems to be the more promising one for modeling cluster tools, while there might exist specific applications where analytic methods can be applied successfully.

3.2 CluSim

In order to perform the simulation studies described in the following two chapters, an object–oriented simulation engine for cluster tools, called *CluSim*, was developed using the programming language C++ during the course of the research for this monograph, cf. (Schmid 1999) and (Bohr 1999). For the management of the events occurring during a simulation run, the software library *simlib++* was used, cf. (Kern and Gerlich 1994).

By creating a simulation software from scratch instead of using a general purpose simulation environment or a commercial cluster tool simulation software, the greatest possible flexibility is given. The degree of detail of the simulation model can be increased or decreased arbitrarily, the logic of the control of the individual parts of the simulation model can be modified, all statistical data necessary can be extracted from the model, etc.

3.2.1 Objects in CluSim

To give an overview of the modeling capabilities of CluSim, the essential objects in a simulation model created with CluSim will be presented. An overview of such a model is given in Figure 3.3.

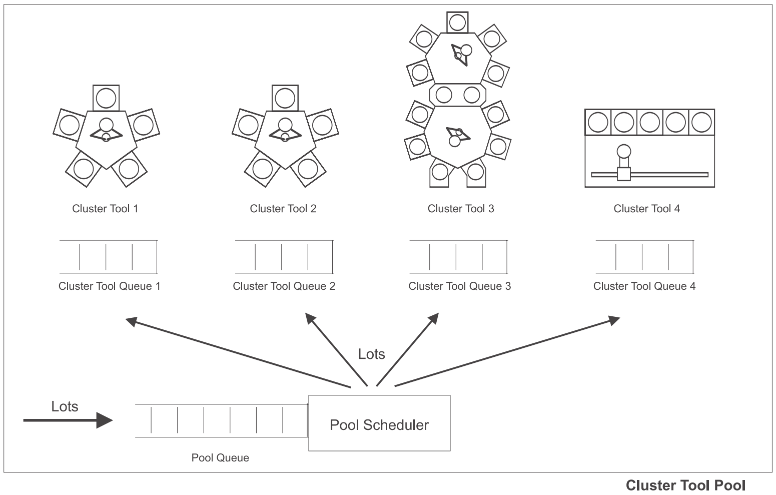


Figure 3.3: *Objects in CluSim*

A simulation model in CluSim can consist of an arbitrary number of cluster tools, each of which can have an arbitrary number of process chambers, load locks, and handlers. Hence, different types of cluster tools can be simulated in a single model.

To make the simulation model as generic as possible, a modular approach was applied. The individual components of the simulation model can be exchanged easily with alternative components. For example, one can choose the module responsible for the control of the handler from a set of different modules.

Cluster Tools

For each of the different cluster tool types defined in the simulation model, the following parameters can be specified:

- number of cluster tools of this type,
- number of load locks,
- number and types of process chambers,
- number and types of handlers,
- type of scheduling algorithm.

The different types of scheduling algorithms that can be chosen will be discussed in detail in Chapter 4.

If a more detailed model is necessary, one can specify additional parameters such as the time necessary to open or close a process chamber.

Each cluster tool is equipped with its own queue, allowing the cluster tool to decide itself which lot to choose next for processing if more than one lot is available.

Load Locks

For each load lock, the model specifies the pump and vent time, as well as the time required for moving a lot of wafers inside the load lock. This movement of the lot is necessary if the handler can not move along the vertical axis.

Process Chambers

The model parameters for a process chamber consist of the time necessary to prepare the chamber for accepting a wafer from the handler, as well as the time to prepare the chamber before the process can be initiated.

Handlers

There are two possible ways to define the move times of a handler. If the exact move times between the individual cluster tool modules are not known, the user can specify a constant move time for all source–destination pairs. A matrix of move times can be used if the different move times for the source–destination pairs are known. In both cases, different move times can be specified for a loaded and an unloaded handler, respectively.

In addition to the move times, the time required for inserting a wafer into a process chamber and for removing it from the chamber can also be defined.

Recipes

Recipes are specified in CluSim by listing the individual processing steps and the process time for a wafer. For each processing step, one can declare alternative processing chambers capable of performing the process step. It is also possible to define recipes that require processing on several cluster tools.

Lots

To model a lot, the number of wafers in the lot and the recipe associated with the individual wafers has to be specified. A wafer can be marked as *scrap*, e.g., to emulate the effects of failures during processing a wafer. It is also possible to assign a higher priority to certain wafers, so they will be processed before other wafers with lower priority.

The time between two consecutive arrivals of a lot at the pool queue can be defined as a random variable. The model builder can choose among the following distribution types: Deterministic, Exponential, Uniform, Erlang, and Normal.

Downtimes

For the components *process chamber, handler, load lock* and for a complete cluster tool, downtimes can be specified. A downtime will occur after the respective component has processed a certain number of wafers.

This number as well as the length of the downtime can be specified as a random variable.

Pool Queue and Scheduler

Whenever a new lot is created during a simulation run, it is put into the pool queue. The pool scheduler will check the pool queue for lots to process whenever a cluster tool becomes available for processing. Alternatively, a new lot can be assigned immediately to a cluster tool capable of processing that lot, and it is placed in the queue associated with that specific cluster tool.

3.2.2 Simulation Run

The simulation model is *data driven*, i.e., when starting the simulation, a text file containing all model parameters is parsed and the specific cluster tool model is generated. With this approach, a series of simulation runs with different models can be run in batch mode. The syntax of the input file has to follow the grammar specified in Appendix A. An example of an input file is presented in Appendix B.

3.2.3 Simulation Results

The simulation program computes a number of output statistics after a simulation run, such as:

- cycle times of wafers,
- raw processing time (cycle time minus waiting time),
- average processing time of a specific recipe,
- utilization of the components of a cluster tool.

For the studies described in this monograph, the total completion time for a given sequence of lots is of special interest.

For debugging purposes and to gain insight into the internal scheduling logic, a trace of the simulation events can be generated during the simulation run.

3.2.4 Sample Model

As an example for modeling a cluster tool with CluSim, the cluster tool presented in (Perkinson et al. 1994) is modeled. For validating the performance data resulting from the CluSim simulation runs against the analytical results of Perkinson et al. (1994) as derived in Eqn. (3.1), two cluster tool models have been investigated. The set of parameters is shown in Table 3.2. N_C denotes the number of process chambers, N_W is the number of wafers in one lot. The time to load and unload a lot to and from the load lock is denoted by T_{load} and T_{unload}, respectively. Finally, T_{proc} is the processing time in all process chambers, and T_{move} is the robot move time between all chambers and the load lock.

Table 3.2: *Parameters for Cluster Tool Models*

	N_C	N_W	T_{load}	T_{unload}	T_{proc}	T_{move}
Model 1	3	25	20 sec	20 sec	100 sec	20 sec
Model 2	3	25	20 sec	20 sec	100 sec	30 sec

A handler move time T_{move} of 20 sec as in Model 1 will lead to idle times of the handler, whereas for $T_{move} = 30$ sec, no idle times will occur.

Table 3.3: *Analytical Results*

	T_{move} (sec)	Cycle Time (sec)
Model 1	20	4760
Model 2	30	6230

For the two models, Perkinson et al. (1994) compute the cycle times displayed in Table 3.3. The simulation results produced by CluSim are as follows for $T_{move} = 20$ sec:

```
                         lots  average  average
                          per    cycle   CT per
lotname   lots   wait     day     time    wafer
-----------------------------------------------
FirstLot     1   0.4%     8.6   4760.0    190.4
```

For $T_{move} = 30$ sec, the results are:

```
                         lots  average  average
                          per    cycle   CT per
lotname   lots   wait     day     time    wafer
-----------------------------------------------
FirstLot     1   0.3%     8.6   6230.0    249.2
```

As can be seen from column ”average cycle time”, in both cases the deterministic simulation produces the same cycle time as predicted by the analytical model.

In order to prove that CluSim produces accurate simulation results also for more complex cluster tool types, we performed several studies of cluster tools that are applied in real wafer manufacturing and compared the results of the simulation models to the performance data measured for the real cluster tools. The recipes that were used in the study contained confidential information and can therefore not be reproduced in this monograph. We simulated cluster tools processing a single lot, as well as cluster tools that processed a sequence of lot in parallel mode. As the main performance measure, the cycle time of the individual lots was used. The comparison showed that the simulation results had a relative error of 5 to 10 percent compared to the real performance data, even for complex cluster tools like the one depicted in Figure 3.4.

3.2.5 Case Studies

We will now present two case studies that show areas of application of the presented cluster tool simulation software.

Effect of Lot Size on Cycle Time

Using a simulation model of the cluster tool depicted in Figure 3.4, we will study how the number of wafers in a lot affects the cycle time of the lot.

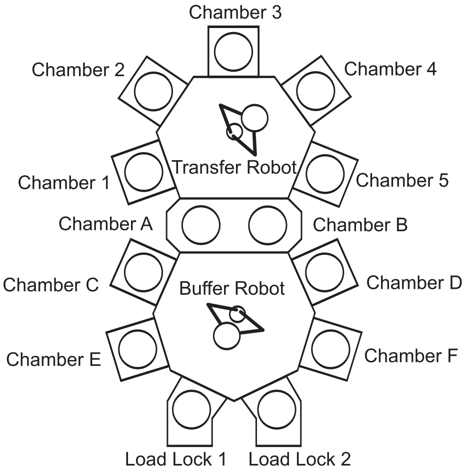


Figure 3.4: *Cluster Tool Model for Study on Lot Size*

In the simulation, a single lot is placed in one of the load locks and the lot cycle time and the average flow factor of the wafers is measured. The *flow factor* FF of a wafer is defined as follows:

$$FF = \frac{(\text{Cycle Time})}{(\text{Raw Process Time})} . \tag{3.13}$$

The *raw process time* of a wafer is defined similarly to the wafer's cycle time, however, now we consider a cluster tool that is in idle mode, and the wafer under consideration is the only wafer that is currently present in the cluster tool. Hence, the raw process time reflects the cycle time of a wafer reduced by all (unnecessary) waiting times.

The lot loading and unloading time is not considered in this study. The wafer recipe used is shown in Table 3.4.

Table 3.4: *Recipe for Study on Lot Size*

Chamber	E|F	A	2	B
Process Time (sec.)	30	0	75	31

Chambers E and F are identical, therefore either of them can be used for the first recipe step. Processing time in Chamber A is zero, because this chamber is only used to pass the wafers through to the upper mainframe of the cluster tool. The sum of the process times is 136 seconds, the sum of the transport times is 105 seconds, therefore the raw process time of wafers with that recipe, including transport time, is 241 seconds.

As can be seen from Figure 3.5, the average flow factor of the wafers increases when the lot size is increased and asymptotically approaches a value around 2.2. The waiting time of a wafer increases for larger lot sizes because the wafer has to wait more often for resources occupied by other wafers to become available.

The evolution of the lot cycle time divided by the number of the wafers shows the opposite behavior. When increasing the lot size, this value approaches 140 seconds, which is slightly higher than half the raw process time of a wafer. This improvement is caused by the parallel processing of wafers, also called *pipelining*.

This study shows that lot sizes larger than 15 wafers do not significantly improve the cycle time. However, the larger the lot sizes, the more wafers can be produced without pumping and venting the load lock to change lots. This improves the productivity of the tool. Therefore, lot sizes of 25 or 50 wafers are common in many wafer fabs.

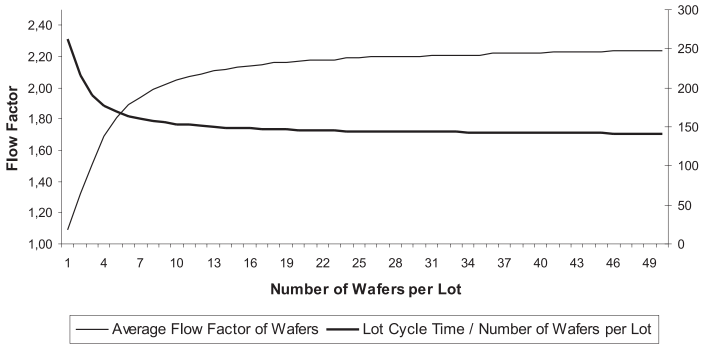


Figure 3.5: *Impact of Lot Size on Cycle Time*

Choosing the Optimal Recipe Mix

For cluster tools that allow the processing of wafers of different recipes in parallel, an improvement in throughput and cycle time can be achieved without changing the internal scheduling mechanisms by choosing the optimal combinations of recipes.

To estimate the performance gain that can be achieved using this approach, we tried to identify combinations of two lots that lead to small lot cycle times when the two lots are processed in parallel, using the cluster tool simulator CluSim, cf. (Dümmler 2000). The following simulation experiment was performed using again the model of the cluster tool depicted in Figure 3.4 with the recipes in Table 3.5.

Starting with an empty cluster tool, we simultaneously put a lot of recipe $i \in \{1, \ldots, 4\}$ in load lock 1 and a lot of recipe $j \in \{1, \ldots, 4\}$ in load lock 2. For all of the 16 possible combination of recipes, we measured the cycle time for both lots. For each combination (i, j), we computed the ratios $T_{i,j}/T_i$, where $T_{i,j}$ is the cycle time of a lot of recipe i when processed together with a lot of recipe j. T_i is the cycle time of a lot of recipe i when

Table 3.5: *Recipes for Recipe Mix Optimization*

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

processed exclusively. The resulting ratios are displayed in Table 3.6.

Table 3.6: *Cycle Time Ratios for Combinations*

Recipes	$j = 1$	$j = 2$	$j = 3$	$j = 4$
$i = 1$	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 recipes 2 and 3 leads to an increase of 29% in cycle time for recipe 2 and of 38% for recipe 3. Hence, this is a more favorable combination than for example recipes 1 and 3, which leads to an increase in cycle time of 85% for recipe 1 and of 50% for recipe 3. Combining recipes 2 and 3 leads to shorter cycle times since recipe 2 does not make use of chamber 2, which is the bottleneck chamber of recipe 3 and vice versa. On the other hand, when combining recipes 1 and 3, wafers of recipe 1 visit chamber 2, the bottleneck resource for recipe 3, causing higher waiting times for both recipes.

Table 3.6 can be used as a guideline for operators to choose combinations of recipes 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.

In several simulation studies, Niedermayer (2002) shows how the cycle times of lots is affected if the processing of the second lot starts with a delay and presents approximation techniques for predicting these effects.

4 Scheduling of Cluster Tool Activities

After purchasing a cluster tool, the semiconductor manufacturer usually depends on the tool manufacturer when adaptations of the control logic of the cluster tool are necessary, e.g., when the cluster tool configuration is changed or when the tool performance in general has to be improved. The tool manufacturers are very reluctant to disseminate information on how the control of the cluster tool processes, especially the sequencing of the individual operations and transport moves, is performed. Hence, only little knowledge is available on the actual control algorithms. Geiger et al. (1997) even note that the development of effective scheduling procedures is lagging behind the rapid development of new cluster tool technology.

The task of scheduling the cluster tool operations is not trivial. The main goal is to maximize the productivity of the cluster tool, i.e., to increase throughput and to reduce cycle times. However, there are also side constraints to take care of. Deadlocks and situations of circular wait can occur, since different wafers are competing for the same resources in a cluster tool. Further constraints are imposed when the chemical or physical processes require that two consecutive operations be performed within a certain time frame. This constraint is called *residency time constraint*.

The goal of this chapter is to present different approaches to the sequencing of cluster tool activities and to compare them to each other. The chapter is structured as follows: First, the scheduling problem is formulated. In the second section, the related literature on this problem and closely related problems is presented. Approaches to deadlock avoidance and deadlock resolution are discussed in the next section. Finally, the algorithms implemented in the cluster tool simulation model CluSim, introduced in the previous chapter, are presented and compared and the results are discussed.

4.1 Problem Description

According to Pinedo (2002), scheduling ”deals with the allocation of scarce resources to tasks over time. It is a decision–making process with the goal of optimizing one or more objectives.” The *scheduler* is the component of a cluster tool responsible for generating for each wafer to be processed in a cluster tool a sequence of process steps, where each step takes place in the individual process chambers of the cluster tool. This sequence must be conforming with the sequence of process steps as defined by the respective wafer’s recipe. In the context of modeling a cluster tool using simulation, the scheduling task is reduced to generating a sequence of handler moves, determining which wafer will be transported next to which process chamber and at what time the processing will start.

Before generating the schedule, the state of the cluster tool must be determined, i.e., the scheduler needs to know the number, state, and position of all wafers in the model as well as the state of the cluster tool resources (idle, busy, or down). The scheduler will start whenever a change in the cluster tool configuration takes place that has not been considered in the current schedule. Such an event can be the failure of a cluster tool resource, a re–configuration of a process chamber, or the arrival of a new lot at a load lock. The scheduling approaches presented in this chapter only consider the current state of the cluster tool and do not take into account future states, e.g., caused by chamber failures.

In order to understand the complexity of the problem, we will investigate the following problem, cf. (Nguyen 2000): Consider a cluster tool with one load lock and one single–blade handler. A lot of N_L identical wafers is to be processed in the cluster tool. N_S process steps are required to process a wafer. Therefore, each wafer must be moved by the handler $N_S + 1$ times, so for the complete lot $N_L(N_S + 1)$ wafer moves must be scheduled.

Nguyen (2000) derives the number of feasible sequences for a cluster tool with one handler and one load lock. As an example, a cluster tool with four process chambers, where chambers three and four can perform the same operation, and a lot of three wafers are considered. If each wafer has to visit chamber one and two and then either chamber three and four, there are 552 possible sequences per lot that the scheduler can choose from.

4.1.1 Gantt Charts

The sequence of instructions generated by the scheduler defines a scheduling plan of the cluster tool resources. Such a plan can be visualized using a *Gantt chart*. In a Gantt chart, resources are displayed as horizontal bars along the time horizon. Active or occupied states of the resources usually are indexed by the job (in our case, the wafers) occupying the resource. Arrows connecting two bars indicate the transfer of wafers from one resource to another. At every point in time, no resource can be occupied by more than one job.

Figure 4.1 shows a Gantt chart for a cluster tool configuration consisting of three process chambers, one load lock, and one handler. The individual wafers are indicated by different textures. The periods when a resource is occupied are divided into *active* periods, if processing takes places, and *passive* periods, if the resource waits for a wafer to arrive or for a wafer to be picked up.

4.1.2 Fundamental Periods

When a wafer occupies a cluster tool resource it blocks this specific resource, so that no other wafer can occupy the same resource. This leads to the generation of points of synchronization among the resources, because the transfer of a wafer from one resource to another can only take place if both resources are in the correct state. If a lot of wafers of the same type has to be scheduled, this synchronization will lead to the formation of cyclic structures within a schedule. This means, that after a certain period of time, the cluster tool will be in a steady state, where a sequence of operations is repeatedly performed within the same time frame until there are no more wafers available. The *fundamental period* denotes the period of

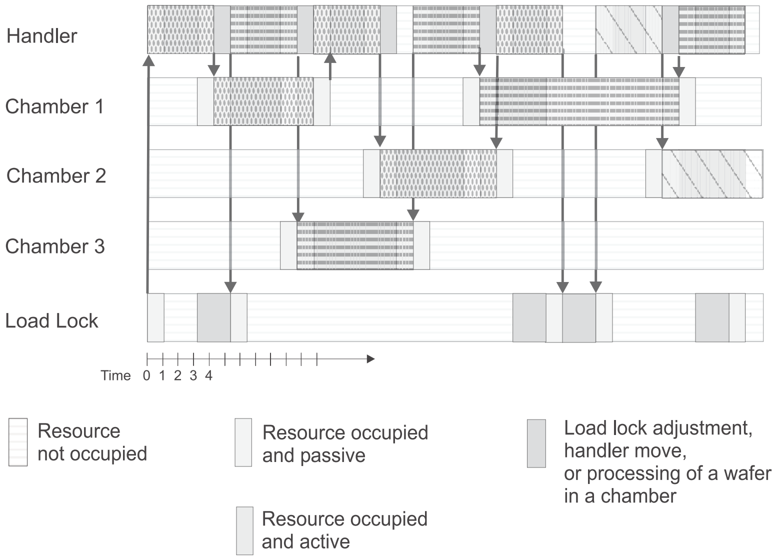


Figure 4.1: *Gantt Chart*

one such cycle in steady state, cf. (Perkinson and Gyurcsik 1996).

The time period until the cluster tool reaches steady state, i.e., until the first wafer is completed, is denoted as the *filling–up phase* or *initial transient phase*. The *completion phase* or *final transient phase* denotes the period following steady state, starting when there are no more new wafers available for processing.

During steady state, a subsequence of operations that loads and unloads each process chamber λ times is called a λ*–unit cycle*. A *cyclic sequence* of length $n > 1$, is formed by repeating n such λ–unit cycles.

4.1.3 Active Schedules

A schedule is called *active*, if no schedule can be constructed where one task of the schedule is executed earlier without delaying another task, cf. (Pinedo 2002). All the implemented scheduling approaches presented later in this chapter produce active schedules.

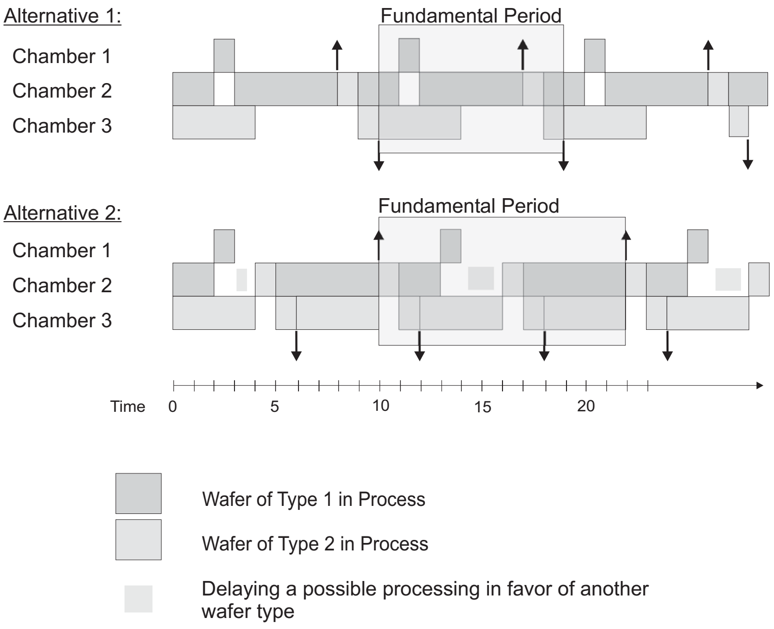
Note, however, that it is not always advantageous to start the next process step as soon as a resource becomes available. Figure 4.2 gives an example of an *active wait* situation of a process chamber leading to higher chamber utilization and throughput in steady state.

The wafer recipes for this example are as follows. Wafers of Type 1 have to be processed in Chamber 2 for two time units, in Chamber 1 for one time unit and again in Chamber 2 for five time units. Wafers of Type 2 are processed for four time units in Chamber 3, then for one time unit in Chamber 2 and finally for one time unit in Chamber 3. It can be seen that by introducing an ”artificial”delay and increasing the waiting time of wafers of Type 1, the overall chamber utilization is increased by five percent and the overall wafer throughput is improved.

4.1.4 Robotic Flow Shops

According to the definition of Crama and van de Klundert (1997), a cluster tool can be considered as a robotic flow shop. This means, it consists of an input station M_0, namely the load lock, an output station M_{m+1}, which is identical to M_0, and a number of m machines or process chambers, $M_1, \ldots, M_m$. The process chambers can contain only one part at a time. Transportation is done by a robot, loading and unloading wafers to and from the chambers. There exist no buffers in the cluster tool. Furthermore, processing in a chamber usually can not be interrupted without damaging the wafer or at least making the operation void, so the operations are non–preemptive. The problem consists of the task of scheduling a number of jobs on the m machines in such a way that a certain performance characteristic, e.g., the cycle time of the wafers, is optimized.

Crama and van de Klundert (1997) show that robotic flow shop



Comparison:

	"Steady State" production time units of wafers of type 1	2	Total chamber utilization
Alternative 1	9	9	52 %
Alternative 2	12	6	56 %

Figure 4.2: *Example for Active Wait*

scheduling with the goal of minimizing cycle time in such an environment is strongly NP–complete, using a reduction from the Bin Packing Problem. Therefore, no algorithms are known yet that can generate an optimal schedule for cluster tools in polynomial time. Of course, in some cluster tools, there exist some deviations from the robotic flow shop model. For example, wafers may skip one or more chambers, and there might be more than one robot. These characteristics, however, do generally not reduce the complexity of the problem.

4.1.5 Regions of Operation

In general, a cluster tool can be operated in two regions: a *process–limited* region and a *robot–limited* region. In robot–limited mode, the bottleneck is the handler. As a consequence, chambers will wait idly for a new wafer to arrive or for the finished wafer being unloaded from the chamber. On the other hand, in the process–limited region, the performance is limited by one or more process chambers and the handler.

4.2 Literature

As already indicated, only a few publications are available on the cluster tool scheduling problem (CTSP). However, due to the relationship of the CTSP to some well–known scheduling problems like the Robotic Flow Shop Scheduling Problem (RFSSP), the Flow Shop Scheduling Problem (FSSP), the Job Shop Scheduling Problem (JSSP), or the Hoist Scheduling Problem (HSP), a large body of literature can be used as a starting point to investigate solutions to the CTSP.

One of the rare publications about schedulers that are actually implemented in cluster tools is (LeBaron and Hendrickson 2000). In the mentioned scheduler, a *Branch–and–Bound* algorithm is used to do an exhaustive search of all possible moves within a specified search horizon. The objective is to maximize utilization of the process modules within the search horizon. The advantage of this approach is that deadlock situations can be completely avoided, since for every search branch leading to a deadlock situation, there exists at least one equivalent branch without a deadlock. Another advantage cited by the authors is that the algorithm quickly adapts to changes in tool configuration or in recipes. Of course, there exists a trade–off between run time of the search algorithm and the quality of the solution. A similar algorithm was implemented in the cluster tool simulation model CluSim and will be presented later in this chapter.

In (Herrmann and Nguyen 2000), another Branch–and–Bound algorithm is presented and it is shown that the results for simple problem instances are better using this algorithm as compared to a simple push or pull approach. The authors also enumerate the one–unit cyclic sequences (cf. Section 4.1.2) for a sequential cluster tool with two and three process chambers and compute cycle time and lot makespan for these cyclic sequences. The cluster tool under investigation is a rather simple type with only one load lock and one handler.

The authors extend this algorithm in (Nguyen and Herrmann 2000) to also handle *hybrid* cluster tools, i.e., cluster tools that can have more than one process chamber for a specific operation. They also present a heuristic that only considers cyclic sequences and has significantly lower computational effort but still produces better results than the push and pull rules. The authors conclude that focusing on cyclic sequences is a good trade–off between computational effort and the resulting quality of the schedule.

An approach that is very flexible and that can be applied to cluster tools with changing setups is using *dispatch rules*, cf. (Morton and Pentico 1993). Dispatch rules are simple if–then rules, that are applied successfully in a variety of environments, cf., for example, (Conway 1965). They are easy to implement and require only little information about the actual environment in which they are applied. In Section 4.4.3, the dispatch rules implemented in CluSim are introduced.

Bierwirth et al. (1995) investigate the application of *Genetic Algorithms* (cf. Chapter 5) to dynamic and non–deterministic job shop scheduling. The authors show that Genetic Algorithms produce schedules that are only slightly better than those generated with simple dispatch rules.

In (Yim and Lee 1999), a scheduling algorithm based on *Simulated Annealing* is presented that is used to schedule the processing of wafers in a series of four cluster tools. The resulting schedules produce smaller makespan than those obtained by using dispatch rules, however, the computation time of the algorithm is significantly higher.

Shin et al. (2000) present a real–time scheduler for cluster tools with a robot with dual opposite blades. They model the cluster tool as a *finite state machine* (FSM). The reaction of the cluster tool to the occurring events, like the arrival of a new lot, has to be modeled in the FSM as well. An occurring event will then start a series of activities, specified in the FSM. As a consequence, the task of finding a good schedule is actually transferred to the person modeling the FSM.

Geiger et al. (1997) consider the problem of scheduling an Automated Wet Station (AWS). They compare two heuristics usually applied to the FSSP in connection with *Tabu Search*. Since the control of the robot moves inside the AWS is fixed, only the sequencing of jobs entering the AWS is considered. The approach produces ”high–quality”solutions within short computational time.

Oh (2000) presents several approaches for reducing the number of resource conflicts in a cluster tool. A resource conflict occurs when two wafers wait for the same chamber or the same handler, and residency time constrains do not permit that one of the wafers waits until the other wafer’s request is satisfied. A formula for computing the number of modules needed to ensure availability of modules at a specific process step is derived. Whereas this approach might be used for chamber conflicts, it usually can not be applied to avoid handler conflicts, because adding additional robots to a cluster tool is costly or even not possible due to technical constrains.

Another solution to the problem of resource conflicts mentioned by Oh (2000) is to apply priorities or dispatch rules. The author remarks that the disadvantage of simple ”if–then”decisions like dispatch rules is that no steady flow of wafers can be guaranteed and hence cycle times can not be predicted adequately. The proposed solution to the problem of transport conflicts therefore is to artificially introduce delays between process steps to maintain a steady flow of wafers in identical patterns. The delays have to be large enough to avoid transport conflicts but on the other hand should be kept as small as possible. Therefore, the optimal delays are computed using a Genetic Algorithm.

A related problem, namely the *Hoist Scheduling Problem* (HSP) is addressed in (Kats et al. 1999). In this study, the produced parts are identical. The authors present an algorithm based on a *Sieve Method* that produces optimal λ–unit cyclic schedules ($\lambda > 1$). One result of the study is that multiple–unit schedules often produce better performance than the simpler single–unit schedules.

A side constraint that is often imposed on the scheduling algorithm for chemical and quality reasons is the so–called *post–processing residency constraint*. For example, CVD (Chemical Vapor Deposition) and Rapid Thermal Processing require that the time a wafer spends in a process module after processing is finished does not exceed a certain limit.

Rostami et al. (2001) propose an algorithm that generates periodic schedules under post–processing residency constraints. Their approach is to first generate a simple periodic schedule according to the procedure presented in (Perkinson et al. 1994). If this schedule satisfies all constraints, the algorithm returns the result. Otherwise, the fundamental period is repeatedly increased, starting from the minimum value derived according to Eqn. (3.2). A schedule with a longer fundamental period is generated by modifying the original schedule with basic operations, for example by moving operations in the respective Gantt chart. Since this approach requires extensive computation, some heuristics are presented. In (Rostami and Hamidzadeh 2002), the proposed approach is extended to handle robot residency constraints, i.e., the time a wafer is allowed to spend on a handler can also be limited.

Several authors use *Petri Nets* (cf. Section 3.1.1) for the task of scheduling cluster tools. Kim et al. (2002) analyze a dual–armed cluster tool with delay time constraints using Timed Petri Nets. In their paper, they examine the region of process times that allows generating a feasible schedule that does not violate the delay time constraints. They also propose an alternative *swap method* for the dual–blade handler. This method uses the handler as a kind of buffer, allowing a wafer on the handler to wait for the next process chamber to become available.

Hendrickson (1997) states that the optimal operation point of a cluster tool is near the crossover between robot–limited and process–limited mode. At this operation point, both the robots and the bottleneck chambers are close to 100 percent in utilization. Hence, one way of improving the performance of process–limited cluster tools is to shorten the process time, e.g., by adding another process chamber to remove the bottleneck. In case of a robot–limited cluster tool, increasing robot speed or optimizing the transport sequence and therefore reducing the number of robot moves will lead to a better utilization of the cluster tool resources.

The articles presented on the Cluster Tool Scheduling Problem and related problems use a broad variety of methods and in most of the cases produce satisfactory results for the specific area of application. Since one of the advantages of introducing cluster tools into the semiconductor manufacturing process is the flexibility of the cluster tool configuration, a viable scheduling technique should be applicable to a broad range of different scenarios and should not be constrained to a certain cluster tool configuration. Therefore, the application of the scheduling techniques presented later in this chapter that were implemented in CluSim are not limited to specific cluster tool configurations and can operate in a variety of scenarios.

If modifying the scheduling algorithms is not possible, other approaches to optimize cluster tool performance have to be chosen. For example, Rogatty and Boebel (1996) present a case study where de–clustering a series of process steps actually improves performance of the tools involved. Another example for performance improvement of cluster tools without affecting the scheduling mechanisms is given in Chapter 5. The case study on optimizing recipe mix presented in Chapter 3 also presents an approach for optimizing throughput and makespan of a single cluster tool without changing the internal scheduling mechanisms.

4.3 Deadlocks

A major problem when scheduling the operations of a cluster tool is the occurrence of *deadlocks* (also known as *mutual blocking*). Coffman et al. (1971) identified four necessary conditions for a deadlock situation:

- Each resource (in the case of cluster tools: chamber or handler) can only be used by a single process (wafer) at a given time.
- While using a resource, a process can require additional resources (e.g., after being processed in a process chamber, a wafer waits for the handler to be picked up).
- Processes can not be forced to release a resource.
- There exists a closed chain of two or more resources, that currently are occupied by wafers and the respective wafers are waiting for the next resource in the chain.

Concerning cluster tools, the first three requirements obviously are satisfied. The fourth condition will be satisfied for some combinations of wafers, or recipes, while for other recipes, no deadlock is possible because condition four will never occur. Figure 4.3 shows a scenario where two wafers block each other.

Wafer 1 has to be processed in Chamber 1 for three time units, and then in Chamber 2 for one time unit. Wafer 2, after being processed for two time units in Chamber 3 and then for two time units in Chamber 2, has to be processed in Chamber 1 for two time units. As can be seen from Figure 4.3, the cluster tool is in a deadlock after four time units, since Wafer 1 can not be transferred into Chamber 2 and Wafer 2 can not be transferred into Chamber 1.

4.3.1 Deadlock Detection

Even though it might be easy for the human eye to detect that a system is in a deadlock situation, in a lot of scenarios the presence of a deadlock is

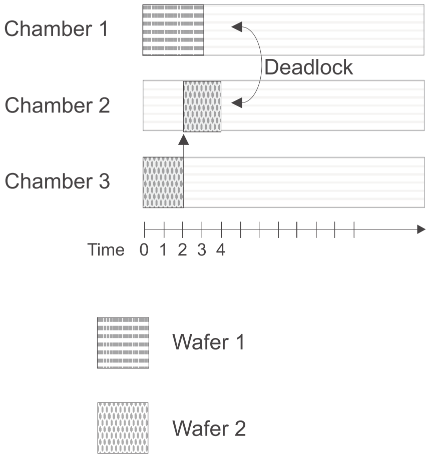


Figure 4.3: *Simple Deadlock Situation*

not trivial to detect by an algorithm.

Schruben (1999) solves the task of deadlock detection by comparing the simulated time needed to process a set of wafers in parallel to the simulated time necessary to process each of the wafers sequentially. If the completion time in parallel mode is longer than in sequential mode, it is supposed that the system is in a deadlock state. When using this approach, significant computational effort is necessary to detect a deadlock.

More efficient approaches continually monitor the system state and detect blocking situations using, for example, graph–theoretic algorithms. The approach presented in (Deuermeyer et al. 1997) is based on *dynamic entity–resource graphs* as proposed by Holt (1972). These directed graphs display the dependencies between resources and processes at a given moment. In the context of cluster tools, the nodes of a graph represent resources and wafers currently in process. An edge leading from a wafer to a resource means that the wafer is waiting for the resource to become available. An edge leading from a resource to a wafer means that the resource is currently occupied by the wafer. Graphs containing at least one *strongly connected component* with no edge leading out of the component are in deadlock state. Figure 4.4 displays the deadlock–situation of Figure 4.3 as an entity–resource graph. C1 to C3 denote Chamber 1 to Chamber 3, respectively.

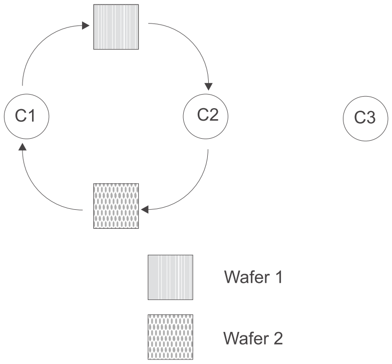


Figure 4.4: *Dynamic Entity–Resource Graph*

4.3.2 Deadlock Resolution

To resolve a deadlock situation, the request for a resource of one of the wafers leading to the deadlock has to be delayed. Schruben (1999) implements this approach by restarting the simulation after detecting a deadlock. The release of the wafer causing the deadlock into the cluster tool is delayed until after the time when the deadlock occurred in the original setting. Since the simulation has to be restarted completely, this method is rather inefficient.

Deuermeyer et al. (1997) propose to remove one of the wafers from the system. This could be done, for example, by placing this wafer back into the load lock, hence releasing a blocked resource. Choosing the wafer that leads to the optimal resolution of the deadlock situation is an NP–complete problem, cf. (Leung and Lai 1979). Therefore, Schruben (1999) proposes to use heuristics for choosing the right wafer. For example, the last wafer that has been taken from the load lock, or the wafer with the least remaining processing time can be chosen.

Both approaches are rather myopic, because by avoiding only the current deadlock situation, it is not guaranteed that a similar situation will not occur with another wafer. Due to the often cyclical structure of the cluster tool schedules, it is quite likely that another deadlock will occur. Furthermore, it is questionable whether the approaches can be implemented in a cluster tool, since process time constraints often do not allow to delay the processing of wafers.

4.3.3 Deadlock Avoidance

One way of preventing the occurrence of deadlocks is to reduce the number of jobs released into the system, such that a deadlock is not possible, independent on how the schedule for that system is derived, cf. (Wu 1999).

This approach offers flexibility in choosing the scheduling strategy and therefore allows one to use the strategy delivering the best performance. On the other hand, when the number of wafers in the cluster tool is limited, it is not possible to avoid *starvation*, i.e., the waiting of resources for new jobs to arrive. As a consequence, the productivity of the cluster tool is reduced and cycle times increase. Furthermore, *blocking* can still occur when a machine finishes a job and can not be unloaded because there is no buffer space.

Wu and Zhou (2001) propose an approach for Automated Manufacturing Systems based on Petri Nets that avoids deadlocks and reduces starvation and blocking. They improve the so called *Maximally Permissive Policy* by introducing a policy that is slightly more restrictive. As mentioned already, by reducing the number of wafers that are allowed to be processed in the cluster tool in parallel, the utilization of the cluster tool is reduced and therefore this approach has significant negative impact on the tool performance.

A more restrictive approach can be denoted as *batching* and is presented in (Bodner and McGinnis 1997). The authors propose to decompose the mix of wafer types to remove potential conflicts by separating wafer types with recipes that might lead to deadlocks and by separating recipes with re–entrant flows (i.e., flows where a wafer will be processed in the same chamber more than once) into different production runs. Since this approach can lead to a significant constraint on the combination of lots for parallel processing and also to a reduction of cluster tool performance, it seems less applicable.

4.4 Scheduling Approaches Implemented in CluSim

In the previous sections, several approaches of scheduling cluster tool operations have been presented. In order to implement the simulation model introduced in Chapter 3, some of these approaches were chosen for implementation. The goal was to develop a simulation model that provides a high degree of flexibility: the user should be able to easily model different cluster tool layouts, a great variety of recipes, handler types, etc. Therefore, most of the presented approaches could not be chosen for application in the simulation model, since they are too restrictive in their modeling assumptions. Many of the algorithms provide useful results only if the solution space is of limited size. If the number of resources, jobs, and job types is increased, the explosion of solution space leads to computation times that exceed the computational resources available in common manufacturing environments.

Algorithms for generating cyclic schedules, like the approaches used in Hoist Scheduling, might provide good schedules in steady state conditions. However, in the context of cluster tools, the time periods when the tool is in steady state are very short, because, for example, the operation of a cluster tool in parallel mode with different wafer recipes prohibits maintaining a steady state for long periods.

Another deficit of many of the presented algorithms is that they do not consider *anticipatory moves*. It is not possible to move an idle handler to a position where it can pick up the wafer that finishes processing next.

Therefore, only two different approaches were implemented in CluSim. These approaches will be introduced later in this section. First, we will introduce the performance criteria that were used to assess the performance of these approaches.

4.4.1 Objective Functions

When faced with the task of choosing an appropriate objective function to evaluate different scheduling strategies, at least two contradictory aspects of optimization have to be considered.

- On the one hand, the goal is to optimally utilize the expensive cluster tool, and therefore to reach a high throughput.
- On the other hand, it is necessary to assure the in–time production of the ordered wafers. To avoid long waiting times, the utilization of the cluster tool resources should be kept low.

Three objective functions will now be introduced, each with a different weighting of these two goals. They will be used in the following sections to compare the implemented scheduling strategies.

Objective Function 1: Average Flow Factor of Wafers

If we choose the *average flow factor* of wafers as the objective function, the goal can be defined as follows:

$$\min \left(\frac{1}{n_W} \sum_{i=1}^{n_W} \frac{CTW_i}{RPTW_i} \right) , \tag{4.1}$$

where

- n_W is the number of wafers considered,
- $CTW_i, i = 1, \ldots, n_W$ is the cycle time of wafer i, i.e., the time from picking this wafer from the load lock until returning it to the load lock, and
- $RPTW_i, i = 1, \ldots, n_W$ is the raw process time of wafer i.

Objective Function 2: Average Chamber Utilization

If the goal is to reach a high utilization of the process chambers, a possible formulation of the respective objective function is:

$$\max \left(\frac{1}{n_C} \sum_{i=1}^{n_C} \rho_{C_i} \right) . \tag{4.2}$$

In this equation,

- n_C is the number of process chambers, and
- $\rho_{C_i}, i = 1, \ldots, n_C$ is the utilization of process chamber i.

Objective Function 3: Wafer Throughput

When using the following objective function, the focus is on achieving a relatively small equivalent raw process time:

$$\max \left(\frac{1}{GPT} \sum_{i=1}^{n_W} RPTW_i \right) , \tag{4.3}$$

where GPT is the global process time or makespan of all n_W wafers.

It is obvious that objective function 1 is more oriented towards assuring the timely completion of wafers or lots, whereas objective functions 2 and 3 aim at a high productivity of the cluster tool.

Of course, in reality, these objective functions will in most of the cases be combined with each other, or with other objectives, like on–time delivery. Furthermore, it will be necessary to associate different weights to the different objectives. These weights are subject to change over time, e.g., when a certain product type has to be finished in short time.

4.4.2 Exhaustive Search

The first of the implemented scheduling approaches is called *ExhaustiveSearch* scheduler.

Application

The ExhaustiveSearch scheduler finds the optimal schedule by browsing the whole search space of possible schedules. Optimality is defined by the choice of one of the objective functions presented in the previous section. This scheduling mechanism is not intended for use in a real cluster tool, since the search for the optimal schedule is computational very expensive. It is rather used for generating the optimal schedule in order to have a benchmark for other heuristic scheduling approaches and to gain insight into the generation of optimal schedules.

Algorithm

The ExhaustiveSearch scheduler uses the current system state of the cluster tool as the starting point. In this context, the system state is defined by the set of wafers available for processing and the wafers' location, as well as the availability of the resources. The algorithm generates a search tree of possible schedules using *breadth–first–search*.

The nodes of the search tree represent handler moves and are divided into loading (picking up the wafer from the current position and putting it into a chamber) and unloading operations (positioning the handler in front of a chamber and removing the wafer from the chamber). A path leading from the root of the tree to one of the leaves describes the schedule for the handlers and therefore of the cluster tool.

Figure 4.5 illustrates the generation of the search tree for a cluster tool consisting of a load lock, three chambers, and one robot. We assume that a lot of three wafers is loaded into the load lock. Each wafer has to be processed in either Chamber 1 or Chamber 2, and then in Chamber 3. For sake of simplicity, the nodes of the search tree represent the number of wafers in the load lock resp. in the chambers, after the robot move has been executed. For example, ”<2 0 0 1>” means ”two wafers in the load lock, one wafer in Chamber 3”.

A node of the search tree is expanded using a method that generates all applicable continuations of that node. A feasible action is executed as soon as possible, leading to *active schedules*, cf. Section 4.1.3.

Pruning the Search Tree

If the generation of new search paths is not restricted, the search tree grows exponentially with the depth of the tree, leading to an explosive growth of the run–time of that algorithm, even for relatively simple models. By detecting transient and non–active parts of a schedule and avoiding the generation of sub–optimal paths, the search space can be reduced.

Detecting the Fundamental Period

Detecting the fundamental period in the search tree is an important means to exclude redundant or sub–optimal schedules from the search tree. The goal is to cyclically repeat the steady–state part of a schedule.

The implemented approach makes use of the fact that the cluster tool has reached a steady state if a series of handler moves is repeated, but no resource is continually occupied during the time period of that series. The Fundamental Period is then defined by that series of moves and can be

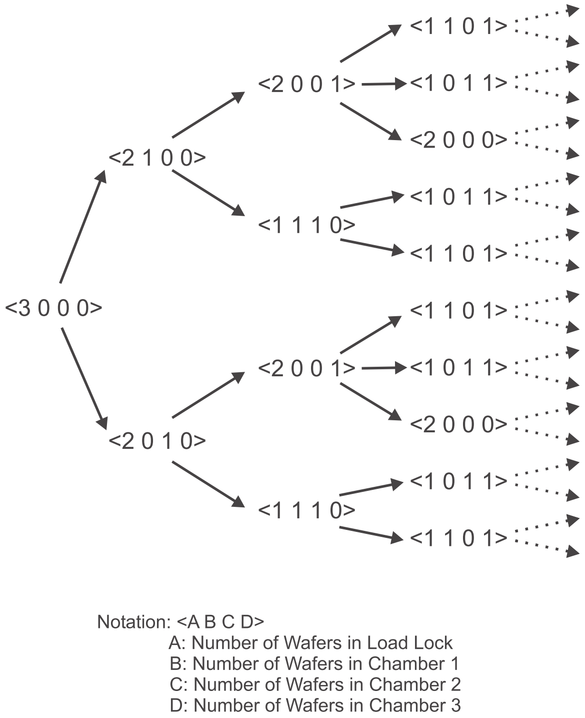


Figure 4.5: *Partial Search Tree*

repeated until no more unprocessed wafers are available and the cluster tool changes to completion phase.

It has to be noted, however, that in certain cases the continuation of a Fundamental Period is not the optimal solution. An example is given in Figure 4.6. It can be seen that using the schedule [b] leads to a cycle time that is 25 percent smaller than the cycle time of schedule [a].

In this example, wafers of type 1 are processed for one time unit in Chamber 1, Chamber 2 and again in Chamber 1. Type–2–wafers are pro-

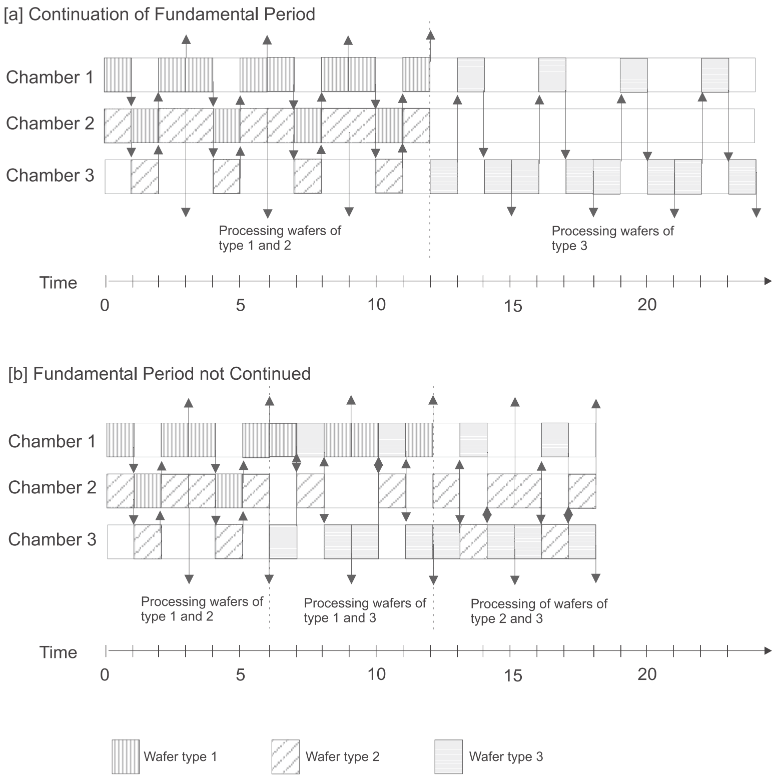

[a] Continuation of Fundamental Period
Chamber 1
Chamber 2
Chamber 3
Processing wafers of type 1 and 2
Processing wafers of type 3
Time
0
5
10
15
20
[b] Fundamental Period not Continued
Chamber 1
Chamber 2
Chamber 3
Processing wafers of type 1 and 2
Processing wafers of type 1 and 3
Processing of wafers of type 2 and 3
Time
0
5
10
15
20
Wafer type 1
Wafer type 2
Wafer type 3

Figure 4.6: *Effect of Continuing Fundamental Period*

cessed in Chamber 2, Chamber 3, and Chamber 2 for 1 time unit. Finally, wafers of type 3 have a processing time of 1 time unit in Chamber 3, Chamber 1 and Chamber 3.

To achieve schedule [b], significant computational overhead is necessary, because the state evolution of the cluster tool has to be stored. Therefore, this modified approach has not been implemented.

Detecting Non–Active Paths

To avoid generating sub–optimal schedules, it is important to detect handler moves leading to non–active schedules. Non–active schedules occur if a handler move is already part of another schedule, but in this schedule it appears at an earlier point in time. The Gantt chart in Figure 4.7[a] represents an active schedule. Chart [b] contains the same moves, but partially scheduled at a later time. Therefore, the corresponding schedule is non–active and does not need to be examined further.

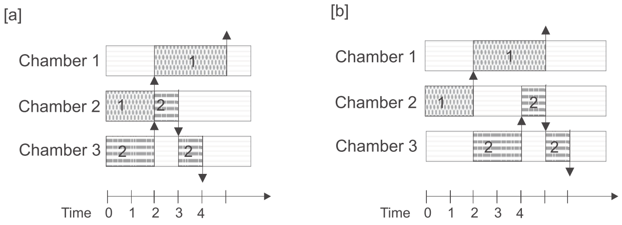


Figure 4.7: *Active [a] and Non–Active [b] Schedules*

Excluding Schedules with Deadlocks

Obviously, schedules leading to a deadlock situation can also be excluded from the search tree. Since it can be assumed that for each search path leading to a deadlock situation there exists at least one path without deadlocks, we did not attempt to resolve deadlock situations in the ExhaustiveSearch scheduler.

4.4.3 The Heuristic StepByStep Scheduler

Application

The StepByStep scheduler aims at generating small schedules, containing only a few robot moves. These sequences are generated by using *dispatch* *rules*. This keeps the computational complexity low and facilitates the use of this scheduler in real cluster tools.

Controlling a cluster tool using dispatch rules causes relatively myopic decisions, since the objective function is evaluated only for a small number of moves, not for the whole schedule. Furthermore, deadlocks can not be avoided when using dispatch rules. To generate more efficient schedules, it is necessary to apply strategies that facilitate a *look ahead* on future cluster tool states. One of these strategies is to place a robot that is currently idle in front of the process chamber that finishes processing next (*Fetch Wafer Look Ahead*), or to initiate the transport of a wafer to a process chamber that can be unloaded by another robot (*Put Wafer Look Ahead*). For the StepByStep Scheduler, several versions of Fetch Wafer Look Ahead have been implemented.

Algorithm

The algorithm used by the StepByStep scheduler that is run when a new robot move is generated can be described with the following rules. The scheduler will generate a new move according to the first of the following rules that applies:

1. Generate a move that transports the wafer currently on the robot to its destination. This situation can occur for example when a process chamber has failed and now returns to state ”up”.

2. If a wafer has been stored into a load lock or process chamber in order to resolve a deadlock situation, generate a move that returns the wafer into the process sequence.

3. Move a wafer waiting in a process chamber to be picked up.

4. Generate a move that takes an unprocessed wafer waiting in one of the load locks. Try to balance the recipe mix in case of a cluster tool with more than one load lock by alternating the load lock to pick from.

5. If a look ahead strategy is used, place the robot in front of the process chamber that will finish processing next.

Whenever a rule applies for more than one wafer, the scheduler picks a wafer according to one of the dispatch rules described in the following paragraph.

Implemented Dispatch Rules

Different *dispatch rules* for choosing the next wafer for transportation have been implemented in the StepByStep Scheduler. To avoid the starvation of wafers of a certain recipe, dispatch rules like *Shortest Process Time First* (SPTF) have not been considered.

- Least Work Remaining (LWKR)

  This anticipatory and dynamic rule chooses the wafer with the smallest remaining processing time. The goal is to minimize the flow factor of wafers.

- Fewest Remaining Operations (FRO) in Combination with LWKR

  Another anticipatory and dynamic rule similar to LWKR, FRO also tries to minimize the flow factor of wafers. In this case, the wafer with the smallest number of remaining recipe steps is chosen. If there is a draw situation between different wafers, the LWKR rule is applied.

- First Come First Serve (FCFS)

  First Come First Serve is commonly applied in queuing systems. The wafer that spent the longest time waiting in a chamber is chosen. The goal is minimizing the maximal waiting time of wafers.

- First Arrival in Shop (FAS)

  FAS is a dynamic rule that causes a behavior similar to LWKR. The goal is to expedite the processing of the wafer that spent the longest time in the cluster tool.

- Service in Random Order (SIRO)

  This rule is only intended for benchmarking purposes. It randomly selects one wafer of the set of available wafers for processing.

The presented rules have been tested in a simulation experiment with the cluster tool depicted in Figure 3.4. The model parameters are specified in Appendix B, the simulation results for the different objective functions are presented in Table 4.1. The simulation was run five times, each time for 200,000 seconds. The resulting performance characteristics were identical in all runs, because the simulation model contains no random elements. To estimate the influence of the dispatch rule on the computational complexity of the simulation, the average run time of the five simulation runs was computed.

Strategy	Avg. Run Time (sec)	Avg. Flow Factor of Wafers (Obj. Fct. 1)	Avg. Chamber Util. (Obj. Fct. 2)	Wafer Throughput (Obj. Fct. 3)
LWKR	152.4	1.41	0.217	7.33
FRO – LWKR	152.7	1.41	0.217	7.33
FCFS	152.4	1.47	0.204	6.88
FAS	136.6	1.50	0.203	6.81
SIRO	142.2	1.43	0.211	7.11

Table 4.1: *Comparison of Different Dispatch Rules*

The differences of the performance values for different objective functions are relatively small. We expected this result, because only a part of the scheduling decisions is made by the dispatch rules. Other decisions are independent of the choice of the dispatch rule. For example, if there are two lots with wafers of different recipes loaded, in all simulation runs the scheduler tries to give equal priority to the two recipes, i.e., over a given period of time, the number of wafers of the two recipes that are processed should be equal.

The rules FCFS and FAS seem to have a negative impact on the cluster tool productivity. On the other hand, LWKR and FRO–LWKR work well in the presented model. If the processing times of the individual recipe steps differ only slightly, the behavior under LWKR and FRO–LWKR is rather similar.

Implemented Look Ahead Strategies

Three different types of strategies for wafer fetching have been implemented in the StepByStep scheduler.

- *NoLookahead*: Handler moves are started only if the destination resources are not busy.
- *FixedLookahead*: An idle handler is placed in front of a processing chamber as soon as processing has started.
- *TemptativeLookahead*: Similar to FixedLookahead. However, if the handler has been placed in front of a busy process chamber, the decision to fetch the wafer in that chamber can be revised if it is more appropriate according to the chosen dispatch rule.

None of the presented alternatives is better than the others in all cases. For all strategies, it is possible to generate cluster tool configurations that lead to the best results under the respective strategy.

Table 4.2 displays the performance characteristics under the different look ahead strategies. Again, the simulated time was 200,000 seconds, the run time is the average of five runs. The dispatch rule applied was Fewest Remaining Operations (FRO) in combination with LWKR.

Strategy	Avg. Run Time (sec)	Avg. Flow Factor of Wafers (Obj. Fct. 1)	Avg. Chamber Util. (Obj. Fct. 2)	Wafer Throughput (Obj. Fct. 3)
NoLookahead	138.0	1.41	0.212	7.15
FixedLookahead	110.1	1.47	0.181	6.11
TentativeLookahead	152.4	1.41	0.217	7.33

Table 4.2: *Comparison of Different Look Ahead Strategies*

Depending on the chosen look ahead strategy, the run times of the simulations differ significantly, since the number of decisions that the scheduler has to generate is dependent on the look ahead strategy. When using TentativeLookahead, the run time is larger than for the other strategies, because each scheduling decision will be revised if a process chamber finishes processing before the wafer has been fetched. For FixedLookahead, we measured the smallest run times. The number of scheduling decisions is reduced with this strategy because when a chamber finishes processing, a handler is waiting in front of the chamber already to pick up the finished wafer and so no scheduling decision is required.

Handling Deadlocks

When using dispatch rules for scheduling a flow shop, deadlock situations can occur. Therefore, a simple deadlock detection and recovery mechanism has been implemented in the StepByStep scheduler.

If all of the following conditions apply during simulation, the cluster tool model is in deadlock:

- There are partially processed wafers in the process chambers or on the handlers.
- No handler is active.
- If a look ahead strategy is applied, all processing in chambers is finished.
- No wafer transportation is possible.

Obviously this mechanism can only detect deadlocks when they have occurred already. It is not possible to avoid a deadlock situation.

To recover from the deadlock situation, the wafer with the longest waiting time is placed back into the load lock. This approach increases the time the wafer spends in the system and, therefore, violations of residency time constraints or inter–process time constraints can occur. Furthermore, the recovery approach is also myopic, since a similar deadlock situation can occur shortly after the recovery.

The effects of deadlocks have been investigated using the cluster tool model in Appendix B. To this end, process steps two and three in recipe number three have been switched in order, so that deadlock situation can occur theoretically. Table 4.3 illustrates the impact of deadlocks on performance for a simulation time of 200,000 seconds.

Model	Avg. Run Time (sec)	No. of Dead-locks	Avg. Flow Factor of Wafers (Obj. Fct. 1)	Avg. Chamber Util. (Obj. Fct. 2)	Wafer Through-put (Obj. Fct. 3)
Original	152.4	0	1.41	0.217	7.33
Step 2 and 3 Switched	141.4	116	1.45	0.203	7.15

Table 4.3: *Performance Loss Caused by Deadlocks*

Optimizing the Scheduler

The deadlock detection can be performed more efficiently than with the presented approach. Using an entity–resource graph as proposed by Deuermeyer et al. (1997) seems to be an efficient alternative.

Another extension of the scheduler is to also implement a *put wafer look ahead*, similar to the fetch wafer look ahead.

4.5 Discussion

To provide a general algorithm that can guarantee to solve the cluster tool scheduling problem optimally seems to be impossible. Many side constraints occur in the manufacturing environment, and a high degree of flexibility should be achieved with the implemented scheduling algorithms. Therefore, most of the more complex approaches presented in literature are not applicable, because they are either too restrictive concerning the cluster tool configuration or require too much computational effort. However, if an heuristic approach is needed, both Branch–and–Bound algorithms and dispatch rules seem to be good candidates.

5 Work Load Distribution in Pools of Cluster Tools

As an application of the simulation model CluSim presented in Chapter 3, this chapter is dedicated to an optimization problem that is derived from an actual problem in a semiconductor manufacturing facility.

5.1 Problem Description

Consider a pool of $m \geq 1$ cluster tools. The m cluster tools can be identical or of different type. Each cluster tool can have an arbitrary number of load locks. It is assumed that the m cluster tools are initially idle and will start processing lots simultaneously. We assume further that there is a number of $n > 1$ lots available for processing. The n lots can be identical or of different types. There is no constraint concerning the recipes according to which the wafers in the lots have to be processed, besides the constraint that the lots can be processed on any of the m cluster tools available in the pool.[1] If a cluster tool has more than one load lock, we assume that parallel processing of any combination of lots on this cluster tool is possible.

The optimization problem consists of two tasks:

1. Find a partitioning of the n lots into m subsets, numbered $1, \ldots, m$. A subset can be empty.
2. For each subset $i, i = 1, \ldots, m$, find a sequence according to which the lots in the subset will be ordered.

The optimal partitioning and sequencing solution for processing the lots of all subsets $i, i = 1, \ldots, m$ on the respective cluster tool i in the sequence derived in Task 2 minimizes a given objective function. For example, the objective can be to minimize the cycle time of the n lots. Figure 5.1 depicts the problem for $m = 2$ and $n = 8$.

The number χ of solutions in the search space of this optimization problem is

[1] In an extension of this problem, this constraint has been removed, so that *tool dedication* can also be modeled.

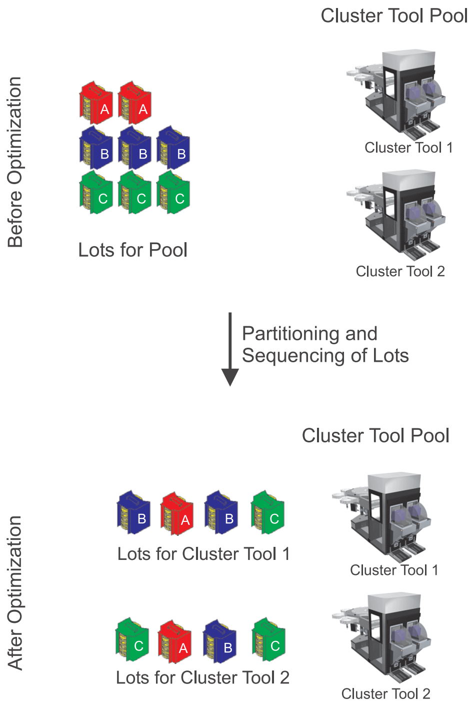


Figure 5.1: *Optimization Problem*

$$\chi = \frac{(n+m-1)!}{(m-1)!}\,. \tag{5.1}$$

We prove En. 5.1 by induction over the number of lots n. Assume the number of cluster tools $m \geq 1$ is arbitrary, but fixed. For $n = 1$ lots, En. 5.1 reduces to

$$\chi = \frac{m!}{(m-1)!} = m \tag{5.2}$$

which obviously is true, since the single lot can be processed on any of the m cluster tools.

Assume now that En. 5.1 is correct for a given $n > 1$. If we consider a valid solution of assigning n lots (of which there exist $\frac{(n+m-1)!}{(m-1)!}$ many according to our assumption), an additional lot can be inserted into the solution in the following ways. It either is processed immediately after lot $i, i = 1, \ldots, n$ on the same cluster tool on which lot i is processed, or it is processed as the first lot on any of the m cluster tools. Therefore, $n + m$ possibilities exist of inserting the $(n + 1)$st lot into the existing solution, which leads to

$$\begin{aligned} \chi &= \frac{(n+m-1)!}{(m-1)!}(n+m) \\ &= \frac{(n+m)!}{(m-1)!}, \end{aligned} \tag{5.3}$$

and finishes the proof.

It is obvious, that even for small instances of m and n, the search space of our optimization problem becomes very large. For example, for $n = 8$ lots and $m = 2$ cluster tools, there are $\chi = 362,880$ possibilities of processing the lots on the cluster tools. Exhaustive search of the solution space is not efficient, if at all possible. It is also questionable whether a human ”expert”is able to find the optimal solution, or even a solution that is close to the optimum, just by inspection.

The problem of scheduling n jobs on m parallel machines, where the processing times p_{ij} of job j on machine i is known, in order to minimize the makespan is NP–hard, i.e., no algorithm is known that solves this problem in polynomial time, cf. (Pinedo 2002). Since, in the problem we consider, parallel processing of jobs on one machine influences the processing time p_{ij} of that job, we can not assume that a polynomial–time algorithm exists to exactly solve the problem. Therefore, approximation algorithms and heuristics have to be applied.

In the following sections, an approach to this problem is presented. The approach is based on *Genetic Algorithms*, as they were introduced by John Holland and his colleagues at the University of Michigan, cf. (Holland 1992).

5.2 Genetic Algorithms

5.2.1 The Basic Idea

The basic idea of a Genetic Algorithm is to imitate evolutionary processes: The best individuals in a population survive and reproduce to pass on their genetic material to the next generation.

The goal is to simulate this evolutionary process by modeling the individuals or *chromosomes* as possible solutions to a given problem. The higher the *fitness* of an individual in terms of solving the problem, the higher the likelihood of the individual to be present in future generations.

In the most basic implementations of Genetic Algorithms, the solutions are encoded as strings of bits, cf. (Goldberg 1989). More advanced implementations of Genetic Algorithms use solutions that are encoded as strings of integers, characters, or more complex structures.

Roughly speaking, a Genetic Algorithm consists of three elements: A data structure to represent the chromosomes (e.g., strings of bits), operators on this data structure that allow the Genetic Algorithm to create new solutions, and an objective function to evaluate the fitness of a chromosome.

Genetic Algorithms have recently been applied for scheduling problems in general and in the area of semiconductor manufacturing. In (Bierwirth et al. 1995), an application of Genetic Algorithms to the deterministic job shop scheduling problem is presented, i.e., the jobs are released into the shop at predetermined points in time. The approach is extended to also handle non–deterministic job shops, where the release of jobs occurs at random points in time.

Yamada and Nakano (1995) introduce a new operator, called *multi–step crossover* and apply this operator to solve the job shop scheduling problem using a Genetic Algorithm. Utilizing the neighborhood structure of two parents p_1 and p_2, new solutions are successively generated by gradually adapting the characteristics of p_1 to those of p_2 instead of combining the characteristics of the parents. In preliminary experiments, they showed that the approach is at least comparable in performance to a *Simulated Annealing* approach.

Chen et al. (2001) use a Petri Net model of a semiconductor manufacturing system to model work in progress and machine status. A Genetic Algorithm is used to search for solutions to the scheduling problem. The solution found by the Genetic Algorithm is fed into a Petri–Net–based schedule builder to generate a near–optimal schedule.

5.2.2 Outline of the Basic Genetic Algorithm

In the context of solving optimization problems using Genetic Algorithms, the search space consists of so–called *phenotypes* or *individuals*. If, for example, the optimization problem is to minimize the makespan of a job shop, the phenotypes are production schedules, telling which job to process at what time on which machine.

Associated with each phenotype is a certain level of fitness. To determine the fitness, an *objective function* is defined. In our context, the objective function can be defined, for example, via the makespan that can be derived by simulation for a specific production schedule. The smaller the makespan, the higher the fitness of the individual.

To model the population of individuals on a computer, a coding scheme has to be developed that transfers those properties of individuals that are relevant for the optimization problem into a set of properties or parameters called the *genotype*. A specific instance of parameters is called a *chromosome* and a set of chromosomes is called a *population*.

Starting with a population of randomly created chromosomes, the Genetic Algorithm repeatedly creates new generations by selecting the chromosomes with the highest fitness from the existing population. By re–combining or *mating* these chromosomes, new chromosomes are generated that combine the properties of the parents. Random mutations of the genetic material create new chromosomes that possibly solve the objective better.

Different versions of Genetic Algorithms are described in literature. One possible implementation of a basic version of a Genetic Algorithm is presented in pseudo–code in Algorithm 1 and visualized in Figure 5.2.

Algorithm 1 Basic Genetic Algorithm

```
generate random population of $s$ chromosomes
**repeat**
  evaluate the fitness $f(x)$ of each chromosome $x$ in the population
  **repeat**
    - with probability $p_{Selection}$ select a chromosome with high fitness
    from the current generation and insert it into the new population
    - with probability $p_{Crossover}$ select two parent chromosomes from
    the current population according to their fitness (the better the fitness, the higher the chance to be selected). Mate the parents to form
    two new chromosomes, containing a re–combination of the genetic
    material of the parents. Insert the offspring into the new population.
    - with probability $p_{Mutation}$ mutate a chromosome in the new population by randomly selecting a parameter and modifying it.
  **until** (new population has size $s$)
**until** (end condition is satisfied, e.g., a certain fitness level is reached
by at least one chromosome in the current population)
return the best solution of the current population
```

The parameters s, $p_{Selection}$, $p_{Crossover}$, and $p_{Mutation}$ have to be adopted to the specific optimization problem. If the parameters were

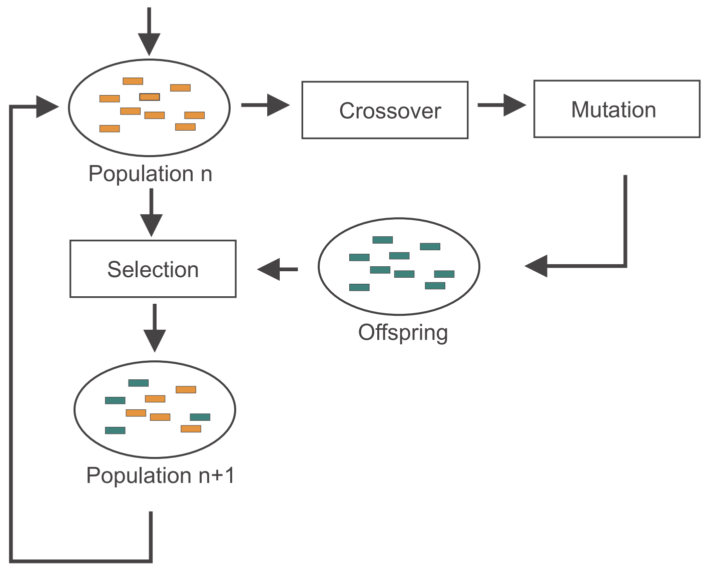


Figure 5.2: *Visualization of Genetic Algorithm*

chosen correctly, the algorithm will hopefully converge to an optimal or nearly–optimal solution. Genetic Algorithms usually are relatively robust concerning the parameter setting, so for a broad range of parameters the algorithm will produce satisfactory results.

Crossover

The *crossover* operation uses two chromosomes of a population to generate two new individuals. In its simplest form, the *one–point crossover*, both chromosomes are cut at the same position, see Figure 5.3. Now the tails of the two chromosomes are exchanged and are concatenated again. Hence, two strings of equal length are generated with new properties, combining properties of the parents.

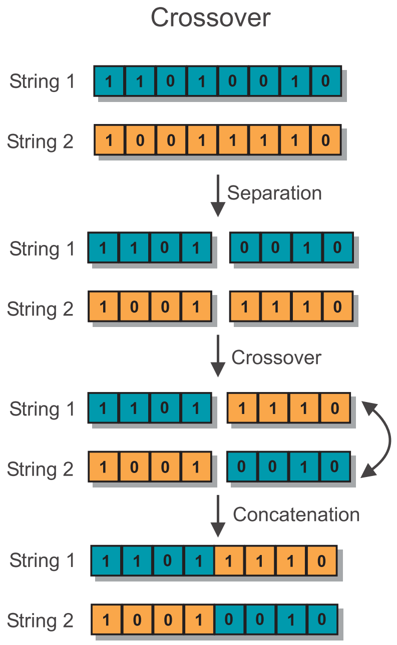


Figure 5.3: *Crossover Operation*

Mutation

The *mutation* operation simply chooses a position in the chromosomes and modifies the value at that position, see Figure 5.4. The aim is to introduce random modifications into the pool of solutions in order to avoid that the search for a solution stops with a suboptimal solution.

Convergence of the Genetic Algorithm

As a stopping criterion of the Genetic Algorithm, different measures can be used. Possible criteria are

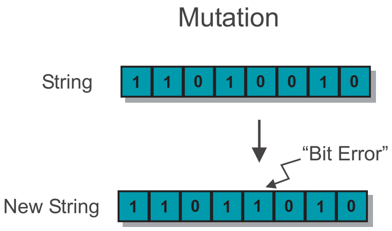


Figure 5.4: *Mutation Operation*

- Number of newly generated populations
  This criterion is especially useful during the parameterization of the Genetic Algorithm, when it is not known yet how long it takes for the algorithm to converge to a solution.

- Convergence of the population
  According to Beasley et al. (1993), the ”population is said to have converged when all of the genes have converged.”A gene (or parameter) has converged, if, for example, 95% of the chromosomes in the population share the same value of the parameter.

- A solution with the desired level of fitness has been found
  This criterion can only be applied if it is known that there exist solutions with at least the desired level of fitness.

If the number of generations is chosen large enough, the algorithm will eventually converge to a near–optimal or optimal solution, because only the best chromosomes are chosen for reproduction. Furthermore, exchanging parts from promising chromosomes often leads to even better solutions after the crossover operation, and the random element of mutation provides a small number of fresh strings in each generation. Convergence of the algorithm to the global optimum can not be guaranteed, however. Therefore, several methods have been presented that avoid *premature convergence* to local optima, cf. (Goldberg 1989).

Properties of Genetic Algorithms

As already mentioned, Genetic Algorithms have been applied successfully to a variety of optimization problems. This can be explained with the following properties:

- Robustness
  Genetic Algorithms can be applied to a broad range of problems and will generate optimal or nearly optimal solutions, independent of the topology of the solution space.
- Transparency
  Using simple standard implementations of the crossover and mutation operation, it is possible to apply a Genetic Algorithm to a optimization problem without having to know many details about the actual problem structure.
- Simple Operators
  The operations necessary to run a Genetic Algorithm generally do not require extensive computational power.
- Arbitrary Objective Functions
  The objective function can be chosen virtually arbitrarily. It is not necessary that derivatives of the objective function exist, and besides the definition, further information on the objective function is not required.
- Reduced probability to get stuck in local optima
  Since the mutation operation introduces a random element into the search for the optimum, new areas of the search space will be discovered. Therefore, it is unlikely that the search will stop in the neighborhood of a local minimum.
- Anytime algorithm
  Genetic Algorithms can be considered as an *anytime algorithm*. If the algorithm is stopped at an arbitrary point in time, it is possible

to return the best solution from the current population. Even if this solution might not be optimal, it still provides a feasible solution if it is needed urgently.

On the other hand, if an efficient algorithm is required, several aspects of the implementation of the Genetic Algorithm require specific attention. A coding scheme of the parameters has to be found that will lead to an efficient representation of the search space. Usually, the operations crossover and mutation have to be adapted to the specific problem so that the newly generated chromosomes are likely to have higher fitness than their predecessors.

Since there exist no rules about how to choose the parameters for a given optimization problem or for a given topology of the solution space, one has to experiment with the parameters like population size or probabilities for crossover and mutation if the convergence speed of the algorithm has to be improved. Finally, in some cases a large number of chromosomes has to be generated to find the optimum. Therefore, Genetic Algorithms are often combined with other optimization techniques, like *Local Search*.

5.3 Implementation of the Optimization Approach

To apply the Genetic Algorithm to the optimization problem presented above, we used the programming library ”GAlib”(Wall 1995). Since GAlib is written in C++, it could be easily integrated into the existing simulation tool CluSim, cf. (Dümmler 1999).

In this application of Genetic Algorithms, the chromosomes of the Genetic Algorithm are represented as follows. If n lots, numbered $1, \ldots, n$, have to be scheduled on m cluster tools, numbered $1, \ldots, m$, a chromosome 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. A sample coding for $n = 8$ and $m = 2$ is presented in Figure 5.5.

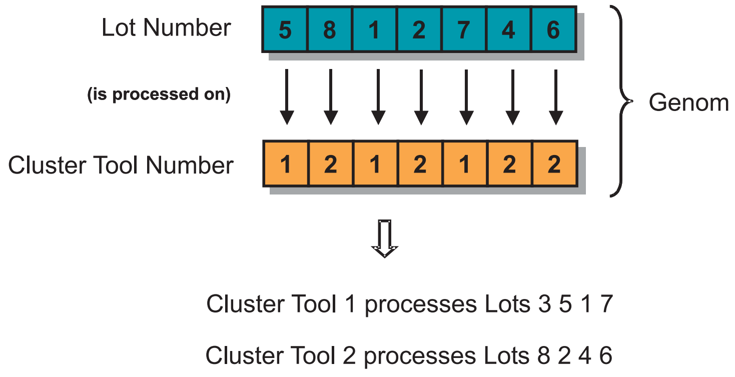


Figure 5.5: *Coding Scheme for GA*

Each chromosome in the initial population is generated by putting the n lots in a random order and randomly assigning a cluster tool to each of the lots.

The default operators on lists and arrays that are implemented and documented in the GAlib library to generate new chromosomes were applied after some modification. The modifications of the crossover and mutation operation were necessary in order to create consistent chromosomes.

To make sure that a crossover operation does not modify the set of lots encoded in the chromosomes, the following implementation is used. Firstly, lot–cluster tool pairs that appear in both parent chromosomes are identified and copied into the child chromosome. Then, the child lot is filled with the remaining lot–cluster tool pairs of the first parent. Figure 5.6 depicts the two steps of the formation of one child chromosome. The second child lot is generated in an analogous way by copying the remaining lot–cluster tool pairs of the second parent chromosome.

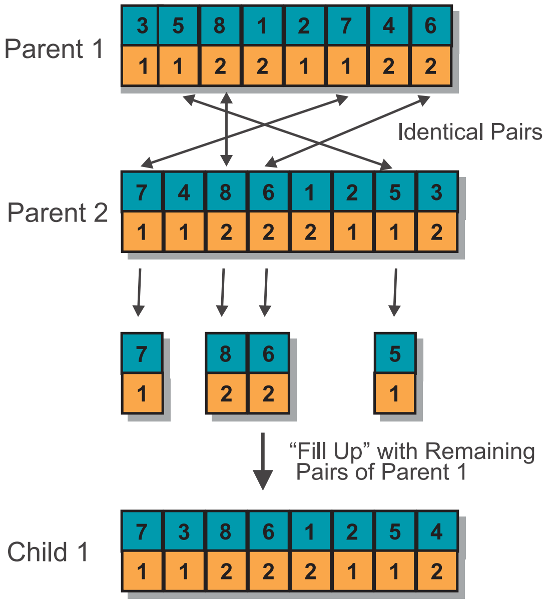


Figure 5.6: *Implementation of the Crossover Operation*

For the mutation operation, one has to distinguish whether it is applied to a position in the list of lots or to a position in the array of cluster tools. For the list of lots, a mutation is performed by switching the position of two genes, i.e., by switching the processing sequence of two lots. If applied to the array of cluster tools, a mutation is simply done by choosing another cluster tool for processing the respective lot.

As the objective function, the time required for processing all lots according to the sequence that the Genetic Algorithm suggests, the *makespan*, is used. This time is derived using the simulation model of the cluster tools. Each cluster tool is simulated in isolation, and the maximum of the makespans of the different cluster tools is used as the objective value. The Genetic Algorithm uses this objective value to evaluate a chromosome and to decide whether it is ”fit”enough to survive and reproduce. In order to avoid simulating the same schedule repeatedly and to speed up the optimization process, simulation results for a given schedule on a specific cluster tool are stored in an array.

5.4 Case Studies

The combination of a simulation model of cluster tools and a Genetic Algorithm was applied in two studies. The problem for the first case study is of a rather limited size, so it is possible to find the optimal solution by enumeration and the performance of the Genetic Algorithm can be evaluated. The second case study presents a more realistic problem size and therefore is able to prove the applicability of the presented approach in an actual semiconductor manufacturing environment.

5.4.1 Case Study 1

The cluster tool model under investigation in the first case study is depicted in Figure 5.7. It consists of two mainframes to which the individual processing chambers are attached. Transportation of wafers in the upper module is performed 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. Wafers from both load locks can be processed in parallel.

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 considered in our case study. We assume that lots of one recipe can be distinguished from each other, for example by an unique ID number.

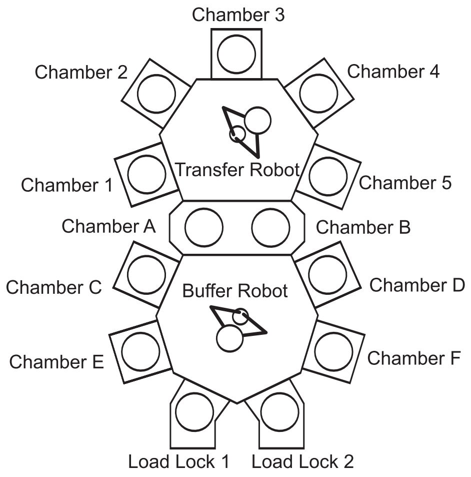


Figure 5.7: *Cluster Tool Model for Case Study 1*

Simulation studies have been conducted for this model for 30 different recipes. In the study presented in this monograph, we restrict the number of recipes to four. The processing times in seconds at each chamber are listed in Table 5.1. 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 because it is only used as a transfer chamber to the upper main module.

Table 5.1: *Recipes for Case Study 1*

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

Test Scenario 1

Three problem instances have been used to test the Genetic Algorithm. In the first problem instance, four lots, one of each recipe, 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. The optimal sequence has a makespan of 10031 seconds. The Genetic Algorithm was run five times for this problem, each time with a different randomly generated initial population. The respective parameters can be found in Table 5.2.

Table 5.2: *Parameters for Test Scenario 1*

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

The results are displayed in Table 5.3. For each of the five test runs, the best lot sequence that the Genetic Algorithm found is displayed. In the following columns, the makespan for the best sequence, the number of sequences tested to find the best sequence, and the total run time of the algorithm are given.

The Genetic Algorithm 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 algorithm needed to test only 14 sequences to find the optimal solution.

Test Scenario 2

In the second problem instance, two lots of each recipe, eight lots in total, are sequenced for processing on one cluster tool. The parameters are given in Table 5.4.

Table 5.3: *Results for Test Scenario 1*

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

Table 5.4: *Parameters for Test Scenario 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 5.5. For each run, the makespan of the best sequence that the Genetic Algorithm found is displayed. Since the search space for the optimal sequence consists of 40,320 possible solutions, the optimal solution can not be found within acceptable time. Therefore, we compare the makespan of the best sequence to the average makespan of 20 randomly generated sequences. The relative reduction in makespan is given in the third column. Finally, the number of sequences generated to find the best solution and the run time of the algorithm are displayed.

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

Table 5.5: *Results for Test Scenario 2*

Run	Best Make-span	% Improved	Sequen-ces tested	Run Time (sec.)
1	19562	12.5	114	42
2	19840	11.3	113	41
3	19601	12.3	117	42
4	20067	10.3	105	39
5	19691	11.9	115	43

Test Scenario 3

Finally, in the third problem instance three lots of each recipe are sequenced for processing on two cluster tools. In this scenario, the search space contains more than 6 billion solutions. The parameters used in this case are the same as in Table 5.4. The results of five test runs are displayed in Table 5.6. 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 5.6: *Results for Problem 3*

Run	Best Make-span	% Improved	Sequen-ces tested	Run Time (sec.)
1	14418	10.8	111	56
2	14182	13.0	100	52
3	13979	13.7	89	43
4	14059	14.9	111	56
5	13860	14.9	118	60

5.4.2 Case Study 2

In the second case study, we investigated a workcenter consisting of a pool of four cluster tools of the type depicted in Figure 5.8.

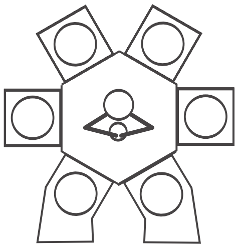


Figure 5.8: *Cluster Tool Model for Case Study 2*

Since the parameters for this case study were taken from a real manufacturing environment, historical data was available on the sequence of lots processed on this workcenter. Hence, the results generated by the Genetic Algorithm could be compared to the performance data of the real workcenter.

The historical data consisted of a sequence of 4,200 lots arriving at the workcenter within a timeframe of 20 days. Within the given timeframe, the average number of lots available for processing, i.e., the *work in progress* (WIP), was 37.

There were 23 different recipes according to which the wafers had to be processed. Since not every cluster tool was able to perform all 23 recipes, there was a rather high degree of dedication of recipes and cluster tools.

For our case study, we picked sequences of 40 lots from the historical data, starting with a randomly chosen lot. Then, by simulation, the makespan for each of these sequences was computed by processing the lots in the order given in the historical data. Finally, we compared this makespan to the makespan that resulted from processing the lots according to the best schedule generated by the Genetic Algorithm.

The respective makespans for five test runs are listed in Table 5.7.

Table 5.7: *Results for Case Study 2*

Run	Historical Make-span	Optimization Result	% Improved	Run Time (sec.)
1	17,835	15,015	15.81	37
2	16,125	14,116	12.46	41
3	19,186	15,757	17.87	45
4	19,839	17,110	13.76	36
5	18,362	15,962	13.07	40

The problem size χ in this case study, according to En. 5.1, is approximately 4.573×10^{26}.

The Genetic Algorithm produced sequences that lead to a reduction in cycle time of at least 12 %, i.e., about 30 minutes less cycle time. Given the small amount of run time of only a few seconds required to produce the optimized schedule, it is obvious that using the proposed approach, a significant improvement in cycle time can be achieved with relatively small effort.

5.5 Discussion

The run time of the presented Genetic Algorithm is small enough to make it useful in the actual dispatching of cluster tools. Schedules 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. If a result is needed before the Genetic Algorithm has terminated, the computation can be stopped and the Genetic Algorithm will respond with the currently best solution.

The interfaces to the optimization algorithm and simulation engine are rather simple and both programs require only data that is available in the manufacturing control system of the wafer fab. As a consequence, an integration of the programs into the manufacturing control can be easily implemented, leading to a significant performance improvement for the cluster tool pools.

Several extensions of the presented approach are possible. As an example, the crossover operation applied to generate new chromosomes can be adapted better to the actual optimization problem, so that generating schedules that apparently have a larger makespan than the existing ones is avoided.

6 Concluding Remarks

Cluster tools play an essential role in today's semiconductor manufacturing environments. The proliferation of this kind of equipment both in established 150 and 200mm wafer fabs as well as in modern 300mm fabs gives proof to the fact that, although these tools are very capital intensive and complex to handle, they are inevitable in the constant effort to make the production of semiconductors more cost effective, to improve the underlying production processes, to ensure on–time delivery, and to increase the quality of the finished products.

In this monograph, first the role of cluster tools in semiconductor manufacturing has been reviewed. An overview of different approaches of modeling cluster tools was given, with a focus on performance modeling and application in production planning. Furthermore, a generic simulation model of cluster tools was introduced. Then, the problem of scheduling the operations within a cluster tool was addressed and different solutions to this problem were presented. Two approaches implemented in the aforementioned generic simulation model were discussed. Finally, a Genetic Algorithm for optimizing the production schedule in a pool of cluster tools was introduced.

The aim of this work was to give an insight into the problems that arise with modeling cluster tools and with controlling their operations. A literature survey of modeling and scheduling cluster tools was given to help in further investigating into these topics. Some suggestions into which direction this research could be heading were given.

The main contribution of this study is to show possible starting points for the optimization of cluster tools. Two main aspects could be identified: The optimization of the scheduling and control of the internal cluster tool operations and the optimization of the production planning of cluster tool pools. It was shown that applying optimization methods can lead to significant performance improvements when using such sophisticated and capital intensive equipment.

In this study we showed that by using simulation it is possible to build generic cluster tool performance models that produce precise results and have execution times that facilitate the integration of these models in existing or future planning systems for semiconductor manufacturing environments. Using this approach, much more reliable predictions of cluster tool throughput and cycle times can lead to more precise production planning.

Concerning the scheduling of cluster tool operations it is obvious that a lot of optimization potential could be tapped if there were a closer cooperation between cluster tool manufacturers, end–users, and researchers. Due to the restrictive information policy of the equipment manufacturers, a lot of uncertainty exists on the end–user side concerning the actual capability of the cluster tools and on scientific side concerning the algorithms actually applied in cluster tools.

The optimization of production schedules using Genetic Algorithms has been applied successfully in this study. It is to be expected that similar areas of application exist in the semiconductor manufacturing environment, e.g., in the back–end production, and in other industries as well.

A Input File Grammar for CluSim

On the following pages, the grammar for defining a cluster tool model and the simulation parameters is presented. The input file in the following section was generated according to this grammar.

The grammar is given in *Extend Backus-Naur Form* (EBNF, cf. (Wirth 1977)), i.e., all literals are written in quotation marks and symbols on the left hand side are syntactic variables. Data types *String*, *Float* and *Int* are written in capital letters. Zero (0) is the default value for all time definitions, e.g., *PumpTime*, *LoadingTime* and *MoveTime*.

```
input_file          factory_block clustertool_block
                    recipe_block lot_block.
factory_block       ”Factory” STRING ”SimulationTime” number.
clustertool_block   clustertool_sec {clustertool_sec}.
clustertool_sec     ”ClusterTool” STRING [”NumberOfTools” INT]
                    sched_stmnt loadlock_sub chamber_sub
                    handler_sub.
interrupt_sub       interrupt_stmnt {interrupt_stmnt}.
interrupt_stmnt     ”NumberOfWafers”
                    STRING distribution distribution.
distribution        number
                    | disttype1 ”(” number ”)”
                    | disttype2 ”(” number ”,” number ”)”.
disttype1           ”exp”
                    | ”det”.
disttype2           ”norm”
                    | ”unif”
                    | ”erl”.
loadlock_sub        loadlock_stmnt {loadlock_stmnt}.
loadlock_stmnt      ”LoadLock” STRING [”PumpTime” number]
                    [”VentTime” number] [”MoveTime” number].
chamber_sub         chamber_stmnt {chamber_stmnt}.
chamber_stmnt       ”Chamber”
                    STRING [”PreparationTime” number]
                    [”AdaptationTime” number] [interrupt_sub].
handler_sub         handler_stmnt {handler_stmnt}.
handler_stmnt       ”Handler” STRING [”LoadingTime” number]
                    [”UnloadingTime” number] movetime_spec
                    [interrupt_sub].
movetime_spec       [”MoveTime” number]
                    | ”MoveTimeArray” ”(” matrix_header ”)”
                    matrix_body.
```

```
matrix_header         STRING {STRING}.
matrix_body           matrix_line {matrix_line}.
matrix_line           STRING matrix_entry {matrix_entry}.
matrix_entry          signed_number "/" signed_number.
sched_stmnt           "Scheduler" ("ExhaustiveSearch"
                      | "StepByStep" [lookahead_mode] [criterion_type]).
lookahead_mode        "NoLookAhead"
                      | "FixedLookAhead"
                      | "TentativeLookAhead".
criterion_type        "FIFOWaitingTime"
                      | "MaxRemainingProcessTime"
                      | "FIFOSystemEntering"
                      | "FewestRemainingSteps".
recipe_block          recipe_sec {recipe_sec}.
recipe_sec            "Recipe" STRING step_stmnt {step_stmnt}.
step_stmnt            "Step" STRING chamber_step_sub
                      {chamber_step_sub}.
chamber_step_sub      "ClusterTool" STRING
                      chamber_step_stmnt {chamber_step_stmnt}.
chamber_step_stmnt    "Chamber" STRING number.
lot_block             lot_sec {lot_sec}.
lot_sec               "Lot" STRING
                      "StartRate" distribution wafer_stmnt {wafer_stmnt}.
wafer_stmnt           "Recipe" STRING "NumberOfWafers" INT.
number            FLOAT
                  | INT.
signed_number     number
                  | "-1".
```

B Endura Model

The following model represents an actual cluster tool type encountered in today's wafer fabs. It contains two identical cluster tools, each consisting of two mainframes, one robot per mainframe and two chambers that are used to pass wafers from one mainframe to the other (chambers *A* and *B*). The cluster tool is displayed in Figure 2.9.

```
Factory TestFactory
    SimulationTime 15000

ClusterTool XEndura
    NumberOfTools 2
    Scheduler StepByStep TentativeLookAhead LeastWorkRemaining
    LoadLock LL1
    LoadLock LL2
        Chamber Chamber1
        Chamber Chamber2
        Chamber Chamber3
        Chamber Chamber4
        Chamber Chamber5
        Chamber ChamberA
        Chamber ChamberB
        Chamber ChamberC
        Chamber ChamberD
        Chamber ChamberE
        Chamber ChamberF
        Handler HandlerA
            MoveTime 1/20
Recipe Recipe1
        Step Step1
             ClusterTool XEndura
                        Chamber ChamberE 80
                        Chamber ChamberF 80
        Step Step2
             ClusterTool XEndura
                        Chamber ChamberC 60
        Step Step3
             ClusterTool XEndura
                        Chamber ChamberD 40
        Step Step4
             ClusterTool XEndura
                        Chamber ChamberA 0
        Step Step5
             ClusterTool XEndura
                        Chamber Chamber1 70
        Step Step6
             ClusterTool XEndura
```

```
                        Chamber Chamber2 40
        Step Step7
             ClusterTool XEndura
                        Chamber ChamberB 30
Recipe Recipe3
        Step Step1
             ClusterTool XEndura
                        Chamber ChamberE 80
                        Chamber ChamberF 80
        Step Step2
             ClusterTool XEndura
                        Chamber ChamberC 60
        Step Step3
             ClusterTool XEndura
                        Chamber ChamberD 40
        Step Step4
             ClusterTool XEndura
                        Chamber ChamberA 0
        Step Step5
             ClusterTool XEndura
                        Chamber Chamber2 90
        Step Step6
             ClusterTool XEndura
                        Chamber ChamberB 30
Recipe Recipe4
        Step Step1
             ClusterTool XEndura
                        Chamber ChamberA 0
        Step Step2
             ClusterTool XEndura
                        Chamber Chamber4 80
        Step Step3
             ClusterTool XEndura
                        Chamber ChamberB 30

Recipe Recipe2
        Step Step1
             ClusterTool XEndura
                        Chamber ChamberE 60
                        Chamber ChamberF 60
        Step Step2
             ClusterTool XEndura
                        Chamber ChamberA 0
        Step Step3
             ClusterTool XEndura
                        Chamber Chamber1 70
        Step Step4
             ClusterTool XEndura
                        Chamber ChamberB 30
Lot FirstLot
    StartRate 4000
    Recipe Recipe1
    NumberOfWafers 25

Lot SecondLot
    StartRate 4000
    Recipe Recipe2
    NumberOfWafers 25

Lot ThirdLot
    StartRate 4000
    Recipe Recipe3
    NumberOfWafers 25

Lot FourthLot
    StartRate 4000
    Recipe Recipe4
    NumberOfWafers 25
```

List of Figures

List of Tables

List of Algorithms

Bibliography

Ames, V. A., J. Gililland, J. Konopka, R. Schnabl, and K. Barber (1995). Semiconductor manufacturing productivity overall equipment effectiveness (OEE) guidebook. SEMATECH Technology Transfer #95032745A-GEN.

Applied Materials, Inc. (2003). Applied materials announces results for third fiscal quarter 2003; new orders increase to $1.05 billion; net sales of $1.09 billion. http://www.amat.com.

Atherton, L. F., R. W. Atherton, M. A. Pool, and F. T. Turner (1990). Performance analysis of multi-chamber manufacturing equipment for the semiconductor industry. Working paper, please contact the authors directly.

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

Aybar, M. and K. Potti (2002). Case studies in improving equipment productivity in TI's DMOS5 fab using ToolSim. In *Proceedings of the International Conference on Modeling and Analysis of Semiconductor Manufacturing (MASM)*, pp. 42–45.

Bader, M. E., R. P. Hall, and G. Strasser (1990). Integrated processing equipment. *Solid State Technology 33*(5), 149–154.

Beasley, D., D. R. Bull, and R. R. Martin (1993). An overview of genetic algorithms. *University Computing 15*(2), 58–69.

Bierwirth, C., H. Kopfer, D. C. Mattfeld, and I. Rixen (1995). Genetic algorithm based scheduling in a dynamic manufacturing environment. In *IEEE Conference on Evolutionary Computation*.

Bodner, D. A. and L. F. McGinnis (1997). Deadlock analysis of cluster tools. Slides.

Bohr, M. (1999). Schedulingverfahren für Cluster Tools in der Halbleiterfertigung. Master thesis, Chair of Distributed Systems, University of Wuerzburg.

Brooks Automation (2002). Brooks Automation web–page. http://www.brooks.com.

Busing, D. and R. C. Leachman (1998). Practical productivity metrics for flexible-sequence cluster tools. Draft.

Chandrasekaran, N. (1999). Operational models for evaluating the impact of process changes on cluster tool performance. Master thesis, Institute for Systems Research, University of Maryland.

Chen, J.-H., L.-C. Fu, M.-H. Lin, and A.-C. Huang (2001). Petri-Net and GA-based approach to modeling, scheduling, and performance evaluation for wafer fabrication. *IEEE Transactions on Robotics and Automation 17*(5), 619–636.

Coffman, E., M. Elphick, and A. Shoshani (1971). System deadlocks. *ACM Computing Surveys 3*, 67–78.

Conway, R. (1965). Priority dispatching and work in progress inventory in a job shop. *Journal of Industrial Engineering 16*, 123–130.

Crama, Y. and J. van de Klundert (1997). Robotic flowshop scheduling is strongly NP-complete. Communication with the authors.

Deuermeyer, B. L., G. L. Curry, A. T. Duchowski, and S. Venkatesh (1997). An automatic approach to deadlock detection and resolution in discrete simulation systems. *INFORMS Journal on Computing 9*(2), 195–205.

Dhudshia, V. H. and H. Clyde (1996). Cluster tool performance tracking. *Future Fab International 1*(1).

Dolman, D., Q. Wang, and J. Crowley (1999). Weighted configuration matrix approach to cluster tool metrics. In *Proceedings of the IEEE International Symposium on Semiconductor Manufacturing*, pp. 179–182.

Dümmler, M. (1999). Using simulation and genetic algorithms to improve cluster tool performance. In *Proceedings of the Winter Simulation Conference*, pp. 875–879.

Dümmler, M. (2000). Steuerung und Optimierung von Cluster Tools in der Halbleiterfertigung. In *Symposium on Operations Research (SOR 2000)*, pp. 295–300.

Gartner, Inc. (2003). Gartner says worldwide semiconductor market to experience 8 percent growth in 2003. http://www.gartner.com.

Geiger, C. D., K. G. Kempf, and R. Uzsoy (1997). A tabu search approach to scheduling an automated wet etch station. *Journal of Manufacturing Systems 16*(2), 102–116.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

Hendrickson, R. A. (1997). Optimizing cluster tool throughput. *Solid State Technology*.

Herrmann, J. W., N. Chandrasekaran, B. F. Conaghan, M.-Q. Nguyen, G. W. Rubloff, and R. Z. Shi (1999). Intergrating process models and operational methods. In *Proceedings of the International Conference on Semiconductor Manufacturing Operational Modeling and Simulation*, pp. 119–123.

Herrmann, J. W., N. Chandrasekaran, B. F. Conaghan, M.-Q. Nguyen, G. W. Rubloff, and R. Z. Shi (2000). Evaluating the impact of process changes on cluster tool performance. *IEEE Transactions on Semiconductor Manufacturing 13*(2), 181–192.

Herrmann, J. W. and M.-Q. T. Nguyen (2000). Sequencing wafer handler moves to improve the performance of sequential cluster tools. Technical Research Report 2000-3, Institute for Systems Research, University of Maryland.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems*. MIT Press.

Holt, R. (1972). Some deadlock properties of computer systems. *ACM Computing Surveys 4*(3), 179–196.

Huntley, D. (1990). Cluster tool communications: The path to an open standard. *Solid State Technology*, 85–88.

Kats, V., E. Levner, and L. Meyzin (1999). Multiple-part cyclic hoist scheduling using a sieve method. *IEEE Transactions on Robotics and Automation 15*(4), 704–713.

Kern, C. and N. Gerlich (1994). The simlib++ library. Technical report, University of Wuerzburg, Chair of Distributed Systems.

Kim, J.-H., L. Tae-Eog, and H.-Y. Lee (2002). Scheduling of dual-armed cluster tools with time constraints. In *Proceedings of the International Conference on Modeling and Analysis of Semiconductor Manufacturing (MASM)*, pp. 36–41.

Koehler, E. J., T. M. Wulf, A. C. Bruska, and M. S. Seppanen (1999). Evaluation of cluster tool throughput for thin film head production. In *Proceedings of the Winter Simulation Conference*, pp. 714–719.

Law, A. M. and W. D. Kelton (1991). *Simulation Modeling & Analysis* (2nd ed.). New York: McGraw–Hill.

LeBaron, T. H. and R. A. Hendrickson (2000). Using emulation to validate a cluster tool simulation model. In *Proceedings of the Winter Simulation Conference*, pp. 1417–1422.

LeBaron, T. H. and M. Pool (1994). The simulation of cluster tools: A new semiconductor manufacturing technology. In *Proceedings of the Winter Simulation Conference*, pp. 907–912.

Lemmen, B., E. van Campen, H. Roede, and J. Rooda (1999). Clustertool optimization through scheduling rules. In *Proceedings of the IEEE International Symposium on Semiconductor Manufacturing*, pp. 89–92.

Leung, J.-T. and E. Lai (1979). On a minimum cost recovery from system deadlocks. *IEEE Transactions on Computing 28*, 671–677.

Leventopoulos, M. M. (1994). A new class of petri nets for modeling, planning and scheduling of flexible manufacturing systems. Master thesis, Graduate School of the University of Maryland.

Lopez, M. J. and S. W. Wood (1996). Performance models of systems of multiple cluster tools. In *International Electronics Manufacturing Technology Symposium*, pp. 57–65.

Lopez, M. J. and S. W. Wood (1998). Systems of multiple cluster tools: Configuration and performance under perfect reliability. *IEEE Transactions on Semiconductor Manufacturing 1*(3), 465–474.

Mauer, J. and R. Schelasin (1994). Using simulation to analyze integrated tool performance in semiconductor manufacturing. *Microelectronic Engineering 25*(2 / 4), 139–146.

Mauer, J. L. and R. E. A. Schelasin (1993). The simulation of integrated tool performance in semiconductor manufacturing. In G. W. Evans, M. Mollaghasemi, E. C. Russell, and W. E. Biles (Eds.), *Proceedings of the 1993 Winter Simulation Conference*, pp. 814–818.

Morton, T. and D. Pentico (1993). *Heuristic Scheduling Systems*. Wiley Series in Engineering & Technology Management. New York: John Wiley & Sons.

Nehme, D. A. and N. G. Pierce (1994). Evaluating the troughput of cluster tools using event-graph simulations. In *IEEE/SEMI Advanced Semiconductor Manufacturing Conference*, pp. 189–192.

Nguyen, M.-Q. T. (2000). Improving cluster tool performance by finding the optimal sequence and cyclic sequence of wafer handler moves. Master thesis, Institute for Systems Research, University of Maryland.

Nguyen, M.-Q. T. and J. W. Herrmann (2000). Sequencing wafer handler moves to improve the performance of hybrid cluster tools. Technical Research Report 2000-31, Institute for Systems Research, University of Maryland.

Niedermayer, H. (2002). Approximation of lot cycle times for cluster tools in semiconductor manufacturing. Master thesis, Chair of Distributed Systems, University of Wuerzburg.

Oh, H. L. (2000). Conflict resolving algorithm to improve productivity in single-wafer processing. In *Proceedings of the International Conference on Modeling and Analysis of Semiconductor Manufacturing (MASM)*, pp. 55–60.

Pampel, S., J. Domaschke, and H. Jetter (2000). Productivity improvement for dry etch equipment through the application of simulation. In *International Symposium on Semiconductor Manufacturing*, pp. 79–83.

Paré, K., U. Dierks, and H. T. LeBaron (2002). Using simulation to understand and improve process tool efficiency. In *Proceedings of the International Conference on Modeling and Analysis of Semiconductor Manufacturing (MASM)*, pp. 46–51.

Pederson, D. E. and C. E. Trout (2002). Demonstrated benefits of cluster tool simulation. In *Proceedings of the International Conference on Modeling and Analysis of Semiconductor Manufacturing (MASM)*, pp. 58–63.

Perkinson, T. L. and R. S. Gyurcsik (1996). Single-wafer cluster tool performance: An analysis of the effects of redundant chambers and revisitation sequences on throughput. *IEEE Transactions on Semiconductor Manufacturing 9*(3), 384–400.

Perkinson, T. L., P. K. McLarty, R. S. Gyurcsik, and R. K. Cavin III (1994). Single-wafer cluster tool performance: An analysis of thoughput. *IEEE Transactions on Semiconductor Manufacturing 7*, 369–374.

Petri Nets Steering Committee (2003). Petri Nets World. http://www.daimi.au.dk/PetriNets.

Pierce, N. G. and M. J. Drevna (1992). Development of generic simulation models to evaluate wafer fabrication cluster tools. In *Proceedings of the Winter Simulation Conference*, pp. 874–878.

Pinedo, M. (2002). *Scheduling. Theory, Algorithms, and Systems* (2nd ed.). New Jersey: Prentice-Hall.

Poolsup, S. and S. Deshpande (2000). Cluster tool simulation assists the system design. In *Proceedings of the Winter Simulation Conference*, pp. 1443–1448.

ProModel Solutions (2002). ProModel web–page. http://www.promodel.com.

Rockwell Software Inc. (2002). Arena Software. http://www.arenasimulation.com/.

Rogatty, U. and F. Boebel (1996). 30% productivity increase of 16Mb-DRAM Gate-Conductor etching without additional investment. In *Proceedings of IEEE/SEMI Advanced Semiconductor Manufacturing Conference*, pp. 64–68.

Rostami, S. and B. Hamidzadeh (2002). Optimal scheduling techniques for cluster tools with process-module and transport-module residency constraints. *IEEE Transactions on Semiconductor Manufacturing 15*(3), 341–349.

Rostami, S., B. Hamidzadeh, and D. Camporese (2001). An optimal periodic scheduler for dual-arm robots in cluster tools with residency constraints. *IEEE Transactions on Robotics and Automation 17*(5), 609–618.

Ruppert, D., L. Schruben, and M. Freimer (2000). Meta-modeling of a cluster tool simulator. In *Proceedings of the International Conference on*

Modeling and Analysis of Semiconductor Manufacturing (MASM), pp. 67–77.

Schmid, M. (1999). Modellierung und Simulation von Cluster Tools in der Halbleiterfertigung. Master thesis, Chair of Distributed Systems, University of Wuerzburg.

Schömig, A. and J. Fowler (2000). Modelling semiconductor manufacturing operations. In *Proceedings of the 9th ASIM Dedicated Conference Simulation in Production and Logistics*, pp. 55–64.

Schruben, D. and L. Schruben (2000). *Event Graph Modeling Using SIGMA*. CustomSimulations.

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

SEMATECH, Inc. (2002). Sematech, inc. homepage. http://www.sematech.org.

SEMI (1996). *Standard for Definition and Measurement of Equipment Reliability, Availability, and Maintainability (RAM)*. SEMI.

SEMI (2002a). SEMI E21-94 (reapproved 1102) - cluster tool module interface: Mechanical interface and wafer transport standard. http://www.semi.org.

SEMI (2002b). SEMI E26-92 (reapproved 0699) - radial cluster tool footprint standard. http://www.semi.org.

SEMI (2002c). SEMI E38-1296 - cluster tool module communications (CTMC). http://www.semi.org.

Shannon, R. (1975). *Systems Simulation: The Art and Science*. Englewood Cliffs, NJ, USA: Prentice–Hall.

Shin, Y.-H., J.-H. Kim, H.-Y. Lee, and T.-E. Lee (2000). Modeling and implementation of a real-time embedded scheduler for CVD cluster tools. In *Proceedings of the International Conference on Modeling and Analysis of Semiconductor Manufacturing (MASM)*, pp. 78–82.

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

Singer, P. (1993). The thinking behind today's cluster tools. *Semiconductor International*, 46–51.

Singer, P. (1995). The driving forces in cluster tool development. *Semiconductor International*, 113–118.

Srinivasan, R. S. (1998). Modeling and performance analysis of cluster tools using petri nets. *IEEE Transactions on Semiconductor Manufacturing 11*(3), 394–403.

Tanenbaum, A. S. (2002). *Computer Networks* (4th ed.). Prentice Hall PTR.

Tracy, D. D. P. (2003). 2003 semiconductor capital equipment outlook. http://www.corning.com/semiconductormaterials/netnews/.

Venkatesh, S., R. Davenport, P. Foxhoven, and J. Nulman (1997). A steady-state throughput analysis of cluster tools: Dual-blade versus single-blade robots. *IEEE Transactions on Semiconductor Manufacturing 10*(4), 418–424.

Wall, M. (1995). GAlib. A C++ library of genetic algorithm components. http://lancet.mit.edu/ga/.

Wang, Q. and C. Christian (1998). Cluster tool equipment performance monitoring. *Future Fab International 5*, 275–277.

Wirth, N. (1977). What can we do about the unneccessary diversity of notation for syntactic definitions? *Communications of the ACM 20*(11), 822.

Wood, S. C. (1996). Simple performance models for integrated processing tools. *IEEE Transactions on Semiconductor Manufacturing 9*(3), 320–328.

Wood, S. C. (1997). Cost and cycle time performance of fabs based on integrated single-wafer processing. *IEEE Transactions on Semiconductor Manufacturing 10*(1), 98–111.

Wood, S. C. and K. C. Saraswat (1991). Modeling the performance of cluster-based fabs. In *Proceedings of IEEE International Semiconductor Manufacturing Science Symposium*, pp. 8–14.

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

Wu, N. and M. Zhou (2001). Avoiding deadlock and reducing starvation and blocking in automated manufacturing systems. *IEEE-TRA 17*(5), 658–669.

Wu, N. Q. (1999). Necessary and sufficient conditions for deadlock–free operation in flexible manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics; Part C: Applications and Reviews 29*(2), 192–204.

Yamada, T. and R. Nakano (1995). A genetic algorithm with multi–step crossover for job–shop scheduling problems. In *IEE/IEEE International Conference on Genetic ALgorithms in Engineering Systems: Innovations and Applications (GALESIA)*, Sheffield, UK, pp. 146–151.

Yim, S. J. and D. Y. Lee (1999). Scheduling cluster tools in wafer fabrication using candidate list and simulated annealing. *Journal of Intelligent Manufacturing 10*, 531–540.

Zant, P. v. (1996). *Microchip Fabrication: A Practical Guide to Semicondutor Processing*. McGraw–Hill.

Zuberek, W. M. (2000). Timed petri nets models of cluster tools. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pp. 3063–3068.

Zuberek, W. M. (2001a). Petri net modeling and performance analysis of cluster tools with chamber revisiting. In *Proceedings of the IEEE*

International Conference on Emerging Technologies and Factory Automation, pp. 105–112.

Zuberek, W. M. (2001b). Timed petri nets in modeling and analysis of cluster tools. *IEEE Transactions on Robotics and Automation 17*(5), 562–575.

Zuberek, W. M. (2001c). Timed petri nets models of multi–robot cluster tools. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pp. 2729–2734.

Index

ISSN 1432-8801