

Sequencing Wafer Handler Moves To Improve Cluster Tool Performance

Jeffrey W. Herrmann
Department of Mechanical Engineering
University of Maryland
College Park, Maryland 20742 USA

Abstract

Cluster tools can perform a sequence of semiconductor manufacturing processes. The sequence of wafer handler moves determines the time needed to complete a set of wafers. This paper presents scheduling problems for sequential and hybrid cluster tools and reviews the performance of several algorithms.

Keywords

Scheduling, cluster tools, semiconductor manufacturing.

1. Introduction

Manufacturing semiconductor devices involves three main steps: formation of p and n-type regions of the required conductivity within the semiconductor chip by doping; formation of reliable metal-semiconductor contacts on the surface of the chip; and encapsulation and packaging of the chip to provide protection and a convenient method of making electrical connection. In the first and second steps, the chips are processed together as wafers. Most operations process each wafer individually. However, identical wafers move together from one process to the next. Each set of wafers is a lot. The container used to move and store the wafers in a lot is called a cassette.

A cluster tool is a manufacturing system with integrated processing modules linked mechanically. Typical cluster tools include load locks that store cassettes of wafers (cassette modules), process modules that modify the properties of the wafers, and single or multiple wafer handler(s) that transport the wafers (transport modules). These modules are linked together by an evacuated transfer space. Because it has multiple chambers, a cluster tool can process multiple wafers simultaneously.

After a lot enters the cluster tool, each wafer must undergo a series of activities (stages). Each activity is performed in a different chamber. The wafer handler transports each wafer from one chamber to another. A sequential cluster tool has one chamber for each stage, so each wafer must visit every chamber. For example, the sequential cluster tool shown in Figure 1 has one load lock (LL), which stores the cassette of wafers (unprocessed and completed), and three process stages (S1, S2, S3). Each wafer starts in the LL, visits the first-stage chamber, visits the second-stage chamber, visits the third-stage chamber, and then returns to LL. Note that there are no locations to store wafers between process steps. A hybrid cluster tool will have one or more stages with at least two identical chambers that are used in parallel. In a hybrid tool, a wafer visits only one of the chambers at each stage.

The sequence of wafers leaving the LL is not important, since the wafers are identical, and an activity's time is the same for every wafer. But the sequence of wafer handler moves, which determines when each activity starts, will change the total time needed to process a lot of wafers. We use the term *lot makespan* to describe the total lot processing time. This paper addresses the problem of sequencing the wafer handler moves to minimize the lot makespan. Reducing the lot makespan can reduce cycle time, reduce tool utilization, and increase tool capacity. Moreover, the lot makespan is a necessary component for calculating overall equipment effectiveness (OEE) and cost-of-ownership (COO), which are usually used to evaluate cluster tool performance [1, 2].

Like machine tools, cluster tools use controllers that supervise the tool operations, monitor the tool status, and handle exceptions that abnormal events cause. Under normal operation, sequencing wafer handler moves is an important responsibility. After completing one move, the wafer handler will wait where it is (if no more wafers are ready to move) or start another move (if at least one wafer is ready). If multiple wafers are ready to be moved, the

cluster tool must decide which move the wafer handler will perform. In practice, cluster tools use a push dispatching rule or a pull dispatching rule to select the next move. The pull rule gives priority to the wafer that has fewer remaining process steps. The push rule gives priority to the wafer that has more remaining process steps.

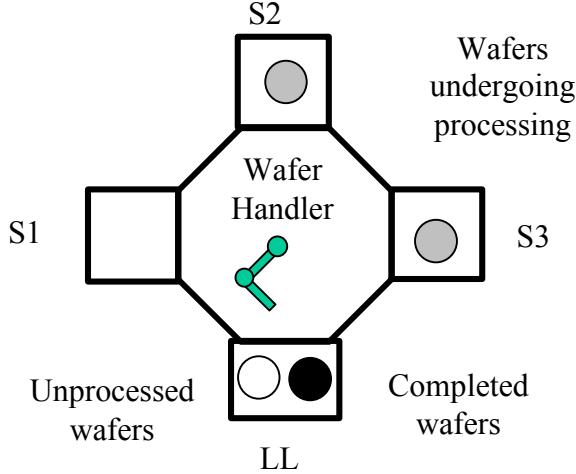


Figure 1. A three-stage sequential cluster tool.

Although useful, these dispatching rules do not guarantee that the resulting sequence has the optimal lot makespan for the given lot size, tool configuration, and activity processing times. Manufacturing facilities can improve cluster tool performance by determining a good sequence of wafer handler moves and providing it to the cluster tool controller, which can then use this sequence to direct normal operations. We will treat the problem as a deterministic machine scheduling problem, since the processing and move times have little variation, and small variations do not invalidate a given sequence.

Wood [3] derives formulas that relate the lot makespan to the number of wafers in the lot for ideal sequential and parallel tools. Considering the transitions at the beginning and the end of the lot, Perkinson *et al.* [4] derive a model that relates the lot makespan to the number of wafers. Both papers present linear models and identify two operating regions: in one region, the lot makespan is constrained by the wafer handling time; in the other region, by the module process time. Venkatesh *et al.* [5] analyze the capacity of a sequential cluster tool with a dual-blade wafer handler. They also identify conditions when the tool operation is constrained by the wafer handler. Srinivasan [6] presents more detailed Petri net models for sequential and parallel tools and uses these to determine the steady state behavior of the tool. Herrmann *et al.* [7] study the impact of process changes on cluster tool performance. To calculate lot makespan they use a network model for a prespecified sequence of wafer moves and cluster tool simulation software when the controller uses a dispatching rule or scheduling algorithm to sequence the wafer moves. Jeng *et al.* [8] study the problem of sequencing robot activities for a robot-centered parallel-processor workcell that processes a set of different jobs. They provide a branch-and-bound algorithm to find a sequence of robot activities that minimizes the makespan. None of the previous work addresses the problem of reducing the lot makespan by sequencing the wafer handler moves.

This paper focuses on single load lock, single wafer handler cluster tools. This paper formulates the wafer handler sequencing problem for sequential cluster tools and reviews the performance of several algorithms. The remainder of this paper is organized as follows. Section 2 formulates the problem and describes a branch-and-bound algorithm used to find solutions. Section 3 presents the results of computational experiments. Section 4 concludes the paper. Due to space constraints, the paper includes neither illustrative examples nor proofs. For more information about this problem, see Herrmann and Nguyen [9] and Nguyen and Herrmann [10].

2. Problem Formulation

The following information about the cluster tool scheduling problem is given. The cluster tool has one load lock (LL) and S stages ($S > 1$). In a sequential tool, each stage has one chamber, so the chambers are numbered 1 to S. Stage i has a wafer processing time p_i . The wafer handler move time is p_r . The lot has L identical wafers. Since each wafer must visit each stage and return to LL, the total number of moves is $L(S+1)$.

The sequence of wafers leaving LL is not important, since the wafers are identical. However, the sequence of moves affects the lot makespan C_{\max} , the total time needed to complete all moves. The scheduling objective is to minimize the lot makespan. By convention, scheduling problems are described by triplets of the form $\alpha \mid \beta \mid \gamma$. The α field describes the machine environment. We use $\alpha = CT1-1$ to describe two-stage sequential cluster tools and $CT1-1-1$ to describe three-stage sequential cluster tools. For our problem, the objective function $\gamma = C_{\max}$.

When processing begins, the wafer handler is at LL, and all of the wafers are unprocessed and in LL. For convenience, we will number the wafers in the order they leave LL. Let $R_{0,j}$ denote the move that takes wafer j from LL to stage 1. Let $R_{i,j}$ denote the move that takes wafer j from stage i to stage $i+1$ ($i = 1, \dots, S-1$). Let $R_{S,j}$ denote the move that takes wafer j from stage S to LL.

A feasible sequence of moves must satisfy the following constraints. All wafers must follow the fixed sequence of processing steps. For all $j = 1, \dots, L$, and $i = 0, \dots, S-1$, $R_{i,j}$ must precede $R_{i+1,j}$. Since there are no buffers (besides LL) to store wafers, the chamber at stage $i+1$ must be free before $R_{i,j}$ begins. That is, the wafer handler must have moved the previous wafer to the next stage. Thus, $R_{i+1,j-1}$ must precede $R_{i,j}$ for all $j = 2, \dots, L$ and $i = 0, \dots, S-1$.

The following facts describe the operation of the cluster tool. Each and every move requires the wafer handler. Since there is just one wafer handler, then, at any time, there is at most one move in process. The wafer handler cannot unload an empty or busy chamber and cannot load a busy or full chamber. (A full chamber has a wafer that has completed processing and is waiting to be moved.)

The chamber at stage i begins processing wafer j when move $R_{i-1,j}$ ends ($i = 1, \dots, S$). This activity cannot be interrupted until the chamber is finished processing the wafer. For example, if stage i starts processing at time t , then the chamber is busy during the interval $[t, t + p_i]$, and the wafer cannot be unloaded during that time.

Move $R_{i,j}$ starts when the chamber finishes processing wafer j and the wafer handler completes any previous move. $R_{i,j}$ requires p_r time units if the wafer handler is already at the chamber that processed wafer j (at LL if the move is $R_{0,j}$). $R_{i,j}$ requires $2p_r$ time units otherwise, for the wafer handler must move to the correct chamber before moving the wafer to a chamber at stage $i+1$ (to LL if the move is $R_{S,j}$). The wafer handler cannot make anticipatory moves. That is, the wafer handler cannot move to the chamber before processing ends. We assume that the wafer handler is already at LL when processing begins. Thus, $R_{0,1}$ requires p_r time units. For $j \geq 2$, $R_{0,j}$ requires p_r time units if and only if the previous move is $R_{S,k}$ for some $k < j$. For $i \geq 1$ and $j \geq 1$, $R_{i,j}$ requires p_r time units if and only if the previous move is $R_{i-1,j}$.

Special cases. We can identify two special cases. If $p_r = 0$, there is no scheduling problem since moves require no time, and all wafers move as soon as they are ready. If all $p_i = 0$, then an optimal solution is $R_{01}, R_{11}, \dots, R_{S1}, R_{02}, R_{12}, \dots, R_{S2}, \dots, R_{0L}, R_{1L}, \dots, R_{SL}$. This sequence has a lot makespan of $L(S+1)p_r$.

Cyclic sequences. Unless the lot size L is very small, a typical sequence has three phases, which we label filling-up, steady state (or cyclic), and completion. The chambers are empty when processing begins. Until the first wafer is completed, the tool is filling up with wafers. Then the tool is in a steady-state phase as it completes wafers and loads new wafers. When there are no more wafers to start, the tool enters the completion phase and completes wafers until the last wafer is unloaded from the last stage. Then processing ends.

Let us define a λ -unit cycle as a subsequence that loads and unloads each stage λ times and thus completes λ wafers. Complete sequences formed by repeating a cycle in the steady state and completion phase we will call λ -unit cyclic sequences. Note that the cycle does not define the filling-up phase, which ends with the first wafer being completed. Although there may exist more than one feasible filling-up phase for a given cycle, the class of cyclic sequences is small enough to enumerate for $CT1-1$ and $CT1-1-1$.

2.1 Cyclic Sequences for $CT1-1$

In this section, we analyze the cycle time and lot makespan of the two 1-unit cyclic sequences that are feasible for a two-stage sequential cluster tool. There are two feasible cycles, which we label σ_1 and σ_2 .

$$\begin{aligned}\sigma 1: & R_{2,j-1} - R_{0,j} - R_{1,j} - R_{2,j} \quad (j = 2, \dots, L), \\ \sigma 2: & R_{2,j-1} - R_{1,j} - R_{0,j+1} - R_{2,j} \quad (j = 2, \dots, L).\end{aligned}$$

Each cycle has only one feasible filling-up phase and thus forms exactly one cyclic sequence. $R_{0,1} - R_{1,1}$ is the filling-up phase for $\sigma 1$. $R_{0,1} - R_{1,1} - R_{0,2}$ is the filling-up phase for $\sigma 2$. For $x = 1$ and 2 , let P_x be the cycle time of σx , and let MS_x be the lot makespan of the cyclic sequence formed by σx . Then we can derive the following properties (see Herrmann and Nguyen [9] for the proofs):

Theorem 1. $P_1 = 3p_r + p_1 + p_2$, $MS_1 = LP_1$. $P_2 = 4p_r + \max\{2p_r, p_1, p_2\}$, $MS_2 = 3p_r + p_1 + p_2 + (L-1)P_2$.

Theorem 2. For CT1-1, if $p_r \geq p_1$ and $p_r \geq p_2$, then $P_2 > P_1$ and $MS_2 > MS_1$.

2.2 Cyclic Sequences for CT1-1-1

For a three-stage sequential cluster tool, there are six 1-unit cycles, which we denote $\sigma 1$ through $\sigma 6$. (Note these are the same six cycles that Sethi *et al.* [11] identify.)

$$\begin{aligned}\sigma 1: & R_{3,j-1} - R_{0,j} - R_{1,j} - R_{2,j} - R_{3,j}. \\ \sigma 2: & R_{3,j-1} - R_{0,j+1} - R_{2,j} - R_{1,j+1} - R_{3,j}. \\ \sigma 3: & R_{3,j-1} - R_{2,j} - R_{0,j+1} - R_{1,j+1} - R_{3,j}. \\ \sigma 4: & R_{3,j-1} - R_{1,j} - R_{2,j} - R_{0,j+1} - R_{3,j}. \\ \sigma 5: & R_{3,j-1} - R_{1,j} - R_{0,j+1} - R_{2,j} - R_{3,j}. \\ \sigma 6: & R_{3,j-1} - R_{2,j} - R_{1,j+1} - R_{0,j+2} - R_{3,j}.\end{aligned}$$

The cycle $\sigma 1$ has one feasible filling-up phase. Each of the other five cycles has two feasible filling-up phases. Thus, there are eleven feasible cyclic sequences for CT1-1-1. Herrmann and Nguyen [9] analyzed the lot makespan for each of these sequences and prove that the shortest 1-unit cycle does not necessarily form an optimal sequence for $CT1-1-1 \parallel C_{\max}$. Hall *et al.* [12] show that, if a mobile-robot cell has three machines and repeats cycle $\sigma 2$, then the cycle time is constant and that, for a mobile-robot cell that produces a single part type, repeating a 1-unit cycle dominates more complicated 2-unit cycles. However, neither result holds for CT1-1-1.

2.3 Branch-and-bound Algorithm

Although the cyclic sequences that use one-unit cycles are a natural class of solutions for the problem, they do not necessarily include an optimal sequence. Thus, we need to consider an algorithm that can generate an optimal sequence. Herrmann and Nguyen [9] describe a branch-and-bound algorithm (Algorithm BB) that begins by using the push dispatching rule to construct a feasible sequence and then using the pull dispatching rule to construct another feasible sequence. The smaller lot makespan becomes the initial upper bound on the optimal lot makespan. For each partial solution constructed, the algorithm creates a lower bound by calculating the completion time of the last scheduled activity. To avoid unnecessary searching, Algorithm BB uses a dominance criterion that limits Algorithm BB to the set of active schedules.

3. Experimental Results

Experiments to determine the performance of Algorithm BB studied how much computational effort was required and whether the algorithm generated sequences that were much better than the cyclic sequences. (See Herrmann and Nguyen [9] for the details.) In addition the experiments determined how lot size and relative move time affect the performance of Algorithm BB. Algorithm BB was compared to the unnamed heuristic that evaluates all feasible 1-unit cyclic sequences and selects the best one. (Note that evaluating the cyclic sequences requires little computational effort.) The push and pull dispatching rules were the benchmarks. (Note that Algorithm BB begins with these sequences).

The experiments were conducted using 18 problem sets of instances. Each problem set had ten randomly generated instances. Each problem set used different parameter values to generate its instances. The parameter values were chosen so that six problem sets (1, 2, 3, 10, 11, 12) contained instances with short move times and long processing times, another six problem sets (4, 5, 6, 13, 14, 15) contained instances with approximately equal move times and

processing times, and the remaining six problem sets (7, 8, 9, 16, 17, 18) contained instances with a long move time and short processing times.

For each instance, a heuristic found the lot makespan of the best 1-unit cyclic sequence. The branch-and-bound algorithm (Algorithm BB), the push dispatching rule, and the pull dispatching rule were used to generate solutions. However, Algorithm BB was stopped if it reached 100,000 nodes and used the best current sequence at that point. Thus, for some instances, Algorithm BB returned a suboptimal sequence.

For the two-stage sequential cluster tools, Algorithm BB required little effort except when the lot size was large and the move times were approximately equal to the processing times. However, the best cyclic sequence was optimal for most of these two-stage sequential cluster tool instances. As the move times increase (relative to the processing times), the optimal sequences were much better than those that the push and pull dispatching rules generated. Indeed, when the move times were long, the performance was dramatically better, as the relative improvement was approximately forty percent.

For the three-stage sequential cluster tools, Algorithm BB required little effort only when $L = 5$. Otherwise, it was usually unable to complete the search in 100,000 nodes. In these cases, the best cyclic sequence was often better. For these three-stage sequential cluster tool instances, as the move times increase (relative to the processing times), the cyclic sequences were much better than those that the push and pull dispatching rules generated. Again, when the move times were long, the performance was dramatically better, as the relative improvement was approximately forty percent. Algorithm BB generated sequences that were also improvements, but they were not as good as the cyclic sequences, except when $L = 5$.

The computing effort for Algorithm BB increased as the lot size increased. Conducting longer searches or using more complicated lower bounds did not improve search performance significantly.

4. Summary and Conclusions

This paper studied the sequential cluster tool scheduling problem. The goal is to improve tool performance by reducing the total lot processing time (which we call the lot makespan). This paper enumerated the class of one-unit cyclic sequences and discussed a branch-and-bound algorithm that can find optimal solutions. The paper described the results of experiments performed to understand the performance of the cyclic sequences and the branch-and-bound algorithm.

The results show that, for two- and three-stage sequential cluster tools, identifying the best cyclic sequence can, in some cases, dramatically reduce the lot makespan and thus improve cluster tool performance. The practice of using a push dispatching rule or a pull dispatching rule yields inferior performance. This is especially true when the move time and processing times are approximately equal and when the move time is longer than processing times. Compared to the cyclic sequences, the branch-and-bound algorithm requires additional computational effort and does not yield better solutions on the instances considered. The computational effort is sensitive to the problem size (the number of stages and wafers).

Sequencing wafer handler moves in hybrid cluster tools is a more complex problem, since there are parallel chambers in some stages (as shown in Figure 2). Nguyen and Herrmann [10] studied an algorithm (Algorithm TBB) to find the best λ -unit cyclic schedule. (λ is the minimum number of chambers in any stage.) This algorithm is called “truncated” because, unlike Algorithm BB, it stops adding moves to a sequence when the sequence has $\lambda + 1$ wafer completions. Algorithm TBB requires significantly less computational effort than Algorithm BB does. Compared to the sequences that current push and pull dispatching rules generate, we found that Algorithm TBB constructs sequences that reduce the lot makespan significantly. This is especially true when the move time and processing times are approximately equal and when the move time is longer than the processing times. Algorithm BB, however, requires excessive computational effort and cannot find better sequences (except for the smaller problem instances). Focusing on cyclic sequences reduces the search effort and yet yields very good sequences that improve cluster tool performance significantly. (Both the filling-up phase and the cycles must be considered when constructing a cyclic sequence.)

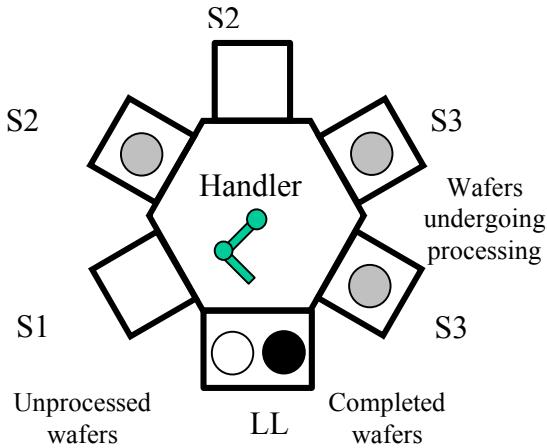


Figure 2. A CT1-2-2 Cluster Tool Configuration.

Acknowledgements

This research was supported by the Semiconductor Research Corporation and the National Science Foundation under grant DMI 97-13720. Mr. Mahn-Quan Nguyen conducted the numerical experiments using the facilities of the Computer Integrated Manufacturing Laboratory at the University of Maryland. Dr. Herrmann has a joint appointment with the Institute for Systems Research.

References

1. Dance, Daren L., Devid W. Jimenez, and Alan L. Levine, 1998, "Understanding equipment cost-of-ownership," *Semiconductor International*, July 1998, 117-122.
2. Murphy, Robert, Puneet Saxena, William Levinson, 1996, "Use OEE; don't let OEE use you," *Semiconductor International*, Sep. 1996, 125-132.
3. Wood, Samuel C, 1996, "Simple performance models for integrated processing tools," *IEEE Transactions on Semiconductor Manufacturing*, 9(3), 320-328.
4. Perkinson, Terry L., Peter K. McLary, Ronald S. Gyucsik, and Ralph K. Cavin, 1994, "Single-wafer cluster tool performance: an analysis of throughput," *IEEE Transactions on Semiconductor Manufacturing*, 7(3), 369-373.
5. Venkatesh, Srilakshmi, Rob Davenport, Pattie Foxhoven, and Jaim 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.
6. Srinivasan, R. S, 1998, "Modeling and performance analysis of cluster tools using Petri nets," *IEEE Transactions on Semiconductor Manufacturing*, 11(3), 394-403.
7. Herrmann, Jeffrey W., Niranjan Chandrasekaran, Brian F. Conaghan, Manh-Quan T. Nguyen, Gary W. Rubloff, and Rock Z. Shi, 2000, "Evaluating the impact of process changes on cluster tool performance," *IEEE Transactions on Semiconductor Manufacturing*, 13(2), 181-192.
8. Jeng, Wu-De, James T. Lin, and Ue-Pyng Wen, 1993, "Algorithms for sequencing robot activities in a robot-centered parallel-processor workcell", *Computers Ops. Res.* 20(2), 185-197.
9. Herrmann, J.W., and M.-Q. Nguyen, 2000, "Sequencing wafer handler moves to improve the performance of sequential cluster tools," Technical Report 2000-3, Institute for Systems Research, University of Maryland, College Park.
10. Nguyen, M.-Q., and J.W. Herrmann, 2000, "Sequencing wafer handler moves to improve the performance of hybrid cluster tools," Technical Report 2000-31, Institute for Systems Research, University of Maryland, College Park.
11. Sethi, S. P., C. Sriskandarajah, G. Sorger, J. Blazewicz, and W. Kubiak, 1992, "Sequencing of parts and robot moves in a robotic cell," *Int. J. Production Res.*, 4, 331-358.

12. Hall, Nicholas G., Hichem Kamoun, and Chelliah Sriskandarajah, 1997, "Scheduling in robotic cells: classification, two and three machine cells," *Operations Research*, 45(3), 421-439.