# An Optimal Scheduling Technique for Dual-arm Robots in Cluster tools with Residency Constraints

*Shadi Rostami*[*], *Babak Hamidzadeh*[*], *Dan Camporese*[**]

* Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC V6T 1Z4, Canada.
Emails: shadi@ece.ubc.ca, babak@ece.ubc.ca.
** Brooks Automation Corp., Richmond, BC V6V 2X3, Canada. Email: dan.camporese@brooks.com.

## Abstract

This paper discusses a scheduling technique, for cluster tools, that addresses post-processing residency constraints and throughput requirements. The residency constraints impose a limit on the post-processing time that a material unit spends in a processing module. The technique searches in the time and resource domains for a feasible schedule with a maximum throughput. Several heuristics are designed and added to reduce the complexity of the scheduling algorithm. The resulting schedules are deadlock free, since resources are scheduled according to the times that they are available. Analytical and experimental analyses demonstrate the correctness and efficiency of our proposed technique.

## 1. Introduction

Cluster tools provide a flexible, reconfigurable and efficient [1][2] environment for several manufacturing processes (e.g., semiconductor manufacturing). The tool is reconfigurable because components responsible for different processing steps, can be replaced with others to reconfigure the system. Flexibility of cluster tools means that it can support multiple manufacturing processes in one configuration.

A cluster tool consists of a number of processing modules (PM), a transport module (TM), and one or more cassette modules (CM). Raw material (e.g., wafers) enters the system through the CM, which interfaces with the environment external to the tool. The TM, which can consist of one or more robot arms, moves the material from one module to another. After all the necessary process steps are done on one material unit (e.g., a wafer), the unit will be returned to the CM that it has been originated from.

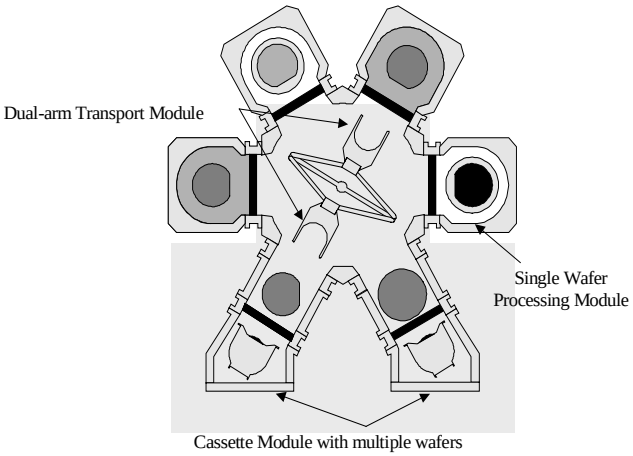


Fig. 1: A Typical Cluster tool

When all the material units of a loadlock are processed, the loadlock will be unloaded and replaced with another loadlock containing a new set of raw units.

Scheduling a cluster tool involves specifying a sequence of TM actions and their times, applied to material units, to move the units between PM's in order to satisfy a number of constraints and objectives. Due to stringent timing and throughput requirements, the scheduling problem in these tools can be quite complex. An important timing requirement, referred to as post-processing residency constraint, is a limit imposed on the post-processing time that a material unit spends in a PM.

In this paper, we present a scheduling technique that addresses post-processing residency constraints and throughput requirements in a cluster tool. The technique searches in the time and resource domains for a feasible schedule with a maximum throughput. It operates in two main phases. In the first phase, a simple periodic schedule is computed with low complexity. For a large number of problem instances, the simple periodic schedule feasibly solves the problem. If a feasible schedule is not found in the first phase, the scheduler enters phase two (with a higher degree of complexity) to compute a feasible schedule. During this phase, the periodic nature of the schedule is preserved. The scheduler incrementally increases the period only if necessary, to keep the throughput at a maximum. Several heuristics are designed and added to reduce the complexity of the scheduling algorithm. The resulting schedules are deadlock free, since resources are scheduled according to the times that they are available. Analytical and experimental analyses are presented to demonstrate the correctness and efficiency of the proposed technique.

The remainder of the paper is organized as follows. Section two describes the model and provides definitions for the terms used throughout the paper. Section three discusses the related work. Section four presents the scheduling algorithm and some of its properties. Section five provides the results of our experimental evaluation. Section 6 provides some concluding remarks.

## 2. Model and Definitions

A manufacturing process consists of a number of process visits, which together form a route. We only model a single wafer cluster tool, which means that each PM can only process one wafer at a time. A process visit ($V_i$) consists of one or more identical PM's. To manufacture one product unit, all visits in a route are assumed to be performed at least once. A multi-visit route is the one in which the processing steps of some visits are performed more than once in the same route. A multi-route configuration in a cluster tool is the one that allows multiple routes to exist in the tool at the same time (i.e. it supports manufacturing of more than one product in the same configuration). In a single-route and single-visit cluster tool with N visits, two numbers are assigned to each $V_i$. The process time, $t_{proc_i}$, shows the time that a wafer must remain on one of the $PM_{i,j}$'s to be completely processed. The residency number, $t_{res_i}$, shows the maximum time that the wafer can stay in that PM after it has been processed. If a wafer leaves one of the $PM_{i,j}$'s before $t_{proc_i}$, it will be premature, and if it stays more than $t_{proc_i} + t_{res_i}$, it will violate its post-processing residency constraint. Having process visits that consist of multiple identical PM's represents the notion of parallelism in a route. For each $V_i$ that has $M_i$ identical PM's, an effective process time, $t_{eff_i}$, is defined. For a single-module visit, this time is equal to $t_{proc_i}$, whereas for a parallel-module $V_i$, it is equal to the processing time of a single module that has the same effect on cluster tool throughput (analytical study) as all the parallel modules of $V_i$.

A TM transfers wafers from one module to another. It performs three kinds of actions: *pick* wafers from a source module (in time $t_{pick}$), *place* wafers into a destination module (in time $t_{place}$), and *rotate* in order to reach a source or destination module. This module can consist of one or more robot arms. Dual arm TM's have two blades pointing in opposite directions each capable of carrying a wafer (see Figure 1). The two blades are tightly coupled by construction, i.e., at anytime only one of the two blades can pick or place a wafer from or into a module [3]. One of the advantages of dual-arm TM is that one of the arms can be used as a temporary buffer.

The CM's are assumed to be loaded alternately into the system so that while the wafers in one CM are being processed, the other CM is being loaded into the system. Using this model, we can assume that at any time wafers exist that are ready for processing. Our Objective is to give an optimal schedule for a single-visit, single-route, single-wafer, and dual-arm cluster tool with parallel-module visits.

One of the common metrics of cluster tool performance is throughput. It has been defined as the processing rate of the cluster tool, usually expressed in the number of wafers completed per hour. We are looking for a periodic solution with a fundamental period (FP), which means that if we find the times that a wafer should enter and leave visits in a route, those relative times will be repeated for the subsequent wafers in the following periods. Each periodic schedule has an initial, steady and terminate state. The initial and terminate states are transients. Because we use two CM's, we can assume that there is always a wafer ready to be processed, so the initial and terminate transient states would be small compared to the steady state. We are looking for a periodic schedule that has the best throughput in the steady state (i.e. a schedule with the minimum FP). We define the time that takes a wafer to be completely processed and returned back to the CM as $t_{total}$. We prefer a schedule with smaller FP, although it may have bigger $t_{total}$. The amount, $t_{add}$, by which we increase $t_{total}$ is equal to the time that each wafer remains idle either on PM's or TM's. To represent a schedule we define all the times that the first wafer ($W_1$) enters and leaves a visit, so $E_i$ and $L_i$ show the times that $W_1$ enters and leaves $V_i$. $L_0$ is the time that it leaves the CM, and $E_{N+1}$ is the time that it is placed back into the CM. The other wafers enter and leave visits at the same relative times as the first wafer with a multiple of FP added to all the times. So for wafer $W_i$, we must add i * FP to all the $W_1$ times. In a single-route cluster tool, because all the wafers go through the same route, the visit whose effective processing time is more than the others is important in determining the throughput of the system. We call this visit the bottleneck. The bottleneck time, $t_{BN}$, is equal to $t_{eff_i}$ of that visit. Figure 2 shows the timing diagram of a cluster tool that has three single-module visits. Schedule of the first and second wafer ($W_1, W_2$) is shown. Filled boxes show the time intervals during which the wafer stays in the PM of $V_i$ which is equal to $t_{proc_i}$ of that visit in this diagram. After that time, the wafer is moved to the next visit. Each of these boxes with its start and end arrows is called a block ($t_{Block_i} = t_{proc_i} + t_{pick} + t_{place}$). The arrows show the transport times. The arrow at the beginning of the block represents a place action, and the arrow at the end of the block represents a pick action. We define $B_{i,j}$ to be the jth block of $W_i$. $B_{i,0}$ and $B_{i,N+1}$ represent a pick from, and a place into the CM's.

## 3. Related works

There has been some important effort in modeling and finding the optimal schedule of a cluster tool. All models focus on modeling a single-wafer cluster tool, which is both single-route and single-visit, but none address residency constraints in their models.

Perkinson [4] developed a model for the cluster tool with a single-arm TM, and assumed that all visits are single module (i.e. no parallelism). In his model, he provides a periodic schedule. The operations of the system are divided into two different regions. The first one happens when the TM is always busy with moving wafers. In this region, which is called the transport-bound region, the period of the system is a function of transport time ($t_{pick}$, $t_{place}$), and N (Number of visits) in the system. It has been proven that change in $t_{BN}$, while the system remains in this region does not affect the period. Assuming that $t_{pick}$ = $t_{place}$ = T, it has been shown that in this region the best possible FP is 2T(N+1).

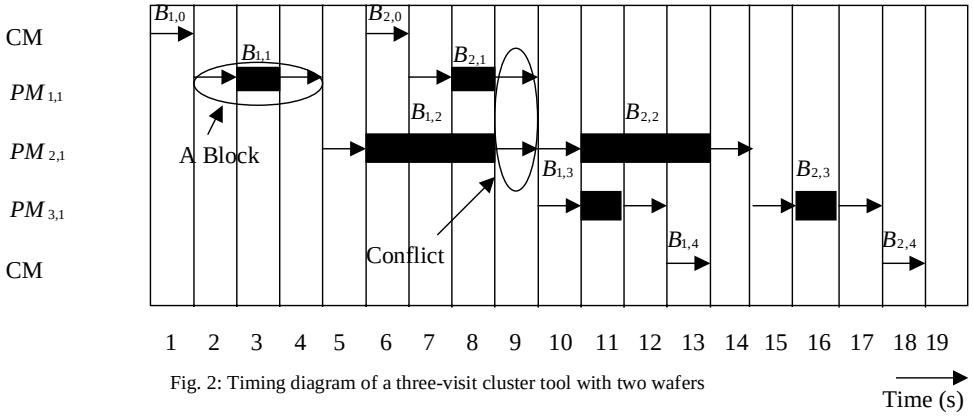


Fig. 2: Timing diagram of a three-visit cluster tool with two wafers

The second operating region occurs when the TM has some idle time. In this process-bound region, $t_{BN}$ determines FP., and the best possible FP is $t_{BN}$ + 4T. N, T, and $t_{BN}$ determine whether the system is in the transport-bound or process-bound region.

The given fundamental periods are the best possible one in those cases. To prove that a feasible schedule can be found with these periods, a very simple but effective schedule has been given. Due to the fact that none of the modules have residency constraints, if we force wafers to remain in each PM for a time equal to FP – 4T, and schedule moves one after another[1], there won't be any conflict in the schedule.

Wood and Perkinson [5][6] presented a model that handles parallel-module visits as well. Perkinson [6] found an equivalent single PM that has the same effect on the system throughput as all the parallel modules in that visit and its processing time can be calculated as: $t_{eff_i} = \frac{t_{proc_i} + 4T}{M_i} - 4T$ .

Venkatesh [3] proposed the optimal solution for a dual-arm cluster tool. He introduced a new action called *exchange*. In this model when $W_i$ is moved from one visit to another (from $PM_{j,l}$ to $PM_{j+1,m}$), the destination can be full (e.g. occupied by $W_k$). TM will pick $W_i$ from $PM_{j,l}$ with one arm, goes to the $PM_{j+1,m}$, picks up $W_k$ with the other arm and places $W_i$ in $PM_{j+1,m}$. By this action, we can improve cluster tool throughput when it is in the process-bound region. The FP in this situation becomes $t_{BN}$ + 2T.In this periodic schedule we have to force the wafers to remain on each PM for a time equal to FP – 2T.

Although we have not found references to any work that addresses residency constraints in cluster tools, there exists literature that deals with manufacturing processes with a form of residency constraint [7][8][9][10]. They refer to this form of residency constraints as time-window constraints. In [7][8][9] authors use only one material handler (as a TM with one robot arm). In [10] two material handlers are used, however, they operate in a completely disjoint manner from each other. This prevents their algorithm from using the material handler as a temporary buffer. Also, they consider the action of *picking* from one process module, moving to the destination, and *placing* the material in the destination module as an atomic action (i.e. they do not consider *exchange* [3] as a possible TM sequence of actions). In our model, we consider *pick*, and *place* to be different actions, so wafers can stay on an arm after being picked up and before being placed in the destination. In such a model we can use the arms as buffers. In fact, our model extends the *exchange* operation to its general form where in the interval between *picking* a wafer and *placing* it in its destination there might exist more than one TM move.

[1] Moves will be like this in each period : Move from CM to $V_1$, Move from $V_N$ to CM, Move from $V_{N-1}$ to $V_N$, …, Move from $V_1$ to $V_2$

## 4. Scheduling Algorithm

Figure 3 shows an overview of the algorithm. For any single-

1. Calculate FP
2. While not Done
   2.1 Schedule ← get-periodic-schedule(FP)
   2.2 IF conflict-free(Schedule) Done
   2.3 Schedule ← Search (schedule)
   2.4 IF conflict-free(Schedule) Done
   Else FP++;
3. End While

Fig. 3: Algorithm overview

module $V_i$, FP must be greater than or equal to its $t_{Block_i}$, for parallel-module visits it must be greater than or equal to the effective block time of the equivalent single module visit. Wafers are assigned to parallel modules of a visit in a round-robin fashion. So in visit $V_i$ with three PM's, wafer 1 is assigned to $PM_{i,1}$, wafer 2 to $PM_{i,2}$, wafer 3 to $PM_{i,3}$, wafer 4 to $PM_{i,1}$, and so on. Considering the definition of $t_{BN}$, the best possible period is equal to $t_{BN} + t_{pick} + t_{place}$. It is the minimum period because any smaller period will be less than the bottleneck block. Line 1 of Figure 3 calculates the best FP. In line 2.1.a simple periodic schedule is made. Figure 4 shows the algorithm to make the timing of a simple periodic schedule, and Figure 2 shows it's timing diagram (only 2 wafers are shown here).

In a feasible schedule these conditions must be satisfied:

1. The time a wafer spends in a PM must be lower bounded by $t_{proc_i}$ and upper bounded by $t_{proc_i} + t_{res_i}$.
2. At each time there must be at most one wafer in each PM.

3. It must be a conflict-free schedule. In a conflict-free schedule, TM's are scheduled such that no two transport actions are scheduled at one time, and at most two wafers reside on the two arms.

In the simple periodic schedule, condition one and two are satisfied, but conflicts may occur between the time slots during which TM's are scheduled to move wafers from one module to another. An example of such conflict is shown in Figure 2.

If the initial schedule is not conflict-free, we try to search and find a conflict-free solution. In our search we start from that simple periodic schedule and try to find a conflict-free solution with that FP, but with a larger $t_{total}$. If no solution can be found with that FP, it would be increased one unit and we try to search for a feasible solution with the new FP.

1. $L_0 = t_{pick}$
2. For i = 1 to N Do
   2.1. $E_i = L_{i-1}$
   2.2. $L_i = E_i + t_{place} + t_{proc_i} + t_{pick}$
3. $E_{N+1} = L_N$

Fig. 4: Timing of the simple periodic schedule

### *4.1. Search*

Due to the periodic nature of this schedule if we can schedule a wafer, such that it doesn't have any conflict with the previous or the next wafers, we are done. To find a conflict-free schedule, it suffices to look and search in a search window (SW) whose length is equal to $t_{total}$. This is due to the fact that the pattern in the SW is repeated over time. In SW we have to consider all portions (blocks) of the previous and next wafers that can take place in that time interval.

To resolve any conflicts in SW, transport moves involved in the conflict must be rescheduled, therefore at least one of the transport actions in the conflict must be moved. Because the relation between the blocks is important not their absolute time, we only move blocks to the left (i.e. to an earlier time). There are two ways of rescheduling a transport action associated with a block:

1. *Slide*: Move a whole block some unit of time to the left. In Figure 2 if we slide both $B_{1,0}$ and $B_{1,1}$ one unit, some conflicts are resolved but some new ones are created.
2. *Stretch*: Stretch a block to the left, so the size of the block becomes bigger. The meaning of stretching a block is to force a wafer to remain in a PM longer than $t_{proc_i}$.

By performing any of these two actions for one unit of time, the time needed to produce a wafer, $t_{total}$, is increased by one, the increased time is equal to the time that we use TM's (in *slide*) or PM's (in *stretch*) as buffers. Lemma 1 and Lemma 2 show a bound ($t_{add}$) on sliding of the first block.

By searching in the SW and *sliding* the $B_{1,0}$ at most $t_{add}$, we look at the entire possible configuration. If none of the resulting states is conflict-free, there is no answer with that period. So we increase the fundamental period by one, and start to search again with this new period.

**Lemma 1:** In a case where all $t_{res_i}$'s are equal to zero, if a feasible solution exists with period FP, it will be found with $t_{add} \le 2 * FP - (t_{pick} + t_{place}) * N$.

**Proof:** In the first simple periodic schedule (step 2.1 in Figure 3) a wafer doesn't have any idle time (it is being processed or moved all the time). $t_{add}$ is the time that each wafer would be idled. In this idle time it must be buffered either by the PM's or the TM's. Because $t_{res_i}$'s are equal to zero, PM's can not be used as buffer. Thus the wafer has spent all its idle time, $t_{add}$, on the TM's and used arms as buffers. For each wafer we have used robot arms for an interval of $2 * FP - (t_{pick} + t_{place}) * N$ as buffer and $(t_{pick} + t_{place}) * N$ for *pick* and *place* from and into the process modules. So each wafer has used arms for 2 * FP time units. If there are N wafers in the system, the total time that the system is operating can be calculated as follows.

System time = Initial state + steady state + terminate state =
Initial state + N * FP + terminate state ≈ N * FP

The time that we have used the TM's is equal to 2 * FP * N. Because we only have two arms, they have been busy all the time. Thus if $t_{add}$ exceeds that bound, the arms will be over utilized, and no solution can be found. ■

**Lemma 2:** In a case where $t_{res_i}$'s are not equal to zero the following provides a bound for $t_{add}$: $t_{add} \le 2 * FP - (t_{pick} + t_{place}) * N + \sum_i Min(t_{res_i}, (FP - (t_{proc_i} + t_{pick} + t_{place})))$.

**Proof:** The summation term over all the PM's is the time that we can use the PM's as buffers. The first part, as in Lemma 1, is the time that we can use the TM's as buffers. ■

1. Current_state ← get-periodic-schedule(FP)
2. $t_{add}$ = $t_{total}$ of the target configuration - $t_{total}$ of the Current_state
3. Current_state ← Slide $B_{1,0}$ of the Current_state $t_{add}$ units.
4. For i = 1 to N Do
   4. stretchlen = ($L_i - E_i$) in the target– ($L_i - E_i$) in current_state
   4.2. Current_state ← stretch $B_{1,i}$ of the current_state stretchlen units.
   4.3. slidelen = $E_i$ in target - $E_i$ in current state
   4.4. Current_state ← Slide $B_{1,i}$ of the current_state slidelen units.
5. End For

Fig. 5: The series of the actions that should be applied on the simple periodic schedule to reach a target configuration

#### 4.1.1. Search Space

To search for a conflict-free schedule with the minimum FP, we must examine all the possible configurations in SW. We have defined two actions, *slide* and *stretch*. Lemma 3 shows a way that we can reach any possible state (configuration of blocks) from the first simple schedule by applying *slide* and *stretch* in a specific order.

**Lemma 3:** Any schedule with period equal to FP can be reached by applying *slide* and *stretch* actions starting from the simple periodic schedule.

**Proof:** Suppose there is a target configuration and we want to find the series of the actions that we need to perform on the simple periodic schedule to reach that target configuration. The procedure in Figure 5 shows the series of the actions that we have to perform on the simple periodic schedule to reach the target state. ■

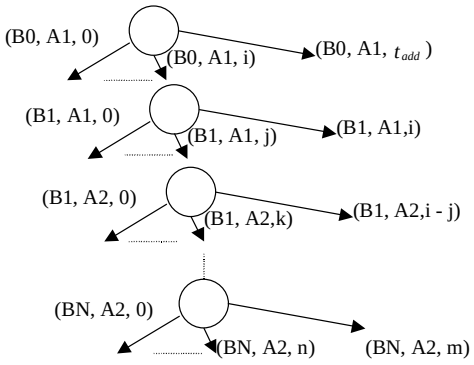


Fig. 6: Search tree

Figure 6 shows the search tree. A node in the search tree represents the state of the system, which consists of the start and end time of the blocks of the first wafer ($E_i$'s and $L_i$'s). The edges represent state transitions. By applying a transition to the current state we reach a new state in the tree. There are three elements in a state transition: Block number, which shows the block that we have to change to reach the new state, action number (A1 means *slide*, and A2 means *stretch*), and a number that shows the number of time units the action should be applied. For example, if the triplet is (B2, A1, 5), it means that we have to slide the second block 5 units.

The nodes of this tree are the configurations of blocks that we have to check to see if any of them is a conflict-free one. From Lemma 3 we can conclude that all of the possible configurations can be found in this tree.

Figure 7 shows the search pseudo code. There is a general list (state_list) that keeps all the nodes that haven't been expanded yet. At each time we get a new state from that list, if that one is conflict-free, we are done. Otherwise expand function (line 2.3.1) finds all the possible child nodes of this state, and they are added to the state_list. If all the nodes are expanded and state_list becomes empty, it means that no conflict_free schedules exist with the current FP. In this case, FP should be increased, and the same search should be repeated for the new FP.

```
Search (input simple-periodic-schedule) :
1. State_list ← simple-periodic-schedule
2. While not empty(state_list)
    2.1.current_state ← get_newstate(state_list)
    2.2.IF conflict_free(current_state)
        2.2.1.  Return (current_state)
    2.3.Else
        2.3.1.  State_list.add(expand(current_state))
3. End While
4. Return (Current_state)
```

Fig. 7: pseudo code of the search algorithm

### *4.2. Heuristics*

Because in the worst case, we have to check every node of the tree to see if it is a conflict-free state or not, the complexity of the algorithm is at the same degree as the number of nodes which is exponential in the number of visits. In this section some of the heuristics that reduce the complexity of the algorithm are presented. Using these heuristics does not affect the correctness and completeness of the solution.

**Heuristic 1:**

As mentioned before the best possible period is equal to $t_{BN}$ + $t_{pick}$ + $t_{place}$. But if the system is in transport-bound region, the best possible period is: ($t_{pick}$ + $t_{place}$)* (N + 1). So heuristic 1 calculates these two periods and starts from the greater one of the two. Therefore, we save the time that we would search for the smaller period without finding any solution.

**Heuristic 2:**

In our algorithm after the two actions of $B_{1,i}$ are examined, if that block still has any conflict with $B_{k,j}$, $B_{k,j}$ must move. So if two blocks have had their actions examined and they still have conflicts with each other, by moving other blocks those conflicts would not be resolved. Thus, the internal node of the tree representing this situation can not lead to any conflict-free node, so we can prune the sub-tree from that node, and there is no need for expanding it further.

**Heuristic 3:**

If we have two parallel modules whose blocks are greater than fundamental period, they may have conflict with each other. The only way of resolving this conflict is by stretching that block. Therefore, if blocks of a parallel-module visit that have conflict with each other, and can not stretch, we can not find any solution down that path and we can prune from that internal node.

## 5. Experimental Evaluation

### *5.1. Experiment Design*

We run the experiment on all topologies (i.e. all possible number of visits and number of PM's per visit) that can be made with 1 to 6 PM's. To assign $t_{proc_i}$ and $t_{res_i}$ to each visit, we assume that $t_{proc_i} = M_i$ * k, where k is a random number generated according to a normal distribution N (μ, σ). In this distribution, μ is a number between 1 and 30, and σ is equal to μ. For each topology, we generate three hundred different cases (ten for each μ). After assigning $t_{proc_i}$'s, $t_{BN}$ is determined. So we assign for each $V_i$, a value to $t_{res_i}$ that is a uniform random number between 0 and ($t_{BN}$ - $t_{proc_i}$) * $N_i$ (Number of parallel modules in that visit). In our experiments we assumed $t_{pick}$ = $t_{place}$ = 1. By these arrangements, 23099 cases have been generated and run on a Sun sparc station (Utra10).

To evaluate the system performance, we define a term to which we refer as the cluster factor. The cluster factor, calculated as $\frac{t_{BN}+2}{2N+2}$, provides a notion of processing over transport. If it becomes less than one, the cluster tool is in transport-bound

region. Otherwise, it is in process-bound region. One of the metrics of measuring cluster tool performance is FP. The other factor is the time, in seconds, that it takes the algorithm to find a solution.

| Scheduling Technique | Worst case running time |
|---|---|
| Without Heuristics | Over 7 days |
| With Heuristic 1 | 80,000(s) |
| With Heuristics 1 and 2 | 250 (s) |
| With Heuristics 1, 2 and 3 | 19(s) |

Table 1: Comparison between the worst case running time of the algorithms

### *5.2. Experiment Result*

We first ran the experiment on the algorithm that didn't use any e heuristics, and it took a long time, about 7 days, to find the solution for some cases. However, when we ran the same test cases on the algorithm that used all of the heuristics, in 99.6 percent of the cases the optimal solution was found in less than a second, and in the worst case it took about 19 seconds. Table 1 shows the impact of the heuristics on the program running time.
Figure 8 shows FP over a range of cluster factors. The solid lines show Ideal FP drawn according to the analytical models proposed in [4] and [6]. The other points refer to the cases where the Ideal FP must be increased, due to the residency constraints, in order to find a feasible solution. In cases with Ideal FP, when cluster tool goes into transport-bound region (cluster factor less than one), FP remains constant no matter what the $t_{BN}$ is. Most of the cases that require the Ideal FP to be increased for a feasible solution to be obtained are in the transport-bound region. As is shown in the figure, Ideal FP is increased by at most two units in transport-bound cases, and when the cluster tool is in process-bound region, the Ideal FP is increased by at most one unit. For cluster-factor values greater than seven, a feasible solution with Ideal FP is found for all cases.

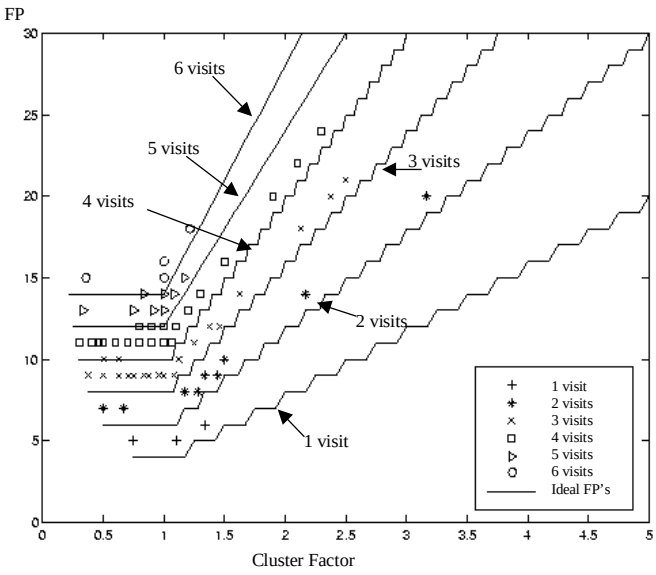


Fig. 8 : Performance, in terms of fundamental period, with respect to cluster factor

## 6. Conclusions

A scheduling technique, for cluster tools, that addresses post-processing residency constraints and throughput requirements was proposed. The technique searches in the time and resource domains for a feasible schedule with a maximum throughput. It operates in two main phases; the first one of which is of lower complexity than the other. This is to increase the complexity of the algorithm progressively, and only if necessary. In our experiments, we found that a large number of problem instances can be efficiently, feasibly and optimally scheduled in the first phase. Several heuristics are used to reduce the complexity of the scheduling algorithm further. The results of our experiments show that the scheduling technique finds feasible schedules with a maximum throughput and solves an overwhelming majority (99.6%) of the problem instances in under one second.

**References:**


[1] M. E. Bader, R. P. Hall, and G. Strasser, "Integrated processing equipment," Solid State Technol., vol. 33, pp.149-154, May 1990.

[2] P. Burggraaf, "Coping with the high cost of wafer fabs," Semicond. Int., vol. 38, pp.45-50, March 1995.

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

[4] T. L. Perkinson, P. K.McLarty, R. S. Gyurcsik, and R. K. Cavin III, "Single-wafer cluster tool performance: an analysis of throughput," IEEE Trans. Semiconductor Manufacturing, vol. 7, pp. 369-373, Aug. 1994.

[5] S.C. Wood, "Simple performance models for integrated processing tools," IEEE Trans. Semiconductor Manufacturing, vol. 9, pp. 320-328, Aug. 1996.

[6] T. L. Perkinson, R. S. Gyurcsik, and P K, McLarty, "Single-wafer cluster too performance: an analysis of the effects of redundant chambers and revisitation sequences on throughput," IEEE Trans. Semiconductor Manufacturing, vol. 9, pp. 384-400, Aug. 1996.

[7] L.W. Phillips ,and P.S. UNGER, "Mathematical Programming solution of a Hoist Scheduling Program", AIIE Trans., vol. 8, pp. 219-225, June 1976.

[8] G.W. Shapiro, and H.L.W. Nuttle, "Hoist scheduling for a PCB Electroplating Facility", IIE Trans., vol. 20, pp. 157-167, June 1988.

[9] H. Chen, C. Chu, and J.M. Proth, "Cyclic Scheduling of a Hoist with Time window Constraints", IEEE Trans. Robotics and Automation, vol. 14, pp.144-152, February 1998.

[10] L. Lei, and T.J. Wang, "The Minimum Common-cycle Algorithm for Cyclic Scheduling of Two Material Handling Hoist with Time Window Constraints", Management Science, Vol. 37, pp.1629-1639, December 1991.