SCHEDULING TASKS IN ROBOT LATTICE PRODUCTION LINES WITH STORAGE

D. W. Carter, Jr., G. Syrmos, R. W. Newcomb

Microsystems Laboratory
Electrical Engineering Department
University of Maryland
College Park, Maryland 20742

I. INTRODUCTION

In this paper a model lattice manufacturing system is presented, then task scheduling is outlined for an arbitrary lattice production line. The algorithm presented takes advantage of the interconnectivity inherent in the lattice structure. The result is a new automated production system that provides more flexibility, utilization, and speed than other systems in use today.

Three problems often found in manufacturing are maximizing throughput-efficiency, failure recovery, and in more advanced systems, concurrency ie., deadlock prevention and recovery. In attacking these problems a proper choice of models for the mechanical and communication systems is essential.

There are many ways to represent lattice structures. Petri-nets are often used because they ease many complex calculations [2]. However, our model is more closely represented by a network lattice with nodes that contain robot(s) at the vertices. The lattice model proposed in this paper has been proven to be the most efficient with regards to its interconnectivity capabilities and reliability [1]. In particular, lattice structures provide maximum interconnectivity which, in turn, yields maximum flexibility for routing and rerouting of tasks in the presence of station failures.

The nodes of our model are always stationary. The order of processing is independently of the placement of the nodes. This is advantageous because the processing order can change without having to relocate nodes. Furthermore, all of the major parts of the system operate independent of each other. The result is massive parallel processing. Using this architecture, all of the nodes can operate simultaneously while objects are moving between the nodes.

Our algorithm is based upon the central control computer which determines the optimum routing or rerouting for the flow of parts in the lattice under the current conditions of the environment. This computer is connected directly with every node. Each node is mechanically connected to its adjacent nodes in the lattice configuration.

II. SCHEDULING TASKS IN LATTICE PRODUCTION LINES

Let us analyze an example lattice production line. We will use a simple example that has a link failure between nodes 2 and 4. This will be helpful in understanding the basic algorithm.

Consider the model of Figure 1. Suppose that all links between the nodes can transport a single item in both directions. Let the transport time between any two adjacent nodes be the same. Adjacent nodes are any two nodes that are separated by one link.

We will work through the example by first deciding what to make. Next, we will choose machines or nodes and define the tasks that can be accomplished by each node. Then we will present a flowchart, Figure 2, of the general steps of the algorithm. Lastly, we will step through the algorithm performing each step on the example until all of the items are scheduled.

Suppose we want to make desks, chairs, and bookshelves. From Figure 1, we know there are four nodes R_1, R_2, R_3, R_4, (l= 4). The next step is to define the capabilities of each node. Let us denote the tasks which can be performed by each node as follows:

- T = $\{T_1, T_2 \ldots T_m\}$ is the set of all tasks that can be performed by any machines;
- T_x = is a task that can be performed by a node :$\forall$ x,y where $1 \le x,y \le m$, $T_x = T_y \Rightarrow x = y$
- t_x = is called a task relative time weight; t_x is the time is takes to perform task T_x;
- P^c = the set of tasks that can be performed at node R_c;
- P = the set of all possible tasks that can be done by the system, ie. $P = \{ P^1, P^2, \ldots P^{l=4} \}$; So, let

$P^1 = \{ T_1, T_2 \}$, $P^2 = \{ T_3 \}$, $P^3 = \{ T_2 \}$, $P^4 = \{ T_1 \}$

Let us assign task relative time weights to each task above, relative to the longest task. The weighted times indicate what percentage of a time interval it takes a node to accomplish a specific task. For P^1 let $P^1 = \{ t_1 = .3, t_2 = .7 \}$. As mentioned earlier, processing does not take place across time interval boundaries. Therefore, the time for all of the tasks that occur at any node in a sequential manner without rest must be less than or equal to one time interval. The converse is also true, ie. the time interval must be greater than or equal to the time it takes to perform the longest task at any node. Within a given time interval, P^1 can:

a) perform t_1 1, 2 or 3 times because $.3 \times 3 \le 1$; or
b) perform both t_1 and t_2 because $.3 + .7 = 1$; or
c) perform t_2 once because $.7 < 1$

For our example let, $P^2 = \{t_3 = .2\}$, $P^3 = \{t_2 = .5\}$, $P^4 = \{t_1 = 1\}$. Now that we have decided what tasks can be accomplished at each node and how much relative time each individual task takes, we can proceed with the algorithm.

We will use periodic synchronous state switching implemented by the flow chart of Figure 2. The 1st block in this figure is to compute production ratios for maximum profit. Then the user would enter the computed ratios into the system. For example, suppose we obtain maximum profit by producing two chairs, one desk and a bookshelf. Let these items be represented as follows:

I = $\{I_1, I_2, \ldots I_n\}$ is the set of all items to be made.

S^a = a set of finite sequences of tasks that will produce I_a;

S^{aj} = a particular production sequence of tasks that will produce I_a; J=$\{1,2, \ldots q\}$, J is the number of ways to produce I_a, j is a specific method of production;

Let I_1 = chairs, I_2 = desks, I_3 = bookshelves, and let their respective sequences be S^1 for chairs, S^2 for desks, and S^3 for bookshelves; so, the production ratio is $S^1(2) : S^2(1) : S^3(1)$. This means that S^1 will appear twice in the production chain, while S^2 and S^3 will only appear once. A production chain denoted E, is a time line of k finite intervals s.t. all elements of all sequences are mapped between the intervals.

The 2nd box is used to designate the use of input/output slots. The lattice model is particularly flexible in this area. Parts can be introduced at any external slot and finished products can be removed from any external slot. In our example we will designate all inputs to occur at R_1 and all outputs to occur at R_3. Let us suppose that the i/o occurs instantaneously for ease of calculation.

In box 3 we assess the resources needed for each time interval and construct a resource allocation table. All possible tasks of the system, P, are available for allocation during each interval.

This step also takes into account the storage capabilities of the station slots at each node. For example, suppose we have a saw at a lumber mill that is capable of cutting a block of wood in half every five seconds. Suppose the next step for the item that is currently being worked on is sanding, which takes three minutes. In an ordinary system, the saw and the block that was cut in half must wait until the sander is finished. If this wait occurs at the saw then the throughput of the saw is greatly reduced.

A common solution is to add more saws, raising the cost of manufacturing. Perhaps this alternative can be avoided. In a lattice system with station slots, the two half-blocks can be sent to the sander to wait at its station slots or they can wait at the station slots of the saw. This frees the saw to process more items.

The 4th box involves ordering the items to make an ordered list. For now let our list be: I_1, I_1, I_2, I_3. There are instances where it is necessary to prioritize items in the list. The primary reason for prioritization is when items need to be produced within time frames that are smaller than the scope of the production chain. The order in which the items are processed affects the length of the production chain. Therefore, all permissible permutations of this list should be computed to produce the optimal production chain. For our example, we shall not prioritize any items.

In box 5 we are to select the shortest production sequence for each item. First, we must define the production sequences of each item. Let each item have the following production sequences:

S^{11} = $\{T_1, T_2, T_3\}$, S^{12} = $\{T_2, T_1, T_3\}$,

S^{21} = $\{T_1, T_3, T_1\}$,

S^{31} = $\{T_1, T_1, T_3\}$, Note: I_1 has 2 shortest sequences.

In the next box, 6a, each item is scheduled as far as possible within one time interval. This is simulated by marking the resources as they are used. In box 6b, after all of the items have been processed, the time interval is incremented and a new (unmarked) resource allocation table is used until the requested items are produced. In other words, we are computing a production chain, E.

Before we can begin scheduling items we must compute priority chains based on the distance between nodes. These chains nodes further away from the current node. Thus, distance penalties are taken into account when allocating node resources. For priority 0 nodes there is no penalty. For nodes with higher priority;

penalty = priority # * (transfer time between two adjacent nodes = .1) and, total task cost = task cost + penalty

Naturally, we always choose the node that offers the lowest total task cost. Returning to our example we have the following: (Commas are used to separate nodes of the same priority; semi-colons are used to distinguish priority levels)

R_1= R_2, R_3, R_4 - all nodes priority 0

R_2= $R_1, R_3; R_4$ - R_1 and R_3 have priority 0, and R_4 has priority 1

R_3= R_4, R_2, R_1 - all nodes have priority 0

R_4= R_3, R_1, R_2 - R_1 and R_3 have priority 0, and R_2 has priority 1

So, generally speaking, an object at node R_4 would try to find the next process for it at R_1 or R_3 before trying R_2. Therefore, E can be computed using the methodology of Table 1.

Table 1, Time interval #1

item sequence		tasks performed during interval	nodes where tasks are active	link utilization intervals
S^{11}	:	T_1, T_2	R_1, R_3;	$link_{1,3}(.4)$, $link_{3,2}(1)$
S^{11}	:	T_1 R_1;	$link_{1,3}(.7)$	
S^{21}	:	T_1 R_1;	$link_{1,2}(.1)$	
S^{31}	:	T_1 R_4;	no links used	

Between time interval #1 and time interval #2 $link_{4,1}$ is used to transfer the object at R_4 to R_1. The other calculations omitted because of space limitations. The resulting E is shown below.

Table 2, Resulting E

		E_0	E_1	E_2 . . .	E_k
S^{11}	:	T_1, T_2	T_3		
S^{11}	:	T_1	T_2, T_3		
S^{21}	:	T_1	T_3, T_1		
S^{31}	:	T_1	T_1, T_3		

The lattice production line used in our example is able to complete a single production chain in only two time intervals. The use of concurrency in this algorithm allows 12 tasks to be completed using four robots in two time intervals where the time interval is equal to the time it takes to complete the longest task.

This scheduling technique prevents deadlock in production chains by elimination of the hold and wait condition that is necessary for a deadlock to occur [5]. This wraps up the example. So, we will finish explaining the algorithm.

If there is a production chain with multiple shortest sequences, such as S^1, that has not been checked, then do boxes 7a and 7b. We will store the production chain and its total time for later comparison. Then we choose a S^{1j} that has not been used for this ordered list and substitute it for one of the $S^{1x's}$ currently in the list. If we substitute S^{12} for the first S^{11} then we have the following

ordered list: { S^{12}, S^{11}, S^{21}, S^{31} }. Then we will do boxes 6a and 6b. Afterwards we will check again to see if there is another substitution that can be made. After we have computed all of the possibilities for this ordered list of items, we will do box 8a, ie. store the shortest production chain.

Next, we will check to see if we have all order possibilities. If not, we will do box 9, ie. reorder the list of items and compute another shortest production chain. This process will be repeated until all possible production chains have been computed. At which time box 8b will be executed storing the optimal production chain.

In box 10 the system initialization is computed. At time interval 1, parts are required at two nodes P^1 and P^4. Earlier we established that all input will occur at R_1. So, because R_4 is further away we must input the part that is need at that node first.

Hopefully, we have presented enough to give a basic understanding of how the scheduling algorithm works.

III. CONCLUSIONS

In concluding, it should be mentioned that interconnectivity and storage dramatically reduce the time lost between different tasks. Each node can perform different tasks on different items in a manner similar to multitasking on a computer because of the storage capabilities. The advantage of high reliability adds significantly to the flexibility of robot lattice production lines and makes them prime candidates for use in today's automated manufacturing systems.

REFERENCES

[1]. Z.N. Cai, A. Farnham, A.Z. Ghalwash, P. Gomez, V. Rodellar, and R.W. Newcomb, "Petri-Nets for Robot Lattices," Proceeding of the 1987 IEEE International Conference on Robotics and Automation, March-April 1987, vol. 2, pp. 999-1004.

[2]. A. Kusiak and G. Finke, "Selection of Process Plans in Automated Manufacturing Systems," IEEE Journal of Robotics and Automation, vol. 4, no. 4, August 1988, pp. 397-402.

[3]. R. Papannareddy, C.A. Niznik, H. Alayan, and R.W. Newcomb, "Processor Requirements for Reliable Automated Manufacturing Robot Networks," Proceedings of the Mini and Microcomputers and their Applications Conference, San Antonio, TX, 1983, pp. 73-77.

[4]. M.A. Peshkin and A.C. Sanderson, "Planning Robotic Manipulation Strategies for Workpieces that Slide," IEEE Journal of Robotics and Automation, vol. 4, no. 5, October 1988, pp. 524-531.

[5]. J. L. Peterson and A. Silberschatz, Operating System Concepts, Massachusetts, Addison-Wesley, 1895

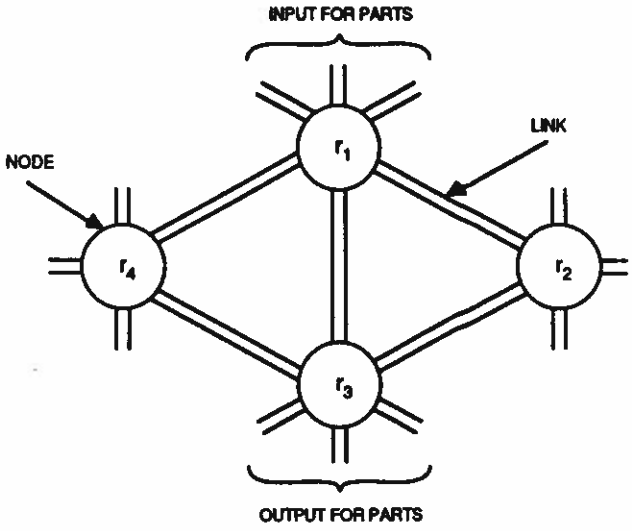

FIGURE 1. Example Lattice System

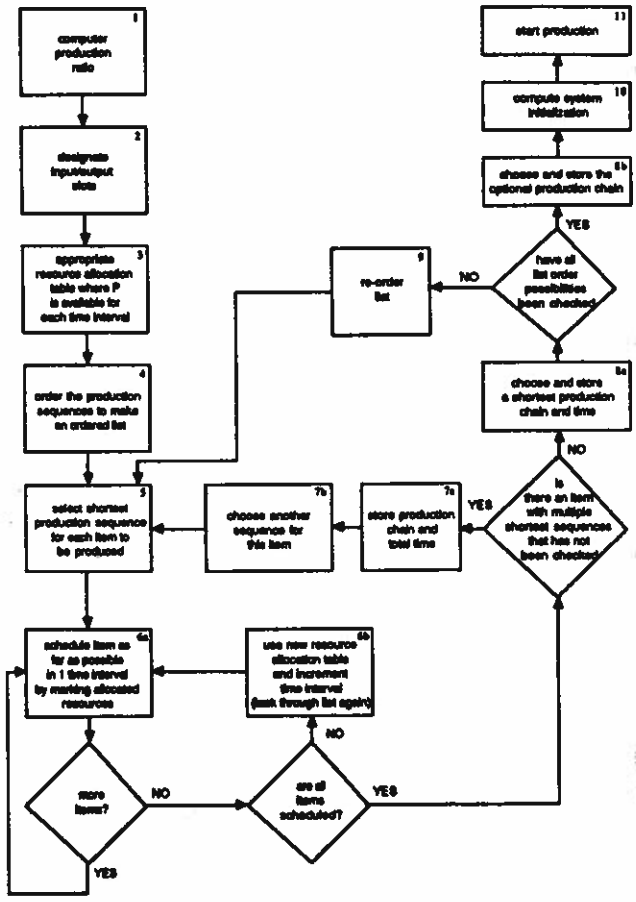

FIGURE 2. Flow Chart For Pre-Production Scheduling Algorithm