# IMPROVING THE THROUGHPUT OF A PIPELINE BY INSERTION OF DELAYS †

Janak H. Patel and Edward S. Davidson
Coordinated Science Lab
University of Illinois
Urbana, Illinois 61801

## Summary

A pipeline is defined to be a collection of resources, called segments which can be kept busy simultaneously. A task once initiated, flows from segment to segment for its execution. A collision occurs if two or more tasks attempt to use the same segment at the same time.

The collision characteristics of a pipeline with respect to a schedule of task initiations are investigated. A methodology is presented for modifying the collision characteristics with the insertion of delays so as to increase the utilization of segments and hence the throughput under appropriate scheduling.

## I. Introduction

Pipelines are becoming increasingly common in many computers, sometimes for achieving high speed computation at a lower cost than would result from simply using higher speed electronic components. However, in most cases it is used because of a better performance per unit cost over other architectures. A pipeline as defined here is a collection of resources called segments which can be kept busy simultaneously. A task once initiated, flows from segment to segment for its execution, in a predetermined manner. The effectiveness of the pipeline lies in the fact that a task can be initiated before the completion of some previously initiated tasks resulting in high performance and segments can be special rather than general purpose resulting in low cost. We term a pipeline in which all the tasks have identical flow patterns, a single function pipeline. In a multifunction pipeline there are two or more distinct possible flow patterns and each task uses one of these flow patterns. Each flow pattern is identified by a function name and it can be displayed in a reservation table, such as Figure 1 and 6. Rows correspond to segments and columns to units of time. A function name, denoted by a single capital letter, is placed in row i and column j (cell (i,j)) if after j units of execution a task with that function name requires segment i. We shall consistently use X as a function name in single function pipelines. Fig.6 is a reservation table of a multifunction pipeline with two distinct flow patterns for two functions A and B.

In our model we assume that a task once initiated must flow synchronously without preemption or wait. There is no restruction on the flow patters, however. In Fig. 1, multiple X's in a row may indicate either a slow segment or segment reusage (feedback). Multiple X's in a column indicate parallel computation. It is the reusage of a segment which poses a problem, namely, two or more tasks may attempt to use the same segment at the same time, resulting in a collision. However, in multifunction pipelines even without any reusage, a collision may occur because of two or more independent and distinct flows of tasks.

In previous work, the central problem treated is to schedule the tasks in a given pipeline so as to achieve high throughput without causing any collision. This problem was first investigated in [1]. Subsequent work on this problem is reported in two doctoral theses [2,3]. An overview of some related results and a more comprehensive bibliography can be found in [4]. Our investigation is from a different perspective and seeks a methodology for modifying the reservation table of a given pipeline so as to increase the utilization of segments and hence the throughput under appropriate scheduling.

The pipeline utilization is limited by its collision characteristics which are a result of the usage patterns of the segments. One way of modifying usage pattern is by segment replication. Another way is to remove our assumption regarding the waiting of a task between two steps and provide internal storage buffers which allow variable delay between segments [4]. Still another way of changing a usage pattern is by inserting noncompute segments, which simply provide a fixed delay between some computation steps. It is the modification of a pipeline by the use of noncompute segments which is the concern of this paper. It is assumed that any computation step can be delayed by inserting usage of a noncompute segment, where each X in the reservation table is considered to be a computation step.

We shall first consider single function pipelines for ease of understanding, since the notational complexity of multifunction pipelines is considerable.

## II. Single Function Pipelines

We start by investigating some collision characteristics of a single function pipeline (referred to simply as pipelines in this and the following section). A usage interval of a segment is defined to be a time interval between two reservations (X's) of that segment by a single task. For example in Fig. 1, all usage intervals of $S_0$ are 2, 3 and 5. Let F be the set of all usage intervals of a pipeline: e.g., F={1,2,3,5} for Fig. 1. Clearly any two tasks will cause a collision if and only if they have the same initiation time interval as one of the usage intervals.

A sequence of task initiations can be completely described by a sequence of initiation intervals between successive tasks (also known as latency). For example, task initiations at time instants 0, 3, 5, 9 and 12 can be described by the latency sequence ⟨3,2,4,3⟩. An initiation interval of 0 is not permissible. Let G be the set of all initiation intervals (not just the intervals between successive initiations) of a latency sequence. Thus G for the latency sequence ⟨3,2,4,3⟩ is {2,3,4,5,6,7,9,12}.

If a subsequence of latencies appear periodically in an infinite sequence, it is termed an initiation cycle. Thus a cycle (2,3,2,5) implies an infinite initiation sequence ⟨2,3,2,5,2,3,2,5,2,3,2,5,2,...⟩. A constant latency cycle is a cycle with only one latency latency; e.g., cycle (4). Let the period, p, of a cycle be defined as the sum of the latencies in the cycle. Thus the period p of cycle (2,3,2,5) is 12 and p of cycle (4) is 4. The average latency, $l_a$ of a cycle is the average of the latencies of the cycle. For example, $l_a$ for cycle (2,3,2,5) is 12/4=3. This implies an average initiation rate of one task every 3 time units.

The initiation interval set G of a cycle is simply the set G of the infinite initiation sequence implied by the cycle. Thus G={4,8,12,16,20...} for cycle (4) and for cycle (2,3,2,5) G is {2,3,5,7,9,10,12,14,15, 17,19,21,22.24,26,...}. Let G mod p be the set formed

† This research was supported in part by the National Science Foundation under Grants GJ-35584X and GJ-40584 and in part by the Joint Services Electronics Program (U.S.Army,U.S.Navy, and U.S. Air Force) under Contract DAAB-07-72-0259.

by taking modulo p equivalents between 0 and (p-1) of the elements of $\underline{G}$. For cycle (2,3,2,5) with p=12, $\underline{G}$ mod 12={0,2,3,5,7,9,10} and for constant cycle (4) with p=4 $\underline{G}$ mod 4={0}. It can be shown that $\underline{G}$ and $\underline{G}$ mod p of a cycle have the following simple properties, remembering that 0 is not a permissible initiation interval.

P1.a. if $g \neq 0$ then $g \in \underline{G}$ mod p $\Rightarrow$ $g+ip \in \underline{G}$ $\forall i \geq 0$

b. $0 \in \underline{G}$ mod p and $ip \in \underline{G}$ $\forall i \geq 1$ always.

P2. if $g \neq 0$ then $g \in \underline{G}$ mod p $\Leftrightarrow$ $(p-g) \in \underline{G}$ mod p.

It is useful to introduce the set $\underline{H}$, the complement set of $\underline{G}$ in $\underline{Z}_+$, the set of positive integers. Clearly $\underline{H}$ mod p = $\underline{Z}_p - \underline{G}$ mod p. Where $\underline{Z}_p$ is the set of integers modulo p. Then the following is a direct consequence of P2.

P3. $h \in \underline{H}$ mod p $\Leftrightarrow$ $(p-h) \in \underline{H}$ mod p.

An initiation interval between two tasks is said to be allowable with respect to a pipeline if these tasks do not collide in the pipeline. A cycle is allowable with respect to a pipeline if all its initiation intervals are allowable. Conversely, we also say that a usage interval or a pipeline is allowable with respect to a cycle if no collision occurs. A collision occurs in a pipeline when a cycle is followed iff (if and only if) some initiation interval of the cycle equals a usage interval of the pipeline. Thus a cycle is allowed by a pipeline iff there are no elements common between the usage interval set, $\underline{F}$, of the pipeline and the initiation interval set, $\underline{G}$, of the cycle; i.e., iff $\underline{F} \cap \underline{G} = \phi$, or equivalently, iff $\underline{F} \subseteq \underline{H}$. Thus $\underline{H}$, the complement set of $\underline{G}$ can be described as the set of allowable usage intervals with respect to the given cycle. By using the property P1 of $\underline{G}$, the allowability condition can be reduced to the following theorem.

Theorem 1: A cycle with period p and initiation interval set $\underline{G}$ is allowed by a pipeline with usage interval set $\underline{F}$, iff ($\underline{F}$ mod p) $\cap$ ($\underline{G}$ mod p) = $\phi$. □

A constant latency cycle ($\ell$) has p = $\ell$. Its $\underline{G}$ mod p is always {0} and hence the following.

Corollary 1.1: A constant cycle ($\ell$) is allowed by a pipeline iff no usage interval is an integral multiple of $\ell$.

It is helpful to look at the allowable usage interval set $\underline{H}$ to see what allowable pipelines can be constructed for a given cycle. Let a row which has an X in each of columns $t_1, t_2, \ldots t_k$ be denoted as row $\{t_1, t_2, \ldots t_k\}$; e.g., the 2nd row of Fig. 1 is row {1, 2,4}. A pipeline is allowed by a cycle if all its rows are allowed. To construct an allowable row we can start by placing an X in some column i. We can place another X in some column j, only if the usage interval $|i-j| \in \underline{H}$; a third X in some column k if $|i-k|$ and $|j-k| \in \underline{H}$, and so on.

However, it is convenient to restrict the column numbers to be between 0 and (p-1), and still retain all the useful information. For this, let us define two elements i, j $\in \underline{Z}_p$, to be compatible if $|i-j| \in \underline{H}$ mod p. The use of the absolute quantity can be avoided by using property P3 of $\underline{H}$ mod p. Thus we have the following lemma.

Lemma 2.1: Two integers $i, j \in \underline{Z}_p$ are compatible iff $(i-j)$ mod p $\in \underline{H}$ mod p. □

Using the definition of compatibility or the above lemma we can form all the compatibility classes on the elements of $\underline{Z}_p$, given a cycle. A compatibility class is one in which each element is compatible with every other element in the class. We need to form only the maximal compatibility classes. A maximal compatibility class is not a subset of any other compatibility class.

If $\{z_1, z_2, \ldots z_k\}$ is a compatibility class with respect to some cycle then the row $\{z_1, z_2, \ldots z_k\}$ is allowed by that cycle. This is because by the definition of compatibility all usage intervals $|z_i - z_j|$ are allowable. In this way we can produce only a limited number of allowable rows. However, with the use of property P3 and Lemma 2.1 it is possible to construct other allowable rows as follows.

Theorem 2: Given a cycle with period p, the following rows, and only those rows, are allowed by the cycle:

row $\{z_1+i_1p, z_2+i_2p, \ldots\}$ $\forall$ integers $i_1, i_2, \ldots$ and $\forall$ compatibility classes $\{z_1, z_2, \ldots\}$ of the cycle. □

Consider a problem in which a pipeline, characterized by its usage interval set, is given and one has complete freedom in choosing an allowable initiation sequence. Bounds on the minimum average latency of such sequences and a branch-and-bound algorithm to discover a minimum average latency allowable cycle are reported in [1] and [4]. Minimum average latency cycles maximize segment utilization, where utilization is measured as the percent of time the segment remains busy.

Here we consider the reverse problem. Namely, a cycle is given and one has complete freedom in choosing any allowable usage pattern. While the solution to the former problem is useful for scheduling a given pipeline, the solution to this problem is useful for designing a pipeline for a given schedule. Theorem 2 completely characterizes the entire class of allowable pipelines. We shall soon see that it is possible to put an upper bound on segment utilization with the help of the compatibility classes. To achieve maximum utilization of a segment for a given cycle, we must increase the number of usages per task; i.e., increase the number of X's in a row. Theorem 2 gives all possible allowable rows and it implies that the maximum number of X's in any allowable row is equal to the size of the largest compatibility class. Thus the maximum achievable utilization of a segment with respect to a given cycle is the ratio of the size of the largest compatibility class to the average latency of the cycle.

Example 1: For cycle (1,9), p=10, average latency $\ell_a$=5, $\underline{G}$ mod 10 = {0,1,9} and hence $\underline{H}$ mod 10 = {2,3,4, 5,6,7,8}. The maximal compatibility classes containing 0 are {0,2,4,6,8}, {0,2,4,7}, {0,2,5,7}, {0,3,5,7}, {0,2,5,8}, {0,3,5,8}, and {0,3,6,8} of which the largest has size equal to 5. Note that classes containing 0 are sufficient to characterize all classes since a constant may be added modulo p to all elements of a compatibility class to produce another compatibility class. Thus by Theorem 2, no allowable row has more than 5 X's. This implies that the maximum possible segment utilization with cycle (1,9) is 5/5=100%. □

Example 2: For cycle (2,3,7), p=12, $\ell_a$=12/3=4, $\underline{G}$ mod 12={0,2,3,5,7,9,10} and hence $\underline{H}$ mod 12={1,4,6,8, 11}. The maximal compatibility classes containing 0 are {0,1}, {0,4,8}, {0,6}, and {0,11} of which the largest has 3 elements. Thus the maximum number of X's in any allowable row is 3 which in turn implies a maximum segment utilization of 3/4=75%. In other words no allowable pipeline for cycle (2,3,7) has a segment which is busy more than 75% of the time. □

Among cycles with same $\ell_a$, those which allow a high utilization and hence more economical realization are clearly preferable. Furthermore they offer more

flexibility in pipeline design. Let us define a cycle to be perfect, if it allows a 100% segment utilization; e.g., cycle (1,9) of Example 1. Unfortunately we cannot test the perfectness of a cycle without forming the compatibility classes. However, we know a special class of perfect cycles which are of considerable interest in single function pipelines.

Theorem 3: All constant latency cycles are perfect.

Proof: For constant cycle ($\ell$), $\underline{G}$ mod p=$\{0\}$ and thus $\underline{H}$ mod p=$\{1,2...(\ell-1)\}$. One can verify that $\{0,1,2,\ldots,(\ell-1)\}$ is a compatibility class with $\ell$ elements. Hence the upperbound on the segment utilization is $\ell/\ell$ = 100%. □

## III. Noncompute Segments

In this section we consider the addition of noncompute segments to a pipeline to make it allowable for a given cycle. The effect of delaying some computation step can be displayed in a reservation table by writing a 'd' before the X which is being delayed. Each d indicates one unit of delay called an elemental delay. In the absence of any other information on precedence, we must assume that all the steps in a column must be completed before any steps in the next column are executed. Therefore, if the steps in column 2 of Fig. 1 are unevenly delayed, we must store the output of some steps so that all the outputs are simultaneously available to the steps in column 3 of Fig. 1. The effect of delaying the step in row 0, column 2 ($X_{02}$) of Fig. 1 by 2 units and $X_{22}$ by 1 unit is shown in Fig. 2. The elemental input delays $d_1$, $d_2$, and $d_3$ require the elemental output delays $d_4$, $d_5$, and $d_6$. Now given some integer i between 0 and (p-1), we are in a position to delay any step arbitrarily such that the step occurs in a column number equivalent to i modulo p. Thus given a cycle, we can make any row of a given reservation table to look like one of the rows of Theorem 2; provided of course, the row does not have more X's than the size of the largest compatibility class of the cycle. Hence we have the following theorem.

Theorem 4: For a given cycle, a pipeline can be made allowable by delaying some of the steps, iff the number of X's in each row of the reservation table is less than or equal to the size of the largest compatibility class of the cycle. □

Corollary 4.1: For a given constant latency cycle ($\ell$), a pipeline can be made allowable by delaying some steps, iff there are no more than $\ell$ X's in each row of the table. □

An important implication of Corollary 4.1 is that by adding elemental delays to a pipeline one can always fully utilize a single function pipeline with the use of a cycle with constant latency equal to the maximum number of X's occurring in any single row of the reservation table. Full utilization of a pipeline here, means that at least one segment is busy all the time. Thus the maximum achievable throughput of that pipeline is attained. Of course complete redesign or replication of selected segments to reduce the number of X's in a row may allow higher throughput.

Example 3: The reservation table of Fig. 1 is to be made allowable with respect to cycle (1,5). The resulting table appears in Fig. 3. For cycle (1,5), p=6, $\underline{G}$ mod 6=$\{0,1,5\}$ and hence $\underline{H}$ mod 6=$\{2,3,4\}$. The maximal compatibility classes containing 0 are: $\{0,2,4\}$ and $\{0,3\}$. The first row of Fig. 3 is row $\{0,2,10\}$, which resulted from the class $\{0,2,4\}$ by constructing row $\{0,2,4+p\}$ as per Theorem 2. The second row, $\{1,3,5\}$ results from class $\{0,2,4\}$ and the third row, $\{2,4\}$ results from class $\{2,4\} \subset \{0,2,4\}$. Thus all the rows are allowable. □

Once we have a modified table, we need to assign the elemental delays to noncompute segments. Noncompute segments are physical resources like any other segment and may be shared by various elemental delays for their efficient utilization. Two elemental delays $d_i$ and $d_j$ are defined to be compatible if $|t_i-t_j|$ mod p $\in \underline{H}$ mod p. Where $t_i$ and $t_j$ are labels of the columns in which $d_i$ and $d_j$ appear. Clearly, if $d_i$ and $d_j$ are compatible, they can share one noncompute segment because the usage interval $|t_i-t_j|$ is allowable. Using the above definition we can form the maximal compatibility classes of all the elemental delays present in the solution. All the elements of a compatibility class can share a single noncompute segment. Now the problem reduces to the standard covering problem; i.e., finding the minimum number of compatibility classes which cover all the elemental delays.

Example 4: The set of elemental delays of Fig. 3 is $<d_1,d_2,d_3,d_4,d_5,d_6,d_7>$. Their corresponding column numbers are <3,6,7,8,9,2,3>. For cycle (1,5), $\underline{H}$ mod 6 is $\{2,3,4\}$ (from Ex. 3). Thus $\{d_1,d_2\}$, $\{d_1,d_3\}$, $\{d_2,d_4\}$, $\{d_2,d_5\}$, $\{d_2,d_6\}$, $\{d_2,d_7\}$, $\{d_3,d_5\}$, $\{d_3,d_7\}$ are the maximal compatibility classes. Noting that the subsets of maximal compatibility classes are compatibility classes, one of many possible minimal covers is $\{d_1,d_2\}$, $\{d_4\}$, $\{d_5\}$, $\{d_6\}$, $\{d_3,d_7\}$. Thus 5 noncompute segments are required. The assignement above is shown in Fig. 4, where $S_3$ through $S_7$ are noncompute segments. □

Besides reducing the number of noncompute segments in a solution, it is also important to reduce the added execution delay. The execution delay of a task in Fig. 1 is 6 units while in the modified table of Fig. 4 it is 11 units. In situations where it often becomes necessary to empty the pipeline; e.g., due to logical dependancies among tasks, the execution delay of a task can become an important parameter in determining the overall throughput. Therefore, we shall take the added execution delay as the objective function to be minimized. Now the problem of making a pipeline allowable can be formulated as follows.

Let I and J be the number of rows and columns in the given reservation table. Let $d_{ij}$ and $d'_{ij}$ be the number of elemental delays to be inserted respectively at the input and output of a step $X_{ij}$ of the reservation table. If no X occurs in cell (i,j) of the table then $d_{ij}$ and $d'_{ij}$ are defined to be zero. Some other $d_{ij}$ can be set to zero if it occurs between two consecutive computation steps which are indivisible. Let D be the added execution delay. Then the problem can be formally stated as:

$$\text{Minimize } D = \sum_{0\le j<J} \left( \max_{0\le i<I} (d_{ij}) \right)$$

subject to the constraints,

integer $d_{ij} \ge 0$.

$$[(c-b)+d'_{ab}+d_{ac}+\sum_{b<j<c} \left( \max_{0\le i<I} (d_{ij}) \right)] \bmod p \in \underline{H} \bmod p.$$

for each pair $<X_{ab},X_{ac}>$ with c > b.

where, $\underline{H}$ is the set of allowable usage intervals with

respect to the given cycle with period p, and

$$d'_{ab} = \max_{0 \le i < I} (d_{ib}) - d_{ab}$$

The constraints result directly from Theorem 1. The term (c-b) is the usage interval which existed between $X_{ab}$ and $X_{ac}$ before the insertion of any delays. The variable $d'_{ab}$ is the number of elemental delays at the output of step $X_{ab}$; $d_{ac}$ is the number at the input of step $X_{ac}$. The summation term in each constraint is the effect of inserted delays in the intervening columns between $X_{ab}$ and $X_{ac}$.

Since all the constraints are in modulo p arithmetic, $d_{ij}$ need only take integer values between 0 and (p-1). Thus the solution space of the above problem is finite. This places an upper bound on the added execution time equal to (p-1)·J, where J is the number of columns in the reservation table. Moreover, the objective function D is nondecreasing in $d_{ij}$. These properties suggest the following branch-and-bound algorithm to find all minimum added delay solutions.

Let the number of X's in the reservation table be n and let the n variables, $d_{ij}$, be stored in any arbitrary order in a one dimensional array V. Let D(i) represent the value of the objective function for given values of V(1) through V(i), with V(i+1) through V(n) taken to be 0.

Algorithm B:

B1. [Initialize] i←0; BOUND←(p-1)·J;
B2. [Advance] i←i+1; V(i)←0;
B3. [Check bounds and constraints] if (V(i)=p) or (D(i)>BOUND) then go to B6; if a completely assigned constraint is violated then go to B5;
B4. [Solution found?] if i<n then go to B2 else output the solution V(1) through V(n) and D(n); BOUND←D(n);
B5. [Try another value] V(i)←V(i)+1; go to B3;
B6. [Backtrack] i←i-1; if i>0 then go to B5 else terminate the algorithm.

The last value of BOUND is the minimum value of the objective function over all possible solutions and therefore the output solutions meeting this bound are all the minimum added delay solutions. If only one optimum solution is desired, the condition D(i)>BOUND in step B3 should be changed to D(i)≥BOUND.

A complete example with the constraints and the backtrack tree are given in Fig. 5. The variables, $d_{ij}$, have been retained in the figure for simplicity. B is the variable BOUND, and '>B' indicates that the bound has been exceeded, and 'i' indicates that the constraint (a) has been violated.

This algorithm is remarkably efficient in our limited experience. For example for one 20 variable problem with a potential $10^{14}$ nodes only $10^4$ nodes were expanded and an optimum solution was obtained in 40 seconds on an IBM 360/67. For a particular class of problems, the technique of [5] may be applicable to estimate the complexity of the algorithm.

## IV. Multifunction Pipelines

Here we present the generalizations of most of the results and definitions of the previous two sections. The variables X and Y will be used in most of these results, where X and Y take function names as their values; the values need not be distinct. Let $F_{XY}$ be the set of usage intervals of all <X,Y> pairs in the reservation table. This set can be formed by taking all pairwise distances between an X and a Y which appears to the right of the X in the same row. For example, for the reservation table of Fig. 6, the sets of usage intervals are: $F_{AA}=\{1\}$, $F_{AB}=\{0,1,2,4\}$, $F_{BA}=\{0,2,3\}$, $F_{BB}=\{2,3\}$.

Similarly we define $G_{XY}$, the set of initiation intervals of all <X,Y> pairs of a cycle, to be the set which contains all intervals of a task of type X from a previously initiated task of type Y. A cycle is described with latencies suffixed with the function name of the task being initiated with that latency; e.g., cycle $(1_A,1_B,2_A)$. The period p is the sum of the latencies. The initiation interval sets for cycle $(1_A,1_B,2_A)$ are: $G_{AA}$ mod $4=\{0,1,3\}$; $G_{AB}$ mod $4=\{2,3\}$; $G_{BA}$ mod $4=\{1,2\}$; $G_{BB}$ mod $4=\{0\}$. The properties P1, P2 and P3 can be generalized as follows.

P4.a. if $g \neq 0$ then $g \in G_{XX}$ mod $p \Rightarrow g+ip \in G_{XX}$ $\forall i \ge 0$.
b. $0 \in G_{XX}$ mod p and $ip \in G_{XX}$ $\forall i \ge 1$, always.
c. if $X \neq Y$ then $g \in G_{XY}$ mod $p \Rightarrow g+ip \in G_{XY}$ $\forall i \ge 0$.

P5.a. if $g \neq 0$ then $g \in G_{XY}$ mod $p \iff (p-g) \in G_{YX}$ mod p.
b. $0 \in G_{XY}$ mod $p \iff 0 \in G_{YX}$ mod p.

P6. if $h \neq 0$ then $h \in H_{XY}$ mod $p \iff (p-h) \in H_{YX}$ mod p, where $H_{XY}$ mod p is the complement of $G_{XY}$ mod p, in $Z_p$.

Theorem 5: A cycle is allowed by a multifunction pipeline iff $(F_{XY} \text{ mod } p) \cap (G_{XY} \text{ mod } p) = \phi$, or equivalently iff $(F_{XY} \text{ mod } p) \subseteq H_{XY}$ mod p, $\forall$ X,Y in the set of function names present in the cycle. □

The generalization of the definition of compatibility is straightforward, except that each integer must be suffixed with an appropriate function name. Thus two elements $i_X$ and $j_Y$, such that $i,j \in Z_p$ and $j>i$, are said to be compatible if $(j-i) \in H_{XY}$ mod p. The following are generalizations of Lemma 2.1 and Theorem 2.

Lemma 6.1: Two elements $i_X$ and $j_Y$ such that $i,j \in Z_p$ are compatible iff $(j-i)$ mod $p \in H_{XY}$ mod p. □

Theorem 6: Given a cycle with period p, all possible rows which are allowed by the cycle are:

row $\{(i+i_1p)_X, (j+j_1p)_Y, \ldots\}$ $\forall$ nonnegative integers $i_1,i_2,j_1,j_2 \ldots$ and $\forall$ compatibility classes $\{i_X, j_Y, \ldots\}$. □

The maximal compatibility classes can be formed in a manner similar to the one for single function pipelines. As an example take again the cycle $(1_A,1_B,2_A)$ whose G sets were formed earlier. The allowable usage interval sets are: $H_{AA}$ mod $4=\{2\}$; $H_{AB}$ mod $4=\{0,1\}$; $H_{BA}$ mod $4=\{0,3\}$; $H_{BB}$ mod $4=\{1,2,3\}$. The maximal compatibility classes containing $0_X$ are: $\{0_A,2_A\}$, $\{0_B,1_B,2_B,3_B\}$, $\{0_A,0_B,1_B\}$, $\{0_B,3_A,3_B\}$.

A compatibility class $C_1$ is said to cover another class $C_2$ if for each function, the number of elements of that function type in class $C_1$ is greater than or equal to the number of elements of the same function type in class $C_2$. In the above example, $\{0_A,0_B,1_B\}$ and $\{0_B,3_A,3_B\}$ cover each other. The same definition for cover applies among rows and also between a row and a compatibility class. Now we have the generalization of Theorem 4.

Theorem 7: For a cycle, a multifunction pipeline can be made allowable by delaying some computation steps iff each row of the reservation table is covered by at least one compatibility class of the cycle. □

Now it is a simple matter to formulate the problem of making a pipeline allowable. In a multifunction pipeline different functions have different execution times. Let D(X) be the added execution delay to function X. The objective function can be any function of the D(X)'s, which is nondecreasing in each D(X); e.g., some linear combination of D(X)'s with positive coefficients. Let $d_{ij}(X)$ be the number of elemental delays to be inserted at the input of a step of function name X in cell (i,j). Let I and J be the number of tows and columns in the reservation table. The added execution delay for a function X can be expressed as

$$D(X) = \sum_{0 \le j < J} \left( \max_{0 \le i < I} d_{ij}(X), \right)$$

While the constraints can be written from Theorem 5, the usage interval between $X_{ab}$ and $Y_{ac}$ can be expressed as: [(the distance of $Y_{ac}$ from column 0) - (distance of $X_{ac}$ from column 0)]. Thus we have the following set of constraints.

Integer $d_{ij}(X) \ge 0$

and $[ ( \sum_{0 \le j < c} \max_{0 \le i < I} d_{ij}(Y)+d_{ac}(Y)+c)$

$$-( \sum_{0 \le j < b} \max_{0 \le i < I} d_{ij}(X)+d_{ab}(X)+b)] \bmod p$$

$$\in H_{XY} \bmod p. \text{ for each pair } \langle X_{ab}, Y_{ac} \rangle.$$

From property 6 we can see that we need construct only one constraint per pair without regard to the magnitudes of b and c. The algorithm to obtain an optimum soltuion is the same as Algorithm B.

## V. Concluding Remarks

We have presented the allowability characteristics of pipelines and cycles. We know the structure of all allowable pipelines for a given cycle. It is seen that one can utilize a pipeline fully by adding noncompute segments to make it allowable with respect to a perfect cycle. For nonperfect cycles, the pipeline can still be made allowable if every row of the reservation table is covered by at least one compatibility class of the cycle.

For single function pipelines, constant latency cycles were shown to be perfect. Thus a single function pipeline can always be utilized fully with the use of an appropriate constant latency cycle.

For multifunction pipelines, there is no straightforward procedure to construct a perfect cycle, given a mix of functions to be executed. However, if a cycle is given, it can always be tested for its perfectness with the use of compatibility classes. Cycles which are most likely to be perfect are those having evenly spaced task initiations, as well as a fairly regular pattern of functions. These cycles have a small set of initiation intervals and hence one has more freedom in choosing an allowable usage interval. For the same reason, these cycles are also most likely to require a small number of noncompute segments in making a pipeline allowable.

For increasing the throughput beyond what would result due to the full utilization of a pipeline, segment replication must be done. Segment replication is also a viable alternative to noncompute segments if the costs are comparable. For a cost effective design, segment replication and addition of delays should be considered simultaneously.

## References


1. E.S. Davidson, "The design and control of pipelined function generators," Proc. 1971 Int. IEEE Conf. on Systems, Networks and Computers, Oaxtepec, Mexico, January 1971.
2. L.E. Shar, "Design and Scheduling of Statically Configured Pipelines," Tech. Report No. 42, Digital Systems Lab., Stanford University, Sept. 1972.
3. A.K. Winslow, "Task scheduling in a class of pipelined systems," Report R-633, Coordinated Science Lab, Univ. of Illinois-Urbana, Nov. 1973.
4. E.S. Davidson, L.E. Shar, A.T. Thomas, and J.H. Patel, "Effective Control for pipelined computers," Proc. Compcon Spring 1975, pp. 181-184, Feb.1975.
5. D.E. Knuth, "Estimating the efficiency of Backtrack programs," Mathematics of Computation, Vol.29, no. 129, pp. 121-136, Jan. 1975.

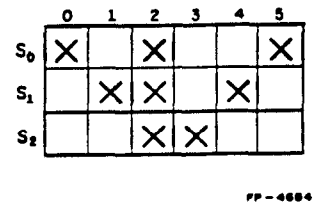




Figure 1. Reservation table

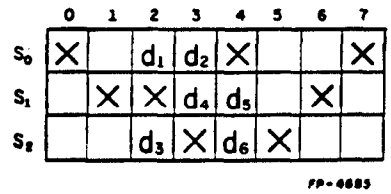



Figure 2. Delaying parallel computation steps

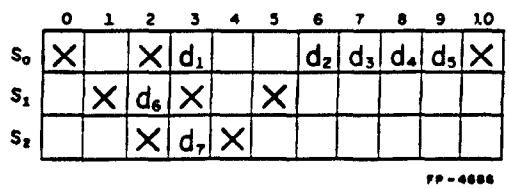



Figure 3. Making a pipeline allowable for cycle (1,5)

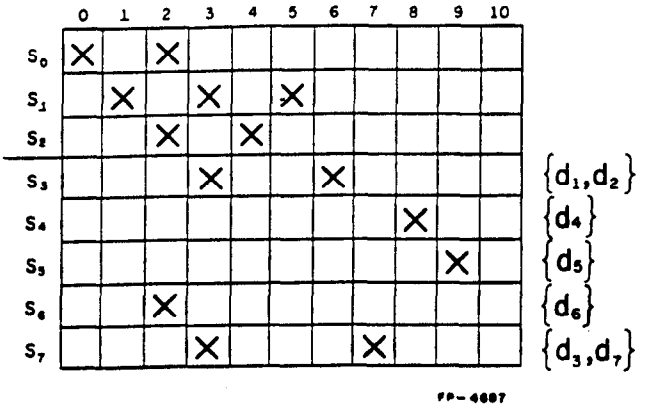



Figure 4. Assignment of elemental delays to noncompute segments

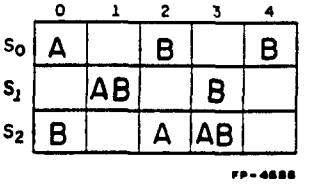
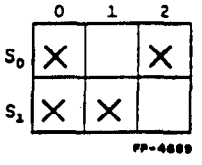



Figure 6. Reservation table for a multifunction pipeline




cycle (2). $\underline{H} \bmod 2 = \{1\}$

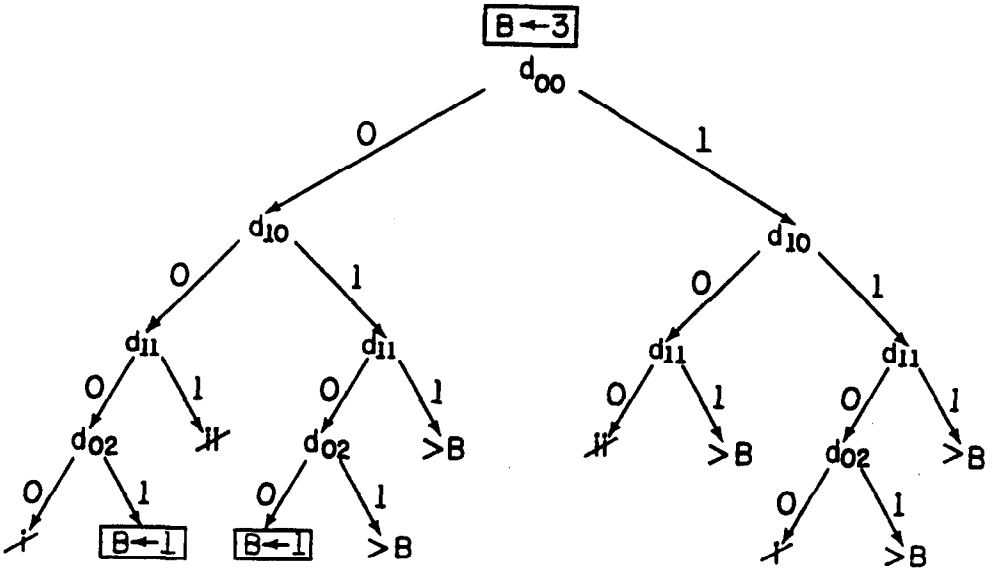
Added delay:

$$D = \max\{d_{00}, d_{10}\} + d_{11} + d_{02}$$

Constraints:

(i) $[2 + \max\{d_{00}, d_{10}\} - d_{00} + d_{02} + d_{11}] \bmod 2 \in \{1\}$.

(ii) $[1 + \max\{d_{00}, d_{10}\} - d_{10} + d_{11}] \bmod 2 \in \{1\}$.




Optimum solutions are: 1. $d_{00} = d_{10} = d_{11} = 0.\ d_{02} = 1.$

2. $d_{00} = d_{11} = d_{02} = 0.\ d_{10} = 1.$

Figure 5. Making the pipeline allowable for cycle (2): A branch-and-bound search for optimum solutions.