THROUGHPUT ENHANCEMENT IN MULTIPROCESSOR ARCHITECTURES FOR PIPELINING AND DIGITAL SIGNAL PROCESSING APPLICATIONS

Sukhamoy Som
ECE Department, Old Dominion University
Norfolk, Virginia 23529-0246
and
Meghanad D. Wagh
EE & CS Department, Lehigh University
Bethlehem, PA 18015

ABSTRACT This paper is concerned with throughput enhancement for pipelining and digital signal processing in a multiprocessor environment. A common objective in pipelining and digital signal processing is the repeated execution of the same computational job consisting of a set of computational operations with high throughput or sampling rates. For a good performance and avoidance of internal conflicts, the concurrent computational operations of successive data sets of a computational job should be properly scheduled. This scheduling problem is known to be intrinsically difficult and a member of the NP complete class of problems. In this paper, heuristic suboptimal scheduling algorithms are developed whose execution time is a polynomial function of the number of items to be scheduled. Insertion of delay is used as a basic tool for better utilization of hardware and thereby increasing the throughput. Rescheduling of computational jobs are directed to architectures consisting of arbitrary number of processors. Simulation results are presented.

I. INTRODUCTION

Pipelining is now widely used in the design of high speed computers in order to overcome the intrinsic speed limitations imposed by the technology [1, 2, 3]. A pipeline is defined to be a collection of processors or hardware stages which can work simultaneously. A Computational job is carried out by splitting the job into various operations and executing the operations in stages in an ordered sequence. Processors are normally special purpose for low cost. A pipeline is efficiently used only for repeated execution of a computational job because operations for successive data sets can be overlapped. In digital signal processing, a number of sensors provide the input data to the multiprocessor computer which then analyzes the input data by some predefined set of computational operations. Several characteristics required of such computers include repeated (usually periodic) execution of the same computational job and very high sampling rates [4, 5, 6].

The topic of this paper is the data control strategy for multiprocessor architectures where only one computational job is carried out repeatedly for several consecutive data sets. The problem is to adjust the data flow such that the same processor (stage) is not accessed by more than one data set simultaneously. The data flow pattern for each data set will be assumed to be same so that control of hardware is simplified. This data flow pattern can be easily represented through the use of a reservation table whose rows are labeled with processors and columns with time units [7]. Figure 1 shows a typical reservation table of a job execution on an architecture with four processors P_0 through P_3 where the processors can be identical or different. They can be special purpose or general purpose hardware doing computations at either fine grain or large grain level. Also the processors may have the same speed or different speed of operations. Each processor may require one or more time units to process the operands. The use of a processor in any time unit is indicated by marking the appropriate cell of the table with a cross. An operation on a processor at time t can only be done if all the operations on all the processors up to time (t-1) are completed for that data set. Crosses in the same column indicate parallel computation and are therefore independent of each other. Multiple crosses in the same row of the reservation table may either indicate a complex operation, processor reusage or a slow processor. The number of crosses in each row need not be same. Once assigned, a processor completes a computational task without preemption or wait so that multiple crosses belonging to the same computational operation should always remain consecutive.

As an example, consider the execution of a polynomial by the Central Arithmatic Logic Unit (CALU) of TMS320 series of microprocessors as shown in Figure 2 [8]. The architecture consists of two processors, namely one 16 x 16-bit parallel multiplier and one 32-bit arithmatic logic unit. The multiplier is capable of computing a 32-bit product in one machine cycle. The arithmatic logic unit (ALU) can add, subtract, and perform logical operations most of which requires only one clock cycle. A 32-bit accumulator is always the destination and primary operand for all ALU operations. Now suppose a polynomial f(z) is to be computed repeatedly in this architecture for successive values of z. Let f(z) be a second degree polynomial given by

$$f(z) = a_0 + a_1 * z + a_2 * z^2$$

As both the multiplier and arithmatic logic unit operates on two inputs only, f(z) is reorganized by Horner's rule as follows [9], $f(z) = (a_2 * z + a_1) * z + a_0$.

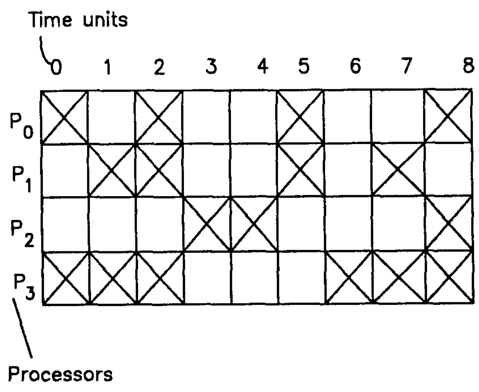


Figure 1. A typical reservation table.

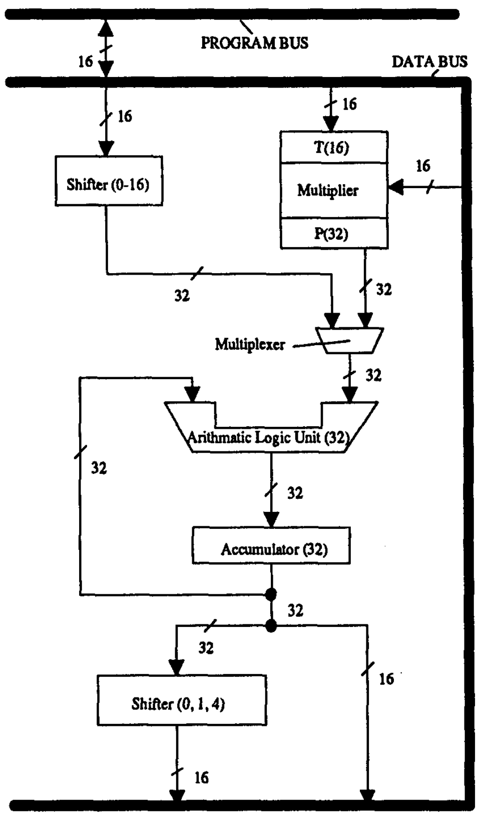


Figure 2. Central Arithmatic Logic Unit (CALU) for the TMS320 series of microprocessors.

The corresponding reservation table is shown in Figure 3. P_0 and P_1 are the multiplier and the ALU, respectively. As the multiplier is as fast as the ALU and assuming equal data transfer times, It is logical to assume that both multiplication and addition requires equal time, say one time unit. For a data set, P_0 multiplies a_2 with input data z and then the product is added to a_1 by P_1. The result is multiplied by z followed by addition of a_0. The reservation table indicates that computation in time unit 2 (multiplication by z) cannot be done unless all computations for time units 0 and 1 are complete. Both rows of reservation table have two crosses. These sequence of computation steps are to be followed for every data set.

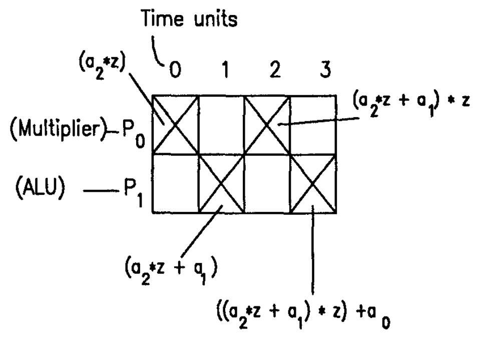


Figure 3. Reservation table for a polynomial.

The initiation interval between the successive data sets should be such that there is no conflict of hardware requirement between the data sets. This initiation interval and hence the interval between successive outputs, is known as latency of a pipeline or sampling interval in digital signal processing. It is also known from the work of Shar [10] that minimum average latency for an pipeline is always greater than or equal to the maximum number of crosses in any single row of the reservation table. Patel and Davidson [11] have shown that a pipeline can achieve minimum average latency and constant interval between initiations and outputs by inserting delays in the rows of a reservation table. This is very important for digital signal processing as external systems usually require sampling of inputs at regular intervals.

It has been shown in [7] that a reservation table with constant latency L will not have collision iff the crosses in any row occur at time units which are distinct modulo L. The basis for restructuring reservation table is that crosses in a row can be made to occur at distinct modulo L time by delay insertion without altering precedence constraints. Patel and Davidson [11] have obtained a branch and bound algorithm which optimizes the delay insertion. This minimization of delays is a known NP complete problem [7]. Thus their procedure to achieve the optimum throughput requires a complete search of all possible solutions. Although the scope of the required search can be reduced by establishing constraints and bounds, the branch and bound algorithm [11] still has a worst case complexity of the order of N^N, where N denotes the maximum number of crosses in any row. However, this level of complexity for scheduling algorithm is unacceptable for most run-time or compile-time scheduling. Also, since the added delay does not affect the overall throughput but only worsens the execution time for any particular data set, it is felt that the delay minimization could be traded for computational complexity of the rescheduling. The approach taken in this paper is therefore based on a search for heuristic procedures to reschedule a reservation table in polynomial time to achieve maximum throughput rate by addition of slightly larger delays. These scheduling algorithms are to be the guiding principles for the operating system of multiprocessor architectures to allocate computational operations to processors.

From Figure 3, as there are two crosses in each row of the reservation table, function f(z) has minimum average latency of 2. To illustrate this, data sets cannot be executed with a throughput of 1 as described in Figure 4. Data set 3 cannot be started until data time unit 4 as the multiplier is not available. From the reservation table, the latency value oscillates between 1 and 3 resulting in an average latency of 2. However, it should be possible to achieve a constant latency (L) of 2 with a periodic sampling of input signal z at 2 time intervals. But the reservation table of Figure 4 contains two multiplications at time units 0 and 2 and therefore it is not possible to inject next data set at time unit 2. Hence, a one time unit delay is added heuristically after computing (a_2 * z + a_1) in the reservation table as shown in Figure 5. In general, a scheduling algorithm is required to determine the proper number and location of delays. A constant latency of 2 is now achieved as shown in Figure 6. Therefore maximum possible throughput is achieved at the expense of increasing computing time for individual data sets by one time unit. Also, this may require additional registers to hold intermediate results for overlapping data sets.

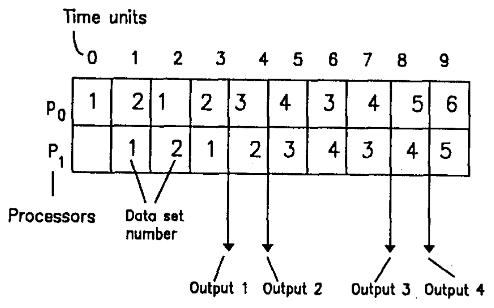


Figure 4. Execution of the polynomial for multiple data sets.

In Section II of the paper, a new concept, Difference Triangle, is introduced for keeping track of relative distance between each pair of crosses in the rows of a reservation table. The Difference Triangle has acted as a basic building block in all the rescheduling algorithms of this paper. Rescheduling of pipelines with stages of equal speeds are considered in Section III. In Section IV, simulation results are discussed.

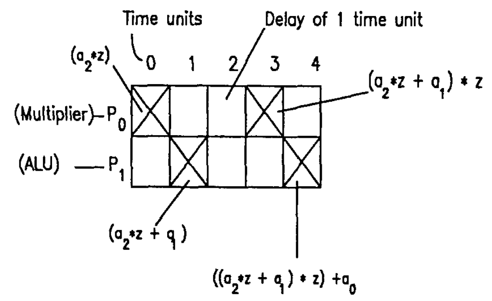


Figure 5. Modified reservation table for polynomial f(z).

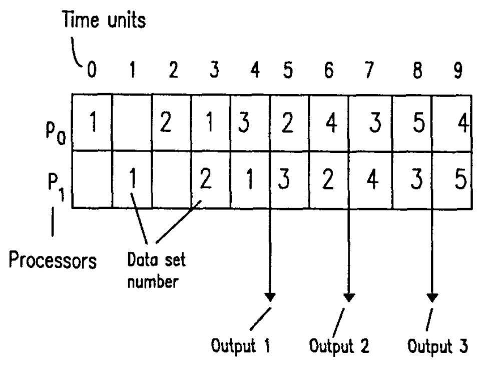


Figure 6. Periodic execution at minimum latency.

II. DIFFERENCE TRIANGLE

The basic problem of avoiding collisions in a reservation table consists of rescheduling the computational job such that all the crosses in a row are located at time units distinct modulo L, where L is the latency. In addition, the precedence constraints in the reservation table should also be preserved. In this section, a new concept, Difference Triangle, is presented for rescheduling reservation tables in polynomial time. The objective is to generate heuristic scheduling algorithms which will optimize throughput by insertion of delays in the reservation table. The overall execution time of a reservation table may be suboptimal but the complexity of rescheduling algorithms has to be a polynomial function of the maximum number of crosses in any row of the reservation table.

Consider a single row reservation table with N crosses. Hence the latency is N. In order to avoid collisions and achieve optimum throughput, the crosses in the reservation table are to be rescheduled so that no two crosses are separated by a time interval multiple of N. This is to be done by adding delays before crosses in the reservation table. Let X(i), i = 0, 1, ... N - 1 be a sequence of N elements where X(i) indicates the execution time of the i-th cross. Also, let Y(i), i = 0, 1, ... N-1 be a sequence of N elements where Y(i) indicates the added delay before the i-th cross. In order to maintain precedence constraints in X sequence, Y(j) must be greater than or equal to Y(i) if j > i. Mathematically, this scheduling problem for a single processor can therefore be stated as follows. Given a monotonically increasing sequence X(i), i = 0, 1, .. N-1, it is required to modify it by adding a monotonically nondecreasing sequence Y(i), i = 0, 1, .. N - 1 such that for no i and j, [X(i) + Y(i)] - [X(j) + Y(j)] is an integer multiple of N.

We now define a new concept, Difference Triangle, for keeping track of relative distance between each pair of sequence elements. The Difference Triangle D of height as well as base of N-1 is defined as

$$D(k, 0) = [X(k + 1) - X(k)] \bmod N, \quad k = 0, 1, .. N-2$$
$$D(k, i) = [D(k, 0) + D(k-1, i-1)] \bmod N, \quad k > 0;\ i = 1, 2, .. k.$$

It is obvious from the definition that

$$D(k, i) = [X(k + 1) - X(k-i)] \bmod N, \quad 0 \leq k \leq N-2,\ 0 \leq i \leq k.$$

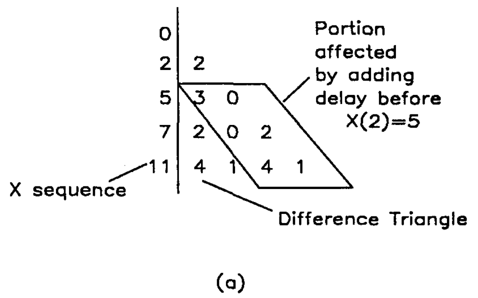
Thus if all the X components are distinct modulo N, no entry in D will be zero (0). Figure 7(a) shows an example of Difference Triangle for the X sequence (0, 2, 5, 7, 11) where N is 5. Note that the first column written here is the X sequence and is not part of D.

Now if all the modified X sequence components are distinct modulo N, the Difference Triangle based on modified X sequence will not have a 0 (zero). So the rescheduling problem can also be stated as follows. Modify X sequence by introducing delay between properly chosen X components to eliminate all the zeros from the Difference Triangle. The elements affected by the introduction of such a delay are enclosed in a parallelogram bounded by the row and a line parallel to the diagonal originating at the X component which is delayed. For example, the zero in the second row of Figure 7(a) can be eliminated by inserting appropriate delay (say 1 unit) before X(2) = 5. This corresponds to the Y sequence (0, 0, 1, 1, 1). The modified X sequence is (0, 2, 6, 8, 12) with the modified Difference Triangle shown in Figure 7(b).

When the architecture has two or more processors, precedence constraints between operations on different processors need to be considered. If a delay of t time units is inserted before an operation on a processor, all subsequent operations on all processors have to be delayed by at least t time units.

III. SCHEDULING WITH DIFFERENCE TRIANGLES

In this section, the concept of Difference Triangle is used for scheduling of computational operations in multiprocessors. The treatment of different classes of architectures are divided in separate sub-sections. For simplicity, processors are assumed to execute computational operations in equal time which implies that each computational operation can be expressed by a single cross in the reservation table with respect to some basic time unit. Also, this is a realistic assumption for fine-grain dedicated processors such as the ALU and the multiplier of TMS320.



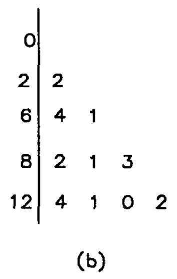


Figure 7. Example of a Difference Triangle.

III.1 Two Processor Scheduling

In this sub-section, reservation tables are restricted to two processors and equal number of operations on each processor. The rescheduling problem can be mathematically stated as follows. Given strictly monotonically increasing sequences X and Y of lengths N (hence latency L = N), it is required to modify them by adding monotonically nondecreasing sequences M_x and M_y such that (1) the relation between X and Y is maintained (i.e., if X(i) > Y(j) then $X'(i) = X(i) + M_x(i) > Y(j) + M_y(j) = Y'(j)$ and if X(i) < Y(j) then X'(i) < Y'(j) for any i, j pair). (2) All X'(i) mod N are distinct and (3) All Y'(i) mod N are distinct.

Modified X and Y sequences are collision free and are obtained by delaying certain operations in the original X and Y sequences without altering the precedence constraints. An algorithm is now described which uses properties of Difference Triangle to restructure the reservation table for two processors. The proof of existence and complexity of the algorithm is also provided.

Algorithm 1.

Step 1. Form the Difference Triangles D_x and D_y for sequences X and Y respectively.

Step 2. Find smallest k_x such that $D_x(k_x, i) = 0$; $0 \leq i \leq k_x$. Similarly find k_y from the Difference Triangle D_y. If no such k_x and k_y exist, the procedure is complete.

Step 3. Depending upon the relation between $X(k_x + 1)$ and $Y(k_y + 1)$ perform one of the steps 4, 5, or 6.

Step 4. If $X(k_x + 1) < Y(k_y + 1)$ or if k_y does not exist, find largest nonzero t less than N missing in row k_x of Difference Triangle D_x. Modify X and D_x as: $X(j+1) \leftarrow X(j+1) + N-t$;

$D_x(j, i) \leftarrow (D_x(j, i) - t) \bmod N$; $k_x \leq j \leq N-2$ $j-k_x \leq i \leq j$. Find the smallest k_1 such that $Y(k_1+1) > X(k_x+1)$. If no such k_1 exists, go to Step 2. Otherwise modify Y and D_y as: $Y(k+1) \leftarrow Y(k+1) + (N-t)$; $D_y(k, i) \leftarrow (D_y(k, i) - t) \bmod N$; $k1 \leq k \leq N-2$, $k - k1 \leq i \leq k$. Go to Step 2.

Step 5. If $Y(k_y + 1) < X(k_x + 1)$ or if k_x does not exist, find the largest nonzero t less than N missing in row k_y of D_y. Modify Y and D_y as $Y(j + 1) \leftarrow Y(j + 1) + (N - t)$; $D_y(j, i) \leftarrow (D_y(j, i) - t) \bmod N$; $k_y \leq j \leq N - 2$, $j - k_y \leq i \leq j$. Find the smallest k1 such that $X(k1 + 1) > Y(k_y + 1)$. If no such k1 exists, go to Step 2. Otherwise modify X and D_x as $X(k + 1) \leftarrow X(k + 1) + N - t$; $D_x(k, i) \leftarrow (D_x(k, i) - t) \bmod N$; $k1 \leq k \leq N - 2$, $k - k1 \leq i \leq k$ and go to Step 2.

Step 6. If $X(k_x + 1) = Y(k_y + 1)$, find the largest nonzero t less than N missing in k_x or k_y rows of D_x and D_y respectively and carry out following modifications: $X(j + 1) \leftarrow X(j + 1) + N - t$, $D_x(j, i) \leftarrow (D_x(j, i) - t) \bmod N$, $k_x \leq j \leq N - 2$; $j - k_x \leq i \leq j$. $Y(k + 1) \leftarrow Y(k + 1) + N - t$, $D_y(k, m) \leftarrow (D_y(k, m) - t) \bmod N$; $k_y \leq k \leq N - 2$, $k - k_y \leq m \leq k$. Go to Step 2.

Proof. A Difference Triangle of N number sequence will have (N - 1) rows. Each row will have at most (N - 1) elements. Hence one will always be able to find a nonzero number t less than N missing from any row which has a zero. Adding N - t to that row and portions of the following rows (see Figure 7(a)) eliminates all the zeros in that row and does not create any new ones in earlier rows. Thus the execution of Step 4 increases the k_x obtained in Step 2 by at least one. Similarly Step 5 increases the k_y obtained in Step 2 by at least one. Step 6 ensures that either k_x or k_y or both increase by at least one. A delay insertion in Step 4, 5, or 6 changes in the order of N^2 elements of the Difference Triangles. Since $0 \leq k_x$, $k_y \leq N - 2$, the algorithm terminates after at most 2 * (N - 1) cycles. Hence, the Difference Triangles can be made zero free in the order of N^3 computations. The precedence constraints are maintained by altering both Difference Triangles for every modification in any one Difference Triangle. []

Example. Let the initial X and Y sequences be 0, 2, 3, 5, 7 and 0, 1, 4, 6, 10. Clearly N = 5. The corresponding Difference Triangles are shown in Figure 8(a). The first rows of Difference Triangles having zeros are given by $k_x = 2$, $k_y = 2$. Since $X(k_x + 1) = 5$ and $Y(k_y + 1) = 6$, Step 4 of the algorithm is carried out. Largest nonzero t less than 5 missing in row k_x of D_x is 4. The modified difference triangles are given in Figure 8(b). Now $k_x = 3$, $k_y = 3$, $X(k_x + 1) = 8$, and $Y(k_y + 1) = 9$. Hence t = 4. Modified D_x and D_y are shown in Figure 8(c). D_x is now free from zeros and therefore k_x does not exist. $k_y = 3$ and t = 2. As $Y(k_y + 1) > X(k_x + 1)$, only Y and D_y need to be modified. The final zero-free Difference Triangles are shown in Figure 8(d). The rescheduled X sequence is 0, 2, 3, 6, 9 and the Y sequence is 0, 1, 4, 7, 13.

III.2 Architectures with Three or More Processors

The attention is now focussed to scheduling a reservation table with three or more processors with equal number of operations and equal processing time. The algorithm presented here is basically the extension of Algorithm 1 presented before. Let the number of processors be m with N operations on each processor. Minimum latency of the reservation table is N. The algorithm is presented below.

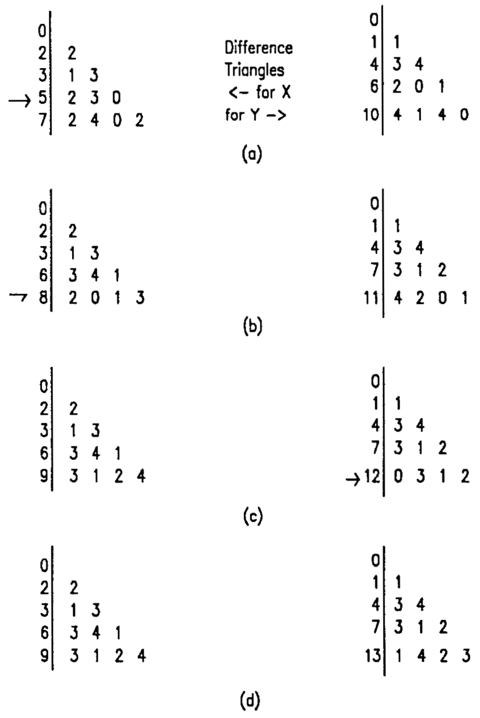


Figure 8. Rescheduling example using Algorithm 1.

Algorithm 2.

Step 1. Form the difference triangles $D_0(k, i)$, $D_1(k, i)$, ... $D_{m-1}(k, i)$ corresponding to sequences X_0, X_1, ... X_{m-1}.

Step 2. Find smallest k_j such that $D_j(k_j, i) = 0$; $0 \leq i \leq k$, $0 \leq j \leq m-1$. If no such k_j exists, the procedure is complete.

Step 3. Compute $X_j(k_j + 1)$ for all J if k_j exists. Find the minimum of them. Let the corresponding j be denoted as s. If there is more than one such s, go to Step 5.

Step 4. If there is only one such s, the zero at row k_j of D_s needs to be eliminated. This can be done similar to Algorithm 1. Go to Step 2.

Step 5. Find the largest nonzero number t less than N missing in row k_s of Difference Triangle D_s for all s. Modify all X_s and D_s as in Step 6 of Algorithm 1. Modify the rest of the sequences and Difference Triangles as follows. $D_j(k, i) = (D_j(k, i) - t) \bmod N$. $Q \leq k \leq N - 2$; $k - Q \leq i \leq k$, where Q denotes the smallest integer such that $X_j(Q + 1) > X_s(s+1)$. Go to Step 2.

Proof. The proof and complexity of Algorithm 2 are similar to that of Algorithm 1.

Example. Suppose the given X_0, X_1, and X_2 sequences are: X_0: 0, 1, 3, 6, 7, X_1: 0, 3, 5, 10, 12, and X_2: 2, 4, 8, 9, 12. N = 5. The corresponding Difference Triangles are shown in Figure 9(a). Denoting the row with first zero by k_j; $k_0 = 2$, $k_1 = 1$, $k_2 = 2$, $X(k_0 + 1) = 6$, $X(K_1 + 1) = 5$ and $X(k_2 + 1) = 9$. Hence the X_1 sequence needs to be modified first. The largest nonzero t less than 5 missing in row 1 of D_1 is 4. The modified Difference Triangles are shown in Figure 9(b). Now $k_0 = 3$, $k_1 = 2$, $k_2 = 1$, $X_0(k_0 + 1) = 8$, $X_1(k_1 + 1) = 11$, $X_2(k_2 + 1) = 9$. Hence X_0 sequence needs to be modified first. The largest nonzero t less than 5 missing in row 3 of D_0 is 4. The modified sequences and Difference Triangles are shown in Figure 9(c). Since now D_0 and D_1 are zero-free, k_0 and k_1 do not exist. The only zero in D_2 is at $D_2(3, 2)$. Largest nonzero t less than 5 missing in row 3 of D_2 is 1. The modified sequences and zero-free Difference Triangles are shown in Figure 9(d). The rescheduled pipeline may be based upon the sequences X_0: 0, 1, 3, 7, 9, X_1: 0, 3, 6, 12, 14, and X_2: 2, 4, 10, 11, 18.

```
 0|                 0|                 2|
 1| 1               3| 3               4| 2
 3| 2 3             5| 2 0             8| 4 1
 6| 3 0 1          10| 0 2 0           9| 1 0 2
 7| 1 4 1 2        12| 2 2 4 2        12| 3 4 3 0

 Difference         Difference         Difference
 Triangle of X0     Triangle of X1     Triangle of X2

                        (a)

 0|                 0|                 2|
 1| 1               3| 3               4| 2
 3| 2 3             6| 3 1             9| 0 2
 7| 4 1 2          11| 0 3 1          10| 1 1 3
 8| 1 0 2 3        13| 2 2 0 3        13| 3 4 4 1

                        (b)

 0|                 0|                 2|
 1| 1               3| 3               4| 2
 3| 2 3             6| 3 1            10| 1 3
 7| 4 1 2          12| 1 4 2          11| 1 2 4
 9| 2 1 3 4        14| 2 3 1 4        14| 3 4 0 2

                        (c)

 0|                 0|                 2|
 1| 1               3| 3               4| 2
 3| 2 3             6| 3 1            10| 1 3
 7| 4 1 2          12| 1 4 2          11| 1 2 4
 9| 2 1 3 4        14| 2 3 1 4        18| 2 3 4 1

                        (d)
```

Figure 9. Rescheduling example using Algorithm 2.

III.3 Unequal Loading of Processors

If the stages in a pipeline are loaded unequally, then the number of crosses in the corresponding rows of the reservation table are not the same. In this case, the complete utilization of all stages is not possible [2]. The best performance is obtained when at least one stage is fully utilized. Minimum latency possible is the number of operations of any stage which has the maximum number of operations compared to all other stages.

Algorithm 3.

Step 1. Compute latency L as $L = \max(N_0, N_1, \ldots N_{s-1})$ where N_i is the number of crosses in the i-th row of an s row reservation table.

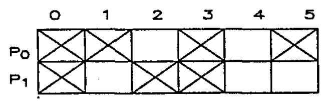
Step 2. Add $(L - N_i)$ fictitious crosses to the i-th row of the reservation table following the last cross in any row.

Step 3. Use Difference Triangles to make the crosses in each row occur at times distinct modulo L while maintaining the precedence constraints of the reservation table.

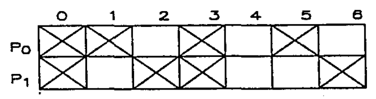
Step 4. Remove all fictitious crosses from the final reservation table to get the rescheduled reservation table.

Proof. The proof and complexity are similar to that of Algorithm 1.

Example. A reservation table with unequally loaded processors is shown in Figure 10(a). Latency of the reservation table is 4. As there is one less operation on P_1 processor, one fictitious cross is added to the corresponding row following the last cross as shown in Figure 10(b). Figure 10(c) shows the sequences and Difference Triangles of this table. It is easy to see that kx = 2, ky = 2, X(kx + 1) = 5 and Y(ky + 1) = 6. The largest nonzero number less than 4 missing in row kx of Difference Triangle D_x is 3. D_x and D_y modified as per Algorithm 1 are shown in Figure 10(d). Now ky = 2, kx does not exist. Moreover, there are no elements of the current X sequence which are greater than Y(ky + 1) = 7. Hence the X sequence need not be modified. The final zero-free D_x and D_y are shown in Figure 10(e) from which the new schedules are X: 0, 1, 3, 6, and Y: 0, 2, 3.



(a)



(b)

```
 0|                Difference            0|
 1| 1              triangles             2| 2
 3| 2 3          <-- for X               3| 1 3
 5| 2 0 1            for Y -->           6| 3 0 2
                      (c)

 0|                                      0|
 1| 1                                    2| 2
 3| 2 3                                  3| 1 3
 6| 3 1 2                                7| 0 1 3
                      (d)

 0|                                      0|
 1| 1                                    2| 2
 3| 2 3                                  3| 1 3
 6| 3 1 2                                9| 2 3 1
                      (e)
```

Figure 10. Rescheduling of a reservation table with unequal loading of processors.

IV. SIMULATION RESULTS

In order to test the effectiveness of the Difference Triangle concept in developing heuristic suboptimal, polynomial time scheduling algorithms, simulations are carried out for multiprocessor architectures with two processors. It is assumed that both processors require one time unit to do their respective computations. Reservation tables are generated by a random number generator and scheduled by Algorithm 1 which has a complexity of N^3. N here denotes the maximum number of operations required on a processor for a data set. Simulation software, written in Pascal, uses fifty reservation tables which are equally loaded in both P_0 and P_1, for number of operations between 5, 7, and 9. All the computations are done on a TI 990 model 26 minicomputer. The average expansion of a reservation table and average scheduling time for fifty reservation tables are computed. The results obtained are tabulated in Table I. As the minimum measurable time on this computer was one second, total scheduling time for 50 reservation tables with N = 5 could not be measured. From the results, scheduling time is minimal and appropriate for compile-time implementation. The reservation table do get extended resulting in a higher storage requirements and higher execution time for individual data sets. However, latency is optimal, constant and both processors are fully utilized.

TABLE I

RESCHEDULING BY ALGORITHM 1

Number of Operations (N)	Average Expansion by Algorithm 1	Average Scheduling Time in milliseconds	
5	6.61	--	Too small to be measured
7	12.47	40	
9	19.86	80	

CONCLUSIONS

Optimum utilizations of hardware is a basic requirement for any scheduling algorithm of multiprocessor architectures. One way of achieving this is insertion of delays at appropriate places of a reservation table. The scheduling algorithms presented in this paper are suboptimal in that execution time for a particular data set is not minimized. On the other hand, this scheduling can be achieved within a polynomial time of the maximum number of operations in any processor.

In the various algorithms presented in this paper, Difference Triangle concept has acted as a basic building block. Difference Triangle is simply a straight forward approach of keeping track of relative distances between operations in a processor. Delay insertion is used to avoid collisions in the architecture. All the algorithms obtained are heuristic in nature but are proved to yield an optimum throughput in an architecture. They have a complexity of the order of N^3 and therefore execute much faster than those suggested in earlier literature. N here denotes the maximum number of operations required on a processor for a data set.

A number of other heuristic scheduling algorithms are developed [13]. The Difference Triangle concept also is applied on architectures with processors requiring unequal time for operations [13]. Several topics can be the subject of continuing and/or future research. To name a few interesting problems, scheduling multiple computational jobs concurrently, timing and control requirements for insertion of delays, a trade-off between delay insertion and rescheduling time, and more studies on architectures with processors of unequal speed of operations deserve attention.

REFERENCES

1. Hui Cheng, "Vector Pipelining, Chaining, and Speed on the IBM 3090 and Cray X-MP," Computer, pp. 31-46, September 1989.
2. George M. Papadopoulos and David E. Culler "Monsoon: an Explicit Token-Store Architecture," Proceedings of the 17th Annual International Symosium on Computer Architecture, IEEE Computer Society Press, pp. 82-91, May 1990.
3. L. E. Shar and E. S. Davidson, "A Multiminiprocessor System Implemented Through Pipelining," Computer, pp. 42-51, February, 1974.
4. S. Som, R. R. Mielke, and J. W. Stoughton, "Strategies for Predictability in Real-Time Data-Flow Architectures," Presented at the 11th IEEE Real-Time System Symposium, Orlando, Florida, December 5-7, 1990.
5. Kesab. K. Parhi and David G. Messerschmitt, "Static Rate-Optimal Scheduling of Iterative Data-Flow Programs via Optimum Unfolding," IEEE Transactions on Computers, pp. 178-195, vol. 40, No. 2, February 1991.
6. E. A. Lee and D. G. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing," IEEE Transactions on Computers, vol. 36, pp. 24-35, January 1987.
7. P. M. Kogge, The Architecture of Pipelined Computers, Advanced Computer Science Series, McGraw-Hill Book Company, 1981.
8. Texas Instruments, "First-Generation TMS320 User's Guide," April 1988.
9. E. Horowitz and S. Sahni, "Fundamentals of Computer Algorithms," Computer Science Press, pp. 426-428, 1978.
10. L. E. Shar, "Design and Scheduling of Statically Configured Pipelines," Digital Systems, Lab Report, Stanford University, California, September, 1972.
11. J. H. Patel and E. S. Davidson, "Improving the Throughput of a Pipeline by Insertion of Delays," IEEE/ACM 3rd Annual Symposium on Computer Architecture, pp. 159-163, 1976.

12. J. D. Ulman, "NP-Complete Scheduling Problems," Journal of Computer and System Sciences, vol. 10, pp. 384-393, 1975.
13. Sukhamoy Som, "Suboptimal Algorithms for Improvement of Pipeline Through Insertion of Delays," M.E. Thesis, Old Dominion University, Norfolk, VA, August 1984.
14. C. V. Ramamoorthy and H. F. Li, "Some Problems in Parallel and Pipeline Processing," Proc. Compcon, IEEE, pp. 177-180, 1975.
15. J. H. Patel "Pipelines with Internal Buffers," Proc. of the 5th Annual Symposium on Computer Architecture, pp. 249-254, April 1978.
16. David Bernstein, Michael Rodeh, and Izidor Gertner, "On the Complexity of Scheduling Problems for Parallel/Pipelined Machines," IEEE Transactions on Computers, vol. 38, September 1989.
17. Sukhamoy Som, "Performance Modeling and Enhancement for the ATAMM Data Flow Architecture," Ph.D. Dissertation, Old Dominion University, Norfolk, VA, May 1989.