Nuclear Instruments and Methods in Physics Research A242 (1986) 450–454
North-Holland, Amsterdam

REAL-TIME MULTIPARAMETER PULSE PROCESSING WITH DECISION TABLES *

K. HULL

EG&G Energy Measurements, Inc., Santa Barbara Operations, Goleta, California, USA

H. GRIFFIN

University of Michigan, Department of Chemistry, Ann Arbor, Michigan, USA

Decision tables offer several advantages over other real-time multiparameter, data processing techniques. These include very high collection rates, minimum number of computer instructions, rates independent of the number of conditions applied per parameter, ease of adding or removing conditions during a session, and simplicity of implementation. Decisions table processing is important in multiparameter nuclear spectroscopy, coincidence experiments, multiparameter pulse processing (HgI_2 resolution enhancement, pulse discrimination, timing spectroscopy), and other applications can be easily implemented.

1. Introduction

Multiparameter collection in real-time presents users with a large number of problems. These include slow collection rates, restrictions on the condition sets, inflexibility of conditions once set, and transportability difficulties, to name a few. Collections using decision table techniques solve most of these problems and allow new applications to be tried in real time. During attempts to improve the performance of mercuric iodide used as gamma-ray spectrometers at EG&G Energy Measurements, Inc. [1], the need to do on-line, real time correction of two-parameter data rekindles the interest in decision tables. In previous work [2] the problem of collecting gamma–gamma coincidence data on a small computer was tackled. It was found that in order to allow moderate speed collections the amount of processing time spent handling the data had to be reduced. To do this the standard serial processing techniques were replaced with a parallel approach that involved the use of decision tables.

2. Decision table concept

The normal method of complex data processing in a computer program is to check the parameters against the determining conditions one at a time; that is, test if the parameters meet the conditions for process one; if not, try for process two; etc. This is called a decision tree and is a serial process. Many problems implemented this way can be implemented in a parallel fashion called a decision table. This method decides which process to execute in one (or nearly one) operation. To do this a table is created, where each location contains an action to be taken in case this combination of parameter values occurs. For instance, this action may be a subroutine jump. Selection of one of the actions is made by creating an index into this table. This is often a major problem, since choosing the wrong index generally results in a much larger table.

An example to illustrate this technique is for a gamma–gamma coincidence experiment. In this hypothetical experiment, it requires too much space to collect the entire two-dimensional spectrum. Let us say we want to record spectra from a gamma detector (B) in coincidence with 100 windows on another detector (A). The result will be 100 spectra – called gates – each in coincidence with a different window. Coincidence events give a pair of parameters (A, B). The collection process involves taking a pair of data and deciding if they belong in one of the windows. In this case then there are 101 possible decisions for what to do with this pair. There are 100 possible spectra that it might be in, and there is a possibility that it is not in any one of the gates. The table can now be made and indexed by parameter A. The table would be the length of parameter A, say 4096 locations. In each of the slots in this table would be one of the 101 possible decisions. Let us take a more detailed look at the processing of a few pairs of hypothetical data. Fig. 1 shows both the standard decision tree and the new table methods. In this example, let us assume that gate 12 is defined by parameter A values of 156 to 165 and that no gate is defined for parameter A values of 154 to 155 and 166 to

* This work was performed under the auspices of the U.S. Department of Energy under Contract No. DE-AC08-83NV10282.

0168-9002/86/$03.50

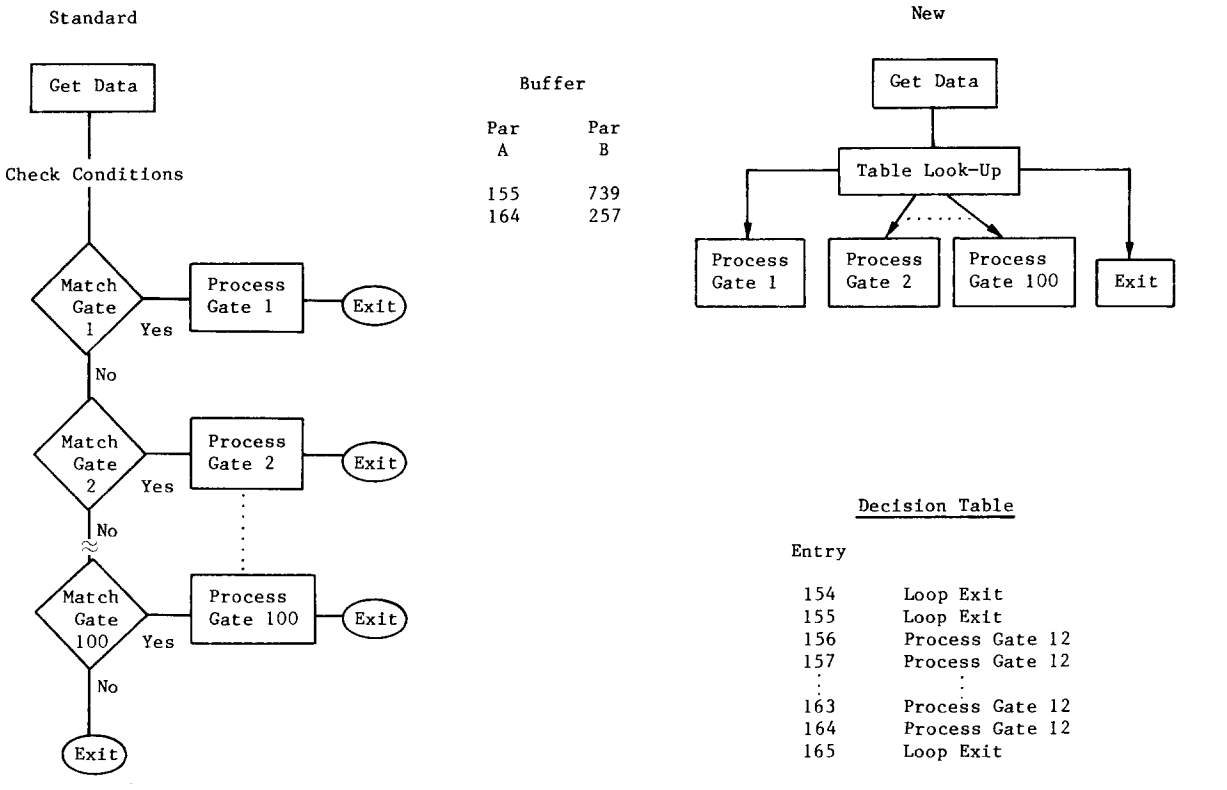

Fig. 1. Block diagram of two-parameter processing showing both standard decision tree and new decision table method.

166. A pair of data are retrieved from the buffer and parameter A is found to be 155. With the serial method, a series of 100 checks would be executed until no gate was selected and the next pair tried. For the decision table technique, the 155 entry in the decision table is examined, causing execution of the loop exit and examination of the next pair. For these data, parameter A is found to be 164. With the serial method, after 12 checks the subroutine for gate 12 is executed. In the current approach the 164 entry is retrieved, which causes execution of the gate 12 subroutine. For both techniques the process is continued until all data in the buffer are examined and processed. The decision table method involves fewer steps and therefore would make the processing faster.

This technique can be extended to cover many other situations. A simple extension to the coincidence case is to add a random coincidence, background correction process. In this case more windows are set corresponding to areas where there would not be any true coincidences. When the parameter matches one of these windows, a new background process is performed where a decrement of the appropriate location in the selected gate is done. This adds 100 more decisions but no new gates. Another application is for spectral enhancement, in which case a two-parameter collection is performed. The collected spectrum needs to have two-dimensional channel shifts done to correct for nonuniformities in detector response; that is, the data at (14,26) may be translated to (18,34), etc. This type of translation can be handled using a two-dimensional table where the correct address is stored at the original one; that is, in location (14,26) an instruction to increment the location (18,36) would be stored. The problem with this type of implementation is that the table gets quite large.

3. Implementation

The decision table process is usually implemented as part of the multiparameter device driver. This is generally part of, or used by, a main program written in a high-level language. The main program may consist of a series of subroutines designed to set up the decision table and handle the interactive dialog with the user (allowing display of data, plotting, storage, and parameter altering, for example). The decision table is applied in the interrupt handler to maximize data processing rates. To maintain high data throughput, the data are usually collected in buffers using a DMA approach.

When the buffer is full, then the entire buffer is processed. This is usually implemented using two buffers such that while one is being processed the other is being filled.

The decision table method has been implemented for several systems [2,3]. In these cases, the decision table is initialized during program startup by filling every slot with a jump to the loop exit. The program then waits for the user to respond interactively (to set the decision table, start the collection, etc.). Defining the gates, applying the conditions, or setting up of the decision table just consists of placing a jump to a process routine in one or more locations of the table. This means that no programming changes are required to change or alter conditions.

Implementation of the decision table process lends itself directly to assembly language, which is often necessary if speed is crucial. In the simplest case the decision table would be just a dispatch table, a list of addresses of the routines to be executed; an example is shown in fig. 2. In this case the same gating example is used as before. This shows a possible implementation using an idealized assembly language. There is an initialization procedure (not shown), which would set the initial buffer address and set the buffer count. There is also a finish procedure (not shown) that would generally create a smooth transition out of the device driver. The average processing time is determined by the amount of time to handle all events plus the initialization and finish time divided by the number events processed. As long as the buffer contains several hundred events, the net processing time is only slightly increased by the initialization and finishing parts. The decision procedure starts by doing the look-up, which consists of determining the address of the entry in the table. This is accomplished by loading the indexing parameter into a register and adding the table's beginning address to it. The decision has now been determined and is carried out by execution of the instruction located at this resultant address. If the decision is to take no action, the

```
loop::  LOAD  Reg  Par A,Indirect   *Get current parameter A
        ADD   Reg  Decision offset  *Determine entry into Decision Table

*
*           Execute Decision
*

*  Several possible method exists, many processors have a execute
*  instruction and can be implemented in one instruction

       XCT     Reg, Indirect

*  On machines that do not have an execute, the operation is loaded and
*  stored in the next available location and then executed

       LOAD    Reg, Indirect          *Get operation - would be a NOP for
                                      *  EXIT or a JUMP SUBroutine for a
                                         gate process.
       STORE   Reg  Exec              *Store it in next location
Exec:: NOP                            *Execute it.

*
*              LOOP EXIT
*

       INC     Par A                  *Address next parameter A in buffer
       INC     Par B                  *Address next parameter B in buffer
       INC     Count                  *Increment buffer count
       CHECK   Count                  *Check for last item in buffer
       JUMP    True  Finish           *Branch if done to finish routine
       JUMP    False LOOP             *Otherwise process next pulse

*
*      Sample Gate Process
*

Gaten::LOAD    Reg  Par B,Indirect    *Get parameter B
       ADD     Reg  Offset N          *Find current datum address
       INC     Reg, Indirect          *Increment it
       RETURN
```

Fig. 2. Processing part of the two-parameter collection driver with decision processes shown.

instruction is a branch to the loop exit or often just a no-operation instruction. The loop exit increments the buffer address and the count, then checks the count for the end of the buffer; if it is not finished, it branches back to do more decisions; if it is done it branches to the buffer finish (not shown). If the decision is a gate process, this process just finds the correct location in the correct slice (gate) and increments it. This is done by simply adding the start address to parameter B and incrementing the location defined by this address. For this simplistic example, it takes eight instructions to handle the exit case and 12 including the gate processing. For typical processors and instruction time of 1 or 2 μs per instruction is common. This yields typical times of about 10–20 μs per case handled, or collection rates of 50–100K pairs per second. If higher speed machines are used, more complex processes can still be performed in 100 μs, thus keeping a minimum rate of at least 10K multiparameter pairs processed per second. If slower micros are used, this simple coincidence process can still be performed in 100 μs or less.

Implementation in a higher level language is possible but not recommended if speed is important. On some computers the computed GOTO is implemented as a dispatch table. If this type of computer is used, high processing rates are still possible; one such example is shown in fig. 3. In this case the table is a simple table of indexes used in the computed GOTO statement. The parameter A in this case is used as an index into the decision table. The GOTO index is retrieved and the computed GOTO executed. On machines that have computed GOTOs implemented as table look-ups, this can be quite fast (tens of microseconds) so that the total time can still be less than 100 μs. On slower machines, it is possible to use other procedures to speed up the process while still using all high-level language programming. One technique uses the EXTERNAL statement in FORTRAN to put the addresses of the decision process subroutines into a variable which can be called through a dummy subroutine call.

4. Other uses

An interesting example of the use of decision tables is a further compression of the results and the space necessary for the coincidence case. In the last example a large amount of space was still needed for program space and results (at least 40K values for the 100 slices of 4096 channels each plus the program). This can be compressed to just two values per coincidence combination: the gross coincidence strength and the random coincidence strength. The coincidence combination is all possible combinations for the number of peaks

```
      INTEGER TABLE(len), PAR-A(lenb), PAR-B(lenb)
c
c     Table is the decision table and is the length of parameter B.
c
c     In this example TABLE(i) can take on values of 0 - n where n
      is the maximum gate number

c     TABLE is defined as:
c       0 - EXIT
c       1 - Process Gate 1
c       N - Process Gate N

c     Initialization section not shown

      DO I=1,NUM_IN_BUFF                !Process buffer of data
      GOTO(91,100,...),TABLE(PAR_A(I))  !Do look-up

c     process gate 1

100   Data(PAR_B(I),1) = Data(PAR_B(I),1) + 1
      GOTO 91

c     process gate 2
   .
   .
   .
c     process gate N

N00   Data(PAR_B(I),N) = Data(PAR_B(I),N) + 1

91    ENDDO
```

Fig. 3. Example of FORTRAN implementation of decision table method using computed GOTOs.

selected taken two at a time; that is, peak 1 with peak 1, peak 1 with peak 2, etc. This means that for N peaks defined, the space required for results would be N squared plus N values, because there are $(N^2+N)/2$ combinations and two values per combination. (The extra N values are for error terms for the self-coincidence cases, which can be eliminated.) The actual process is done by performing an additional decision table look-up on the B parameter. This is, in the first operation we reduced the L^2 two-dimensional spectrum to a length L one-dimensional spectrum. Now this one-dimensional spectrum is reduced in the same manner to one point, the coincidence strength. For the random coincidence strength (background correction), instead of decrementing this the gross count, a separate location is acquired. The implementation only involves adding an additional table for the second detector. (Only one table is required if both detectors have the same gain.) This time though, the tables could be filled with zeros for no action and a signed peak number: plus for gross coincidence, minus for background. The processing would involve checking for a zero in each look-up; one zero and processing of that pair stops. To find the address to increment, the pair of peak numbers are used as matrix address (i.e., say found peak 26 for parameter A and 8 for parameter B, then form the address 26,8). This address is ordered so the row is the lowest value (from 26,8 to 3,26) for positive pairs. For background events the row is ordered to be the highest. The only exception is for negative pairs on the diagonal ($-1,-1$; $-2,-2$; etc.), which have to be assigned to the extra N locations. Once the address is determined, all that is done is to increment the value stored at this location. In this way, in the same space (40K values) over 200 peaks could be defined and collected simultaneously.

One last short example is a compression of the spectral enhancement case. In that, it was stated that it could be done with a two-dimensional table look-up. This table gets very large fast. For a 1024-by-1024 channel spectrum it would require over a million values. Therefore what can be done is to seek a new index (computed from the parameter pair) that uses a compressed table. For the spectral enhancement case it was found that the correction was dependent on the depth of interaction. By applying some additional arithmetic steps to create a normalized depth parameter, the sorting operation required only one table of 1024 values. The difference was that with the megaword table only one step was required to get the result; with the new computed indexing parameter a 1024-word table was required plus the addition of a minimum one divided, one multiply, one subtract, and a few shifts. Some speed was sacrificed to save memory.

5. Conclusion

The basic idea of using decision tables has been presented. Through their use, the number of steps necessary to determine the correct processing procedure has been reduced to nearly one for the two-parameter case. For applications of more than two parameters, the method expands easily as $n-1$ decision table for n-parameter. An example of a three-parameter experiment, using decision tables has been implemented [3] for the case of a delayed coincidence experiment. For these n-parameters cases, the steps required for the decision then approaches $n-1$. Also, it has been shown that the decision table process is relatively easy to implement in most languages. In assembly language it only requires a few basic instructions found on any common processor, making it relatively easy to transport.

A large variety of applications can be implemented using this technique; the range is limited only by the user's creativity. All of these features make decision tables a viable choice in solving a variety of collection problems where the complexity of processing makes the serial process too slow or cumbersome and may make new techniques possible.

References

[1] A. Beyerle, V. Gerrish and K. Hull, these Proceeding (6th Symp. on X- and Gamma-Ray Sources, Ann Arbor) Nucl. Instr. and Meth. A242 (1986) 443.
[2] K. Hull, Thesis, University Microfilms (1979).
[3] E. Kao, Thesis, University Microfilms (1977).