MULTIPLE INSTRUCTION ASSOCIATIVE PROCESSING

by

Keki B. Irani and Jamshed D. Mulla

Technical Report 78-13

THE UNIVERSITY OF MICHIGAN

Computer, Information and Control Engineering Program

MULTIPLE INSTRUCTION ASSOCIATIVE PROCESSING

Keki B. Irani
and
Jamshed D. Mulla
Systems Engineering Laboratory
The University of Michigan
Ann Arbor, Michigan 48109

Abstract -- This paper introduces the concept of multiple instruction associative processing as a departure from conventional SIMD associative processing.

A formal definition of the conventional associative processor (CAP) is provided. The CAP represents current SIMD associative processing architectures. Next, the multiple instruction associative processor (MIAP) is introduced and defined. The MIAP eliminates the drawbacks of the CAP caused by the undue synchronization required for SIMD execution of associative procedures. A multiprocessor computer architecture for implementing the MIAP is also described.

The performances of the CAP and MIAP systems are analyzed. The performance of each system is measured in terms of the length of the instruction sequence required to execute a given associative procedure. Algorithms for computing the measure for each system for general procedures are given. Results are provided for some special families and specific examples of associative procedures.

Introduction

The term associative processing is used here to mean the manipulation of a set of data items by a common process in such a way that operations performed on a particular datum are dependent on its value or some function thereof.

Early associative processors (AP's) were simple extensions of associative memories. These extensions were achieved by incorporating elementary processing functions into each word of the memory. Present AP's, such as PEPE [1], have processing elements (PE's) that are capable of executing a broad repetoire of arithmetic and logical operations on their data memories.

The common limitation of all present AP's is that they are all Single Instruction Multiple Data (SIMD) systems. Since the PE's are essentially only execution units, the total control in the system remains in a main CPU that broadcasts operations to the set of PE's from a single instruction stream.

This work was supported jointly by Rome Air Development Center under contract F30602-76-C-0029 and Army Ballistic Missile Defense Advanced Technology Center under contract DASG-60-78-C-0073.

In this paper, we investigate a more general system consisting of programmable PE's. We call the corresponding concept of associative processing, multiple instruction associative processing. We will first outline a general representation scheme for an associative process. Then we will define the conventional associative processor (CAP) that represents the current AP's. Next, the concept of multiple instruction associative processing will be introduced and a formal definition of the MIAP will be provided. A multiprocessor computer architecture to implement the MIAP will be described. Finally, the performance of the CAP and MIAP systems will be analyzed and compared and some results for certain classes of associative procedures will be given.

The APL language [2] is used frequently in the paper to represent certain functions and operations of associative processors.

Definition 1: An associative process is a two tuple (A, Q) where,

(1) A is a set of n data items. Let

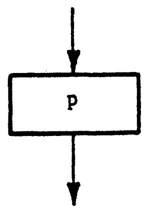
$$A = \{A_1, A_2, \ldots, A_n\}$$

and $A_i \varepsilon \mathcal{W},\ i \le 1 \le N$

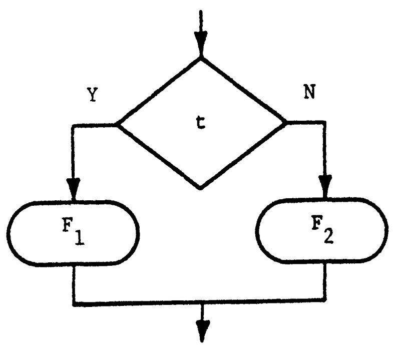
$\mathcal{W}$ is the set of all possible values of a data item.

(2) Q is a common procedure applied to each item A_i of the data set A. Conventional flowchart structures will be used to represent the procedure Q graphically.

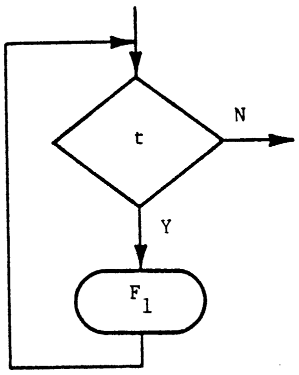
A flowchart is defined recursively as follows. If F_1 and F_2 are flowcharts, then so are the structures (i) through (iv) shown below.



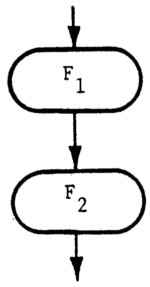
(i)



(ii)



(iii)



(iv)

In flowchart (i), the operation p is a iapping

$$p:\mathcal{W}\to\mathcal{W}$$

.nd its execution causes the data item A_i to be 'eplaced by $p(A_i)$.

In flowcharts (ii) and (iii), the operation corresponds to a characteristic function

$$t:\mathcal{W}\to\{0,1\}\ .$$

:xecution proceeds to the Y path of the flowchart .f $t(A_i) = 1$, otherwise it proceeds to the N path.

■

Note that the set of flowcharts defined above :orresponds to the set of D-charts defined in [3]. 'he flowcharts describe the procedure carried out ın a single datum A_i. For the associative pro- :ess, the same procedure is applied to all data .tems in A independently.

Conventional Associative Processing

All present AP's can be classified as CAP's. 'EPE [1] and STARAN [4] are two particular exam-)les.

A CAP consists of two major subsystems. The ıeart of the CAP is an associative processing ıemory (APM) that stores and manipulates the set)f data items to be processed. The APM can select data items by associative search opera- :ions and can process the selected subset in)arallel.

The APM receives its instructions from the)ther subsystem of the CAP called the global con- :rol unit (GCU). Besides broadcasting operations :o the APM, the GCU performs other functions to synchronize the APM elements. We will first lefine the APM and then describe the GCU and its :unctions.

Definition 2: The associative processing nemory (APM) of the CAP is a four tuple $(A,R,\mathcal{T},\mathcal{P})$ where:

1) $A = (A[1], A[2], \ldots, A[n])$ is a memory array of n words. $\mathcal{W}$ is the set of all possible values for a single word $A[i]$ and $\mathcal{A}$ is the set of all possible values of the array A.

2) $R = (R[1], R[2], \ldots, R[n])$ is a response register of n bits. $\mathcal{R}$ is the set of all possible values of the register R.

3) $\mathcal{T}$ is a set of search functions

$$\mathcal{T} = \{T_1, T_2, \ldots, T_N\}$$

$$T_i : \mathcal{A}\times\mathcal{R}\to\mathcal{R}\ , \qquad 1 \le i \le N\ ,$$

where each T_i has a corresponding mapping t_i,

$$t_i:\mathcal{W}\to\{0,1\}$$

and $T_i((A[1], A[2], \ldots, A[n]),$

$$(R[1], R[2], \ldots, R[n])) =$$

$$(r_1, r_2, \ldots, r_n)$$

where

$$\underset{1\le j\le n}{r_j} = \begin{cases} 1 & \text{if } t_i(A[j]) = 1 \quad \wedge \quad R[j] = 1 \\ 0 & \text{otherwise} \end{cases}$$

The operation of a search function T_i causes $T_i(A,R)$ to become the new contents of the response register R. That is, each search function T_i performs the operation represented by the following APL statement:

$A \leftarrow (R \wedge (TI\ A))$

4) $\mathcal{P}$ is a set of processing functions

$$\mathcal{P} = \{P_1, P_2, \ldots, P_M\}$$

$$P_i : \mathcal{A}\times\mathcal{R}\to\mathcal{A}\ , \qquad 1 \le i \le M\ ,$$

Each processing function P_i has a corresponding function p_i such that

$$p_i : \mathcal{W}\to\mathcal{W}$$

and

$P_i((A[1], A[2], \ldots, A[n]),$

$$(R[1], R[2], \ldots, R[n])) =$$

$$(a_1, a_2, \ldots, a_n),$$

where

$$\underset{1\le j\le n}{a_j} = \begin{cases} p_i(A[j]) & \text{if } R[j] = 1 \\ A[j] & \text{otherwise} \end{cases}$$

The operation of a processing function P_i causes $P_i(A,R)$ to become the new contents of the memory array A. That is, each processing function P_i corresponds to the following APL statement,

$A \leftarrow ((R \times (PI\ A)) + ((\sim R) \times A))$

The global control unit (GCU) of the CAP performs several overall functions to control the activity of the APM. The following are the main functions performed by the GCU:

(i) The GCU possesses a stack which it uses to store the contents of the response register R of the APM.

Let S[0], S[1], ... represent the stack on which the response register is saved. The GCU can perform three operations involving the stack and the response register.

(a) The PUSH operation causes the current value of R to be pushed on the stack.

(b) The POP operation causes the value of the response register R to be replaced by the topmost value on the stack.

(c) The COMP operation causes the value of R to be replaced by its complement relative to the top of the stack, i.e.,

$R \leftarrow (\sim R) \wedge S[TOP]$

where S[TOP] is the most recently pushed value on the stack.

(ii) In addition to broadcasting the sequence of search and processing functions to the APM, the GCU can perform certain conditional branches by testing all response bits of the APM. In particular the GCU can branch to a particular instruction if the current response register is all zeros. We will denote this jump as

IF NONE GOTO x

where x is the label of the target instruction.

(iii) The GCU can also perform unconditional branches. This instruction is denoted

GOTO x .

The above control functions are sufficient for executing a general associative process. Similar control mechanisms have been proposed [5] and are also present in existing AP's such as PEPE [6].

We can now define an algorithm to determine the "program" for a CAP for any valid associative procedure flowchart as described in Definition 1.

Algorithm 1: Given a flowchart F, we can determine the sequence of operations $S_c(F)$ required to execute F on a CAP by applying the following rules recursively.

(i) If F is a flowchart of type (i), then

$S_c(F) = P$

(ii) If F is a flowchart of type (ii), then

$S_c(F) = \text{PUSH, T,} S_c(F_1)\text{, COMP,} S_c(F_2)\text{, POP}$

(iii) If F is a flowchart of type (iii), then

$S_c(F) = \text{PUSH, L1:T, IF NONE GOTO L2,}$

$S_c(F_1)\text{, GOTO L1, L2: POP}$

(iv) If F is a flowchart of type (iv), then

$S_c(F) = S_c(F_1),\ S_c(F_2).$

■

Consider the flowchart representation of a program shown in Figure 1.

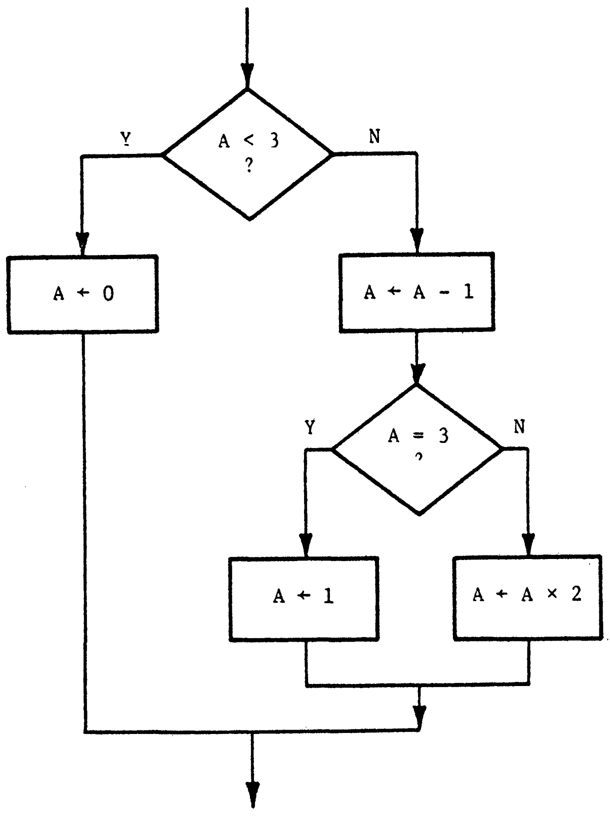


Figure 1: A Sample Flowchart

Using Algorithm 1, we can determine the CAP program for the above flowchart as:

$S_c(F) = \text{PUSH, } T_1, P_1, \text{COMP, } P_2, \text{PUSH, } T_2,$

$P_3, \text{COMP, } P_4, \text{POP, POP}$

where the search operations T_1, T_2 and the processing operations P_1, P_2, P_3, P_4 are defined in APL as follows.

$T_1 \equiv R \leftarrow (R \wedge (A < 3))$
$T_2 \equiv R \leftarrow (R \wedge (A = 3))$
$P_1 \equiv A \leftarrow (R \times 0) + ((\sim R) \times A)$
$P_2 \equiv A \leftarrow (R \times (A - 1)) + ((\sim R) \times A)$
$P_3 \equiv A \leftarrow (R \times 1) + ((\sim R) \times A)$
$P_4 \equiv A \leftarrow (R \times (A \times 2)) + ((\sim R) \times A)$

There are three major drawbacks in the conntional associative processing system. First, large percentage of the operations executed by e GCU are overhead operations necessary to synronize the activity of the APM. The POP, PUSH, d COMP operations are required solely to ensure at the proper subset of PE's is active when the arch and processing operations are broadcast to e APM.

Second, since the CAP is a SIMD machine and der the control of a single control unit, only subset of the PE's in the APM can participate instruction execution at any instant. As shown Flynn [7], conditional branching has a severe trimental effect on the performance of SIMD chines such as the CAP.

Finally, since the GCU broadcasts a single struction stream, it must ensure that all ssible paths of the program are traversed.

One of the objectives of the extensions of e basic CAP is to minimize the amount of synronization required between PE's of the APM. the case of the CAP, such synchronization is tal because the control of the system lies tirely in the GCU. This restriction is relaxed the multiple instruction associative processor allowing each PE to fetch and execute search d processing operations independently and ynchronously. Synchronization of the PE's by a ntrol unit is forced only when it is absolutely cessary.

Multiple Instruction Associative Processing

The most important change from conventional multiple instruction associative processing is at the restriction of SIMD operation is relaxed. stead of PE's receiving the same associative structions broadcast by the GCU, each PE is graded to a programmable processor so that it n fetch, decode and execute instructions indendently. By providing each PE processor with a py of the set of instructions for an associative ocessing program, the system can execute it with eater efficiency and significant increase in eed. Since some global control instructions are ill required, the MIAP also contains a control it corresponding to the GCU. However, in this se, global synchronization operations on the 's are less frequent than in the case of the CAP.

Figure 2 shows a block diagram of an MIAP system.

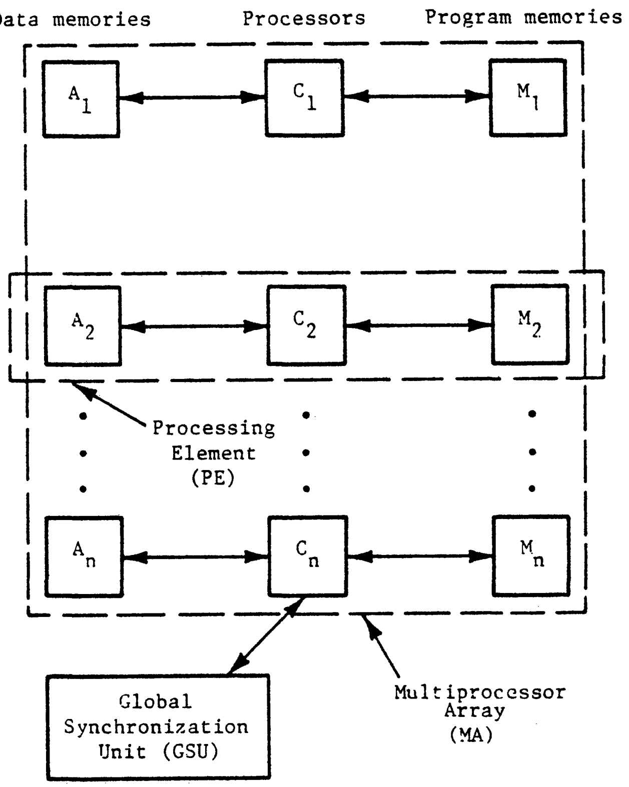


Figure 2: MIAP System Block Diagram

We will first define two subsystems, namely, the multiprocessor array (MA) and the global synchronization unit (GSU) and state conditions under which they constitute a multiple instruction associative processor.

Definition 3: A multiprocessor array (MA) is a four tuple (A, R, M, C) where:

1) A is a data memory array of n words.

$A = (A[1], A[2], \ldots, A[n])$.

$\mathcal{W}$ is the set of all possible contents of a single memory word A[i].

2) R is a response register of n bits.

$R = (R[1], R[2], \ldots, R[n])$

$R[i] \in \{0,1\}$, $1 \leq i \leq n$

3) M is a program memory array consisting of n program memories $M_1, M_2, \ldots, M_n$. Each M_i can contain the three types of instructions described below:

(i) A test instruction T can set the response bit R[i] depending on the result of a corresponding binary valued function t.

$$t : \mathcal{W} \to \{0,1\}$$

The execution of a test instruction T causes t(A[i]) to become the new value of the corresponding response register bit R[i].

(ii) A processing instruction P can alter the contents of the data word A[i] by applying a corresponding mapping p,

$$p : \mathcal{W} \to \mathcal{W}$$

The execution of a processing instruction P causes p(A[i]) to become the new contents of the corresponding data word A[i].

(iii) A branch instruction can transfer control to a target instruction either unconditionally or based on a test of the response bit R[i]. The two instructions are denoted

GOTO x ,

and

IF NOT GOTO x,

respectively, where x is the label of the target instruction and the conditional branch is executed if R[i] = 0.

4) C is a processor array consisting of n processors $C_1, C_2, \ldots, C_n$. Each C_i can execute instructions from its program memory M_i and thus manipulate its corresponding data memory A[i] and response bit R[i]. The processors can operate independently. ■

The global synchronization unit (GSU) performs two major functions to control the activity of the MA. It can initiate the execution of the MA processor programs by the respective processors. We will denote this operation as INIT. The GSU must also wait for all the MA processors to terminate their programs before continuing with any other operation. This operation will be denoted as WAIT. The GSU can perform normal sequential processing operations but these do not relate to the present discussion and will not be treated here.

Multiple instruction associative processing is performed by the MA and GSU pair when each program memory array M_i in the MA is loaded with the identical sequence of instructions. Whereas the PE's of the CAP either execute or ignore broadcasted instructions depending on the value of their data items, the processors of the MA select the appropriate sequence of instructions from the common program in their program memories depending on their data memory contents. Assuming that this program contains some conditional branches, the instruction sequence, and hence the data transformation operations performed by the individual processing element of the MA is data dependent and, therefore, associative.

We will refer to a system consisting of an MA and GSU under the conditions described above as a multiple instruction associative processor.

We can now define a corresponding algorithm for the multiple instruction associative processor to determine the GSU and MA "programs" for any valid flowchart.

Algorithm 2: Given a flowchart F, we can determine the sequence of instructions required to execute F on an MIAP as follows:

The GCU program for a flowchart consists of only two instructions, namely, INIT and WAIT. The INIT instruction initiates the MA program corresponding to the flowchart F. The MA program $S_p(F)$ can be determined by applying the following rules, recursively:

(i) If F is a flowchart of type (i), then

$$S_p(F) = P$$

(ii) If F is a flowchart of type (ii), then

$$S_p(F) = T,\ \text{IF NOT GOTO L1},\ S_p(F_1),\ \text{GOTO L2},\ \text{L1: } S_p(F_2)\ ,$$

where L2 is the label of the instruction following the sequence $S_p(F)$.

(iii) If F is a flowchart of type (iii), then

$$S_p(F) = \text{L1: } T,\ \text{IF NOT GOTO L2},\ S_p(F_1),\ \text{GOTO L1}$$

where L2 is the same as above.

(iv) If F is a flowchart of type (iv), then

$$S_p(F) = S_p(F_1),\ S_p(F_2)$$

■

We can now derive the MA program for the flowchart in Figure 1 by using the above algorithm. The program $S_p(F)$ is given by:

$$S_p(F) = T_1,\ \text{IF NOT GOTO L1},\ P_1,\ \text{EXIT},\ \text{L1: } P_2,\ T_2,\ \text{IF NOT GOTO L2},\ P_3,\ \text{EXIT},\ \text{L2: } P_4$$

The test and processing instructions for the ith processor are defined as follows:

$$T_1 \equiv R[I] \leftarrow (\ A[I] < 3\ )$$
$$T_2 \equiv R[I] \leftarrow (\ A[I] = 3\ )$$
$$P_1 \equiv A[I] \leftarrow 0$$
$$P_2 \equiv A[I] \leftarrow A[I] - 1$$
$$P_3 \equiv A[I] \leftarrow 1$$
$$P_4 \equiv A[I] \leftarrow A[I] \times 2$$

e use EXIT to denote GOTO instructions that ter-
inate execution of the MA program.

Comparing the above program with the corres-
onding CAP program, we see a significant
ecrease in the length of the instruction
equence. In particular, the longest sequence of
nstructions executed by the PE's of the MA is 6
ompared to the fixed CAP program of 12 instruc-
ions. A comprehensive performance analysis study
f these two systems for a broad class of associa-
ive processing problems is provided in a later
ection.

Referring back to the three major deficien-
ies of the CAP, we see that the MIAP system
voids all of them. First, since the PE's execute
ndependently, there is no need for global
nstructions to synchronize activity during execu-
ion of a flowchart program. This also eliminates
he need for PE's to be idle while waiting for
ther PE's to execute different instructions.
inally, since each PE selects its own path
hrough the program flowchart, the execution time
epends on the longest path and not on the total
raversal of the flowchart as in the case of the
AP.

The cost of using this system as opposed to
he CAP is the cost of the program memory array
o store copies of the PE programs. The cost of
oading the programs into the memory array is not
ignificant since program partitioning is a deter-
inistic problem and dynamic loading of programs
s not necessary.

An Implementation of an MIAP System

We now describe a multiprocessor system on
hich multiple instruction associative processing
an be implemented. This architecture is similar
o that described by Kober [8].

An overall block diagram of the system archi-
ecture is shown in Figure 3.
central processor (control unit) P_c is con-
ected to a memory M_c which is divided into equal
locks M_1, M_2, ..., M_n. Each M_i has an asso-
iated processor P_i. The M_i-P_i pairs serve the
unction of processing elements.

Each P_i can reference only its own M_i and
an execute programs and operate on data from it
ndependently of and asynchronously with all
ther P_i's. The P_c can also access all M_i's as
ell as its own private memory M_p.

The P_c can access the M_i's in two modes.
ne is the conventional mode where the set of
$_i$'s is perceived as one large contiguous memory
lock. The second is the "multiwrite" mode where
he P_c can write data into the same relative
ocation of a selected subset of M_i's simultan-
ously.

Although there is no direct data link
etween P_c and P_i's, there are direct control
ines connecting them. The P_c can instruct any
particular module processor or a selected subset
of P_i's to begin execution of a program at a
specified starting address stored within their
M_i's. After executing the programs, each P_i can
interrupt the P_c and pass its results or other
termination data through the module memory M_i.

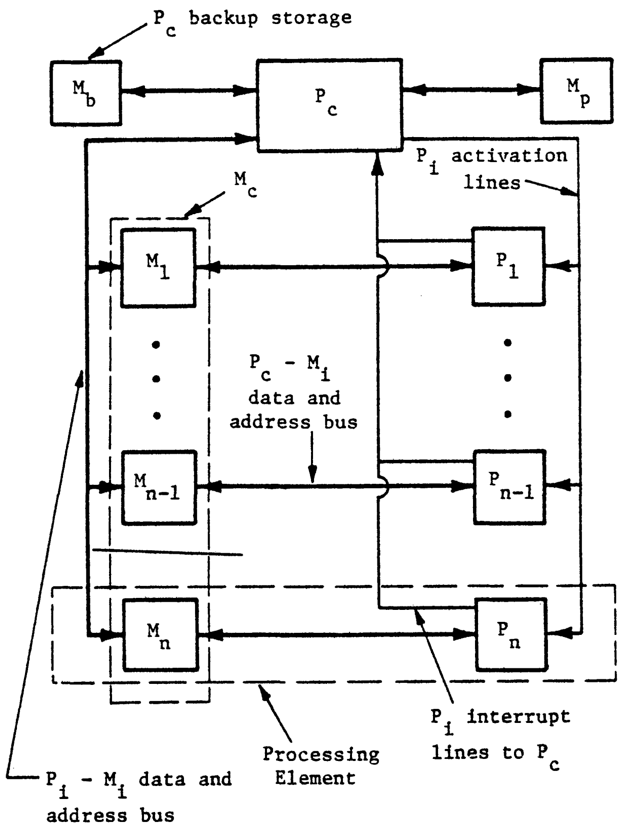


Figure 3: Multiprocessor Architecture for the MIAP

The Performance Measure

The performance measure used to compare the
CAP and MIAP systems will be the length of the
instruction sequences required to execute flow-
charts. In order to obtain quantitative results
of this analysis, certain assumptions will be
made while modelling the characteristics of
associative procedures represented by flowcharts.
These assumptions will be explicitly stated in
the following discussion as they are used.

For a given flowchart F, the performance
values $N_{C,F}$ and $N_{M,F}$ for the CAP and MIAP,
respectively, are the number of instructions
executed by each system from the corresponding
sequences $S_c(F)$ and $S_p(F)$. Since the actual
number of instructions executed for a particular
flowchart is dependent on data conditional
branching, the performance values $N_{C,F}$ and $N_{M,F}$
are random variables. The assumptions we make
about conditional branching probabilities of
certain flowchart structures will enable us to
determine distributions and expected values of
the variables involved.

We will first give an algorithm by which
the performance can be computed for any given
flowchart structure and then analyze some results

for certain families of structures. Due to the lack of space, details of computations are not provided in this paper but may be found in [9].

Algorithm for General Flowchart Structures

In the following algorithm we assume that the associative processors are operating on a data set of cardinality n. It is assumed that the data items are randomly selected from the space of allowable values.

Algorithm 3: Given a flowchart F, the performance values $N_{C,F}$ and $N_{M,F}$ for the CAP and MIAP, respectively, can be determined as follows:

$N_{M,F}$ is the length of the longest sequence from a set of n independent sequences executed by the n PE's of the MIAP. To determine $N_{M,F}$, the length of the sequence executed by a single independent PE, $N_{P,F}$ will be determined first. $N_{M,F}$ is given in terms of $N_{P,F}$ by

$$N_{M,F} = (N_{P,F})_{(n)}$$

where, $Y_{(n)}$ denotes the n-th order statistic [10] of the random variable Y. In other words, $Y_{(n)}$ is the largest of n independent samples of the random variable Y.

For a given flowchart F, $N_{C,F}$ and $N_{P,F}$ can be determined by applying the following rules recursively.

(i) If the flowchart F is of type (i), then

$$N_{C,F} = N_{P,F} = 1$$

(ii) If the flowchart F is of type (ii), then

$$N_{C,F} = 4 + N_{C,F_1} + N_{C,F_2}$$

If we assume that each branch of the flowchart F has equal probability of being executed by an arbitrary PE, then $N_{P,F}$ has the following distribution:

$$\Pr\{N_{P,F} = x\} = \frac{1}{2}(\Pr\{N_{P,F_1} = x - 2\}$$
$$+ \Pr\{N_{P,F_2} = x - 2\}) .$$

(iii) If the flowchart F is of type (iii), then

$$N_{C,F} = 4 + \sum_{i=1}^{X_{(n)}} (2 + (N_{C,F_1})_i)$$

and

$$N_{P,F} = 2 + \sum_{i=1}^{X} (2 + (N_{P,F_1})_i)$$

where $(N_{C,F_1})_i$ and $(N_{P,F_1})_i$ are the performance measures for the flowchart F_1 for the CAP and MIAP, respectively, during the ith iteration of the loop.

X is the random variable that characterizes the number of loop iterations required for a single datum. If we assume that at any iteration, the loop termination condition has probability p of being satisfied, then we can represent X by a random variable that is distributed geometrically with parameter p. Hence,

$$\Pr(X=i) = p(1-p)^i \qquad i = 0, 1, 2, \ldots$$

(iv) If flowchart F is of type (iv), then

$$N_{C,F} = N_{C,F_1} + N_{C,F_2}$$

and

$$N_{P,F} = N_{P,F_1} + N_{P,F_2} .$$

Results

We will now analyze and compare the performance of the CAP and MIAP for two families of flowchart structures. The two major families will consist of loop and loop-free programs. Finally, performance values for some mixed flowchart examples will be compared.

Loop Programs

In this section we will analyze the performance values for two classes of programs containing loop structures, namely, programs with a sequence of loops and programs with a nested pair of loops.

Loop Sequences

Consider a program that consists of L consecutive loop structures as shown in Figure 4 below.
We assume that the sequence consists of L consecutive loops and that the data set cardinality is n. We also assume that an arbitrary datum requires X iterations of a loop to be executed, where X is distributed geometrically with parameter p.

The performance values for the flowchart F in Figure 4 are then:

$$N_{C,F} = 4L + 3 \sum_{i=1}^{L} X_{(n)}$$

and

$$N_{M,F} = 2L + 3 \left(\sum_{i=1}^{L} X\right)_{(n)}$$

Figure 5 shows the plot of the ratio of the expected values of $N_{C,F}$ and $N_{M,F}$ against L for p = 0.3 and various values of n.

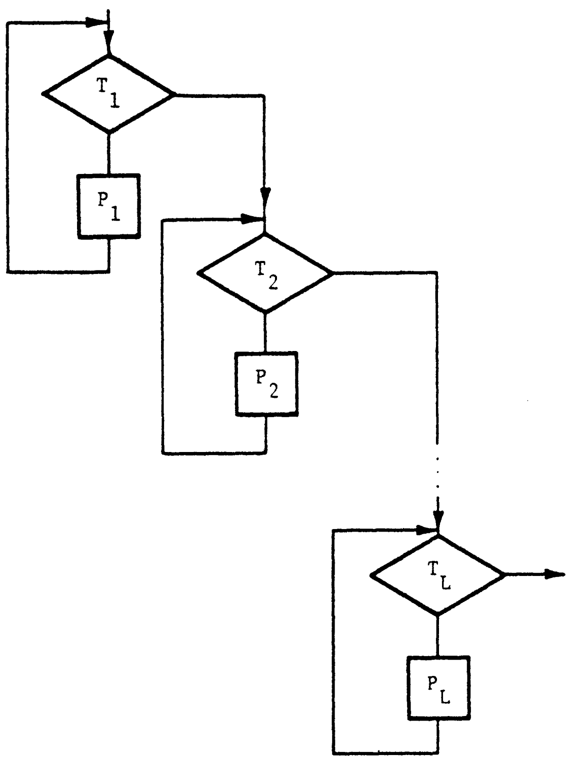


Figure 4: Loop Sequence Program

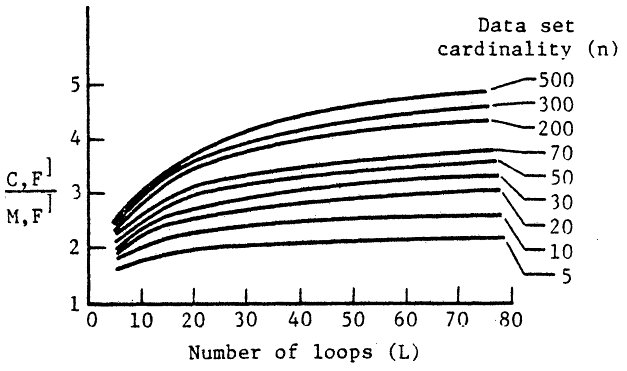


Figure 5: Expected Performance Ratio for Loop Sequence Programs (p = 0.3)

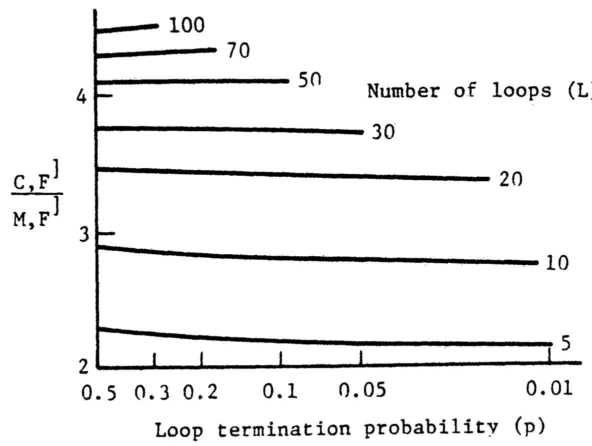


Figure 6: Expected Performance Ratio for Loop Sequence Programs (n = 200)

Figure 6 shows similar results for a fixed n = 200 and various values of L.

As shown by the plots in Figures 5 and 6, the relative performance of the MIAP over the CAP improves for longer sequences of loops and larger cardinality of the data set.

Nested Loops

Consider a program that consists of a pair of nested loop structures as shown in Figure 7.

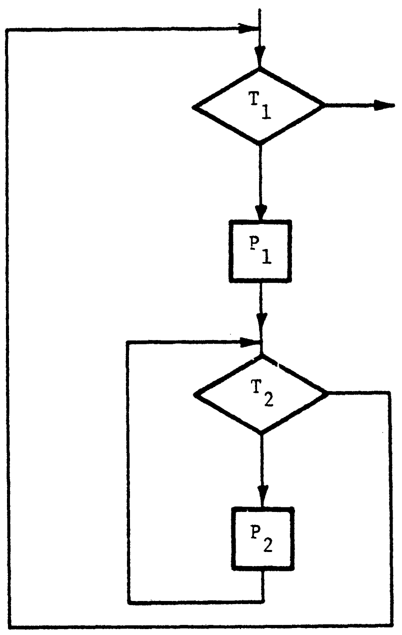


Figure 7: Nested Loop Structure

By applying the rules of Algorithm 3, we can find the two performance values for flowchart F in Figure 7 as:

$$N_{C,F} = 4 + 7\, X_{(n)} + 3 \sum_{i=1}^{X_{(n)}} X_{(n)}$$

and

$$N_{M,F} = 2 + (5\, X + 3 \sum_{i=1}^{X} X)_{(n)}$$

Figure 8 shows the curves for the ratio of the expected values, $(E[N_{C,F}]/E[N_{M,F}])$, against p for various values of n.

Once again it can be seen that the relative performance of the MIAP increases with n. Similar results for a greater number of nested loop structures may be found by using Algorithm 3. However, computation of expected values in such cases is not tractable.

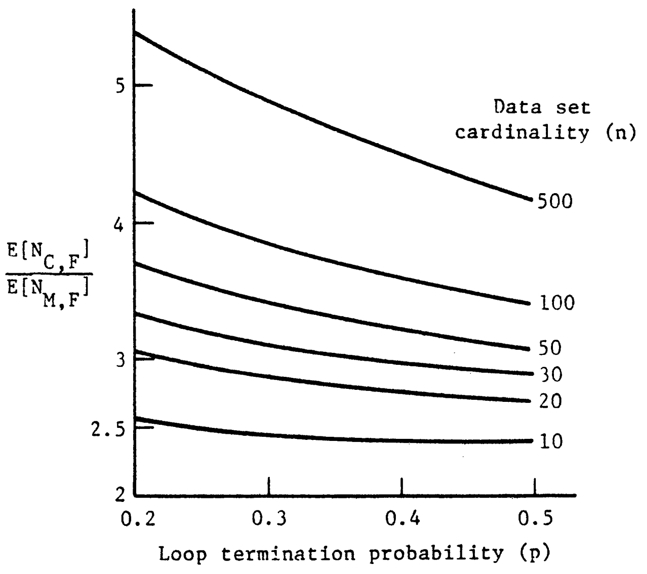


Figure 8: Expected Performance Ratio for Nested Loop Programs

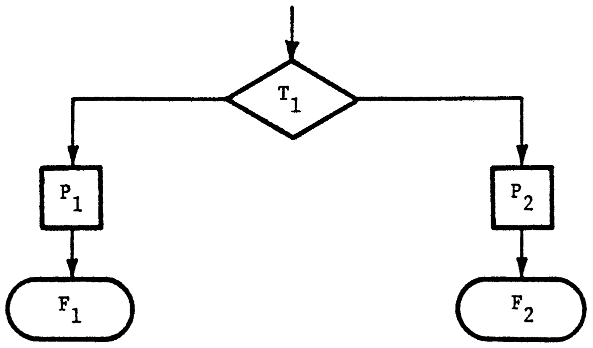
Loop-free Programs

The loop-free constructs considered here consist of nested conditional constructs called T-charts. We define this family of flowcharts as follows.

Definition 4: A T-chart is a flowchart that can be constructed using the following rules recursively:

(i) The null structure is a T-chart.

(ii) If F_1 and F_2 are T-charts, then the following structure is also a T-chart:



We are interested mainly in two subclasses of T-charts, namely, the full and bare T-charts. Full T-charts are those which contain the maximum number of conditional constructs at each level. Bare T-charts are those which contain only a single conditional at each level. The height of a T-chart is the maximum depth of nesting of conditional constructs. Figures 9 and 10 show sample full and bare T-charts of height 2.

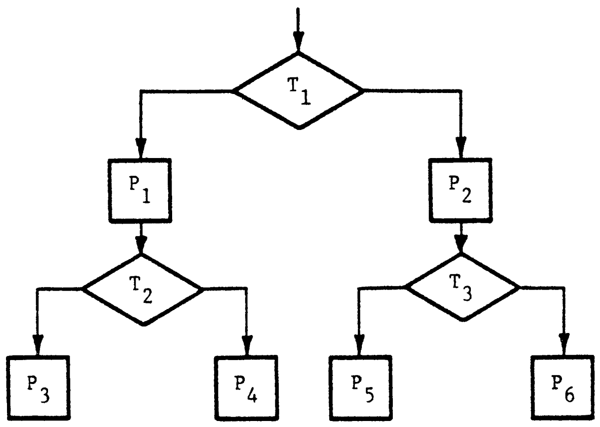


Figure 9: Full T-Chart of Height 2

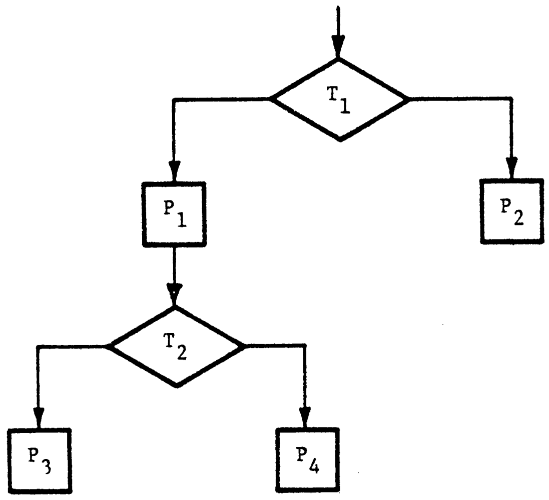


Figure 10: Bare T-Chart of Height 2

For a full T-chart F, of height h, the performance values are:

$$N_{C,F} = 6(2^h - 1)$$

$$N_{M,F} = (N_{P,F})_{(n)} = (3h)_{(n)} = 3h$$

For a bare T-chart B, the performance for the CAP is:

$$N_{C,B} = 6h$$

The performance of the MIAP for a bare T-chart B is non-deterministic and lies between the following bounds

$$3 \leq N_{M,B} \leq 3h$$

Figure 11 shows the curves for the performances for bare and full T-charts for different values of height h. For the performance value $N_{M,B}$, the bounds $N_{P,B,MAX}$ and $N_{P,B,MIN}$ are shown.

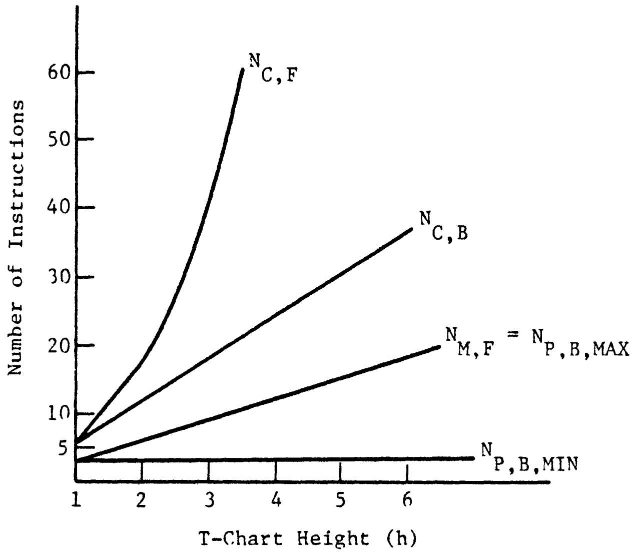


Figure 11: Performance for Bare and Full T-Charts

xed Flowchart Structures

All the programs considered so far contain ther loops or conditional branches but not both. particular, the loop programs are made up of op constructs that contain only a single pro- ssing operation in each loop body. T-charts ɜ built up of conditional structures that con- in single processing operations in each branch.

The performance of the MIAP is dependent on ɜ complexity of the flowchart. In general, the rformance of the MIAP relative to that of the ? improves substantially if a greater number of ɜmentary structures are composed or nested. To lustrate this point, we give results for some :iations of the basic families of flowcharts ısidered previously. In each case, a single P ɜration is replaced by composite subflowchart.

(i) Consider a sequence of 20 loops where :h loop body contains a full T-chart of height Assume that the parameter p for each loop is ; and the cardinality of the data set is 500. · this flowchart,

$$E[N_C]/E[N_M] = 16 .$$

(ii) Consider a full T-chart of height 5 ːre each branch of conditional statements con- .ns a single loop construct. Assume parameter ıf each loop is 0.2 and the cardinality of data : is 50. The expected performance ratio is

$$E[N_C]/E[N_M] = 24 .$$

(iii) Consider the same problem as in (ii) ept that a nested pair of loops is substituted each branch. In this case,

$$E[N_C]/E[N_M] > 65 .$$

This is a conservative lower bound for the ratio. In fact, if we approximate the distribution of the performance value of a single PE, (Np), by a Gaussian distribution (which is reasonable for small values of p and large values of h), the expected ratio is

$$E[N_C]/E[N_M] \simeq 106 .$$

Conclusions

This paper has introduced the concept of multiple instruction associative processing that removes the restrictions of SIMD operation from conventional associative processing techniques. The concept is realizable through a highly modular and extendible multiprocessor computer architecture. A study of the relative execution speeds of the conventional and multiple instruction associative processor shows that significant improvements can be expected depending on the structure of the associative process under consideration.

An implementation of the MIAP using Texas Instruments TI990/4 microcomputer modules is in progress at the University of Michigan. The system is supported by several software facilities, including a high level language oriented towards associative processing problems, a cross compiler for the language, and an operating system providing run-time support for the language.

References

[1] A.J. Evensen, and J.L. Troy, "Introduction to the Architecture of a 288-Element PEPE," Proc. 1973 Sagamore Computer Conference on Parallel Processing, (1973), pp. 162-169.

[2] L. Gilman, and A. Rose, "APL—An Interactive Approach," J. Wiley & Sons, (1974), 384 pp.

[3] S.R. Kosaraju, "Analysis of Structured Programs," Jour. of Comp. Sys. Sciences, (9, 1974), pp. 232-255.

[4] K. Batcher, "STARAN Parallel Processor System Hardware," AFIPS 1974 NCC, (43, 1974), pp. 405-410.

[5] H.J. Siegel, "Masking Scheme for Determining the Active/Inactive Status of SIMD Machine Processors, Purdue University, TR-EE-77-25, (May, 1977), 40 pp.

[6] J.R. Dingeldine, "Parallel Fortran (PFOR) PEPE Assembly Language (PAL) User's Manual," Systems Development Corp., SDC Rep. No. TM-HU-046/400/01, (Aug, 1976), 415 pp.

[7] M.J. Flynn, "Some Computer Organizations and Their Effectiveness," IEEE Trans. Comp, (Sept, 1972), pp. 948-960.

[8] R. Kober, et al, "SMS 101 - A Structured Multimicroprocessor System with Deadlock Free Operation Scheme," Euromicro Newsletter, (2, 1976), pp. 56-64.

[9] J.D. Mulla, "Associative Processing and its Extensions," (Ph.D. Thesis), The Univ. of Mich., (1978), (also SEL Tech. Rep. 119, Systems Engin. Lab., Dept. of Elec. & Comp. Engin., Univ. of Mich.).

[10] H.A. David, "Order Statistics," J. Wiley & Sons, (1970).