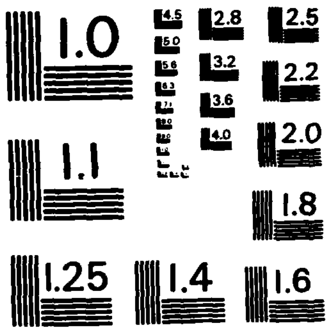
AD-A160 206 CONFLICT SENSITIVITY OF ALGORITHMS PART 1 A CRAY X-MP STUDY(U) MICHIGAN UNIV ANN ARBOR SUPERCOMPUTER ALGORITHM RESEARCH LAB.. D A CALAHAN 15 MAR 85 SARL-7 1/1

UNCLASSIFIED AFOSR-TR-85-0764 AFOSR-84-0096 F/G 9/2 NL

END

FILMED

DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5

Report SARL #7

Conflict Sensitivity of Algorithms

Part I: A CRAY X-MP Study

D. A. Calahan

March 15, 1985

Sponsored by
Air Force Office of Scientific Research
Los Alamos National Laboratory

Supercomputer Algorithm Research Laboratory
Department of Electrical Engineering & Computer Science

Conflict Sensitivity of Algorithms

Part I: A CRAY X-MP Study

D. A. Calahan

March 15, 1985

Sponsored by

Air Force Office of Scientific Research

under Grant 84-0096

and

Los Alamos National Laboratory

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

AD-A160 206

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY ---	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public release; distibution Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) SARL #7	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 85-0764
6a. NAME OF PERFORMING ORGANIZATION University of Michigan	6b. OFFICE SYMBOL (If applicable)
7a. NAME OF MONITORING ORGANIZATION AFOSR	
6c. ADDRESS (City, State and ZIP Code) Dept. of Elec. Eng. & Computer Science Ann Arbor, MI 48109	7b. ADDRESS (City, State and ZIP Code) Bldg. 410 Bolling AFB, D.C. 20332
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFOSR	8b. OFFICE SYMBOL (If applicable) NM
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER AFOSR-84-0096	
8c. ADDRESS (City, State and ZIP Code) Bldg. 410 Bolling AFB, D.C. 20332-6448	10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
61102F	2304	A3	

11. TITLE (Include Security Classification) Conflict Sensitivity of Algorithms Part I: A CRAY X-MP Study

12. PERSONAL AUTHOR(S)
D. A. Calahan

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Interim	FROM ____ TO ____	15 March 1985	23

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	Linear algebra, Computer memories, Supercomputers
XXX	XXXXXXXX	XXXXX	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The delay of algorithm execution due to memory conflicts in a 16-processor CRAY X-MP extension is considered. The association between memory access delays of reads and writes, and delays in the resultant algorithm execution is studied by defining an incremental algorithm delay sensitivity and relating it to simulated large-delay and random variations. It is shown that, by devision algorithms with zero incremental sensitivity, library software highly resistant to large delays may be achieved in a ~~many-processor~~ X-MP.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☒ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL John P. Thomas Jr., Capt. USAF	22b. TELEPHONE NUMBER (Include Area Code) (202)767-5026
22c. OFFICE SYMBOL NM	

DD FORM 1473, 83 APR   EDITION OF 1 JAN 73 IS OBSOLETE.

Unclassified
SECURITY CLASSIFICATION OF THIS PA

Abstract

The delay of algorithm execution due to memory conflicts in a 16-processor CRAY X-MP extension is considered. The association between memory access delays of reads and writes, and delays in the resultant algorithm execution is studied by defining an incremental algorithm delay sensitivity and relating it to simulated large-delay and random variations. It is shown that, by devising algorithms with zero incremental sensitivity, library software highly resistant to large access delays may be achieved in a many-processor X-MP.

Acknowledgement

The assistance of Ken Elliott III in development of a (proprietary) CRAY X-MP simulator is acknowledged.

TABLE OF CONTENTS

I. ALGORITHM CONFLICT SENSITIVITY.

A. INTRODUCTION

In a companion report [1], the effect of different memory conflict resolution protocols on delays of memory accesses was studied. These access delays produce a delay in algorithm execution or algorithm delay. However, the relationship between these two delays has not been investigated in the literature, in part because the hardware cannot measure access delay in general, and partly because a memory access simulator is far easier to develop than the full instruction-level timing simulator necessary to measure algorithm delay.

A priori, it may not seem worth correlating access and algorithm delays. An architect may be comfortable with the assumption that there is a general correspondence between the two. Indeed, it will be one of the purposes of this research to determine a rule-of-thumb relationship by examining some typical scientific application codes; the question of whether the code itself is responsible for enhancing access delays will therefore be answered. Algorithmically, however, it will be shown that codes can be designed to exploit local conflict-free memory and achieve virtual independence of main memory access delays. These will be termed conflict-resistant algorithms. Their study may have short term value in the immediate task of developing library codes for the CRAY family of multiprocessors, and long term value in establishing an additional application of cache and local memory in MP supercomputer architectures.

The experimental vehicle for this largely empirical study is a CRAY X-MP simulator. This instruction-level simulator produces numerical and timing results; the latter are accurate to within .2% for typical codes executing on a uniprocessor X-MP-2 without interprocessor conflicts. The conflict mechanism of the X-MP-2 is also simulated and has been found to be exact for large kernels of read and write instructions only. The conflict protocol of the X-MP-2 has been adopted in extension to 16 processors (even though the X-MP-4 is known to have some variations), except that processors are paired to achieve adequate memory bandwidth for buffer fetches. More validation is given in an appendix of [1].

B. MEMORY ACCESS VERSUS ALGORITHM DELAYS

1. Critical Path

The X-MP operates by a system of register and functional unit reservations. Instructions begin execution either (1) when reservations expire on the resources they require, or (2) when elements of a vector operand become available from a functional unit ("chaining"). When reads or writes are delayed by conflicts, the associated register reservations are held and chains are delayed.

A particular instruction issue and/or execution may or may not influence total execution time. For example, address formation is often masked by floating point computation in a vectorized code. When such influence does exist, the instruction is on the critical path of execution.

Determining which instructions are on this critical path is difficult even with a simulator. For example, a hold on instruction issue does not guarantee that the instruction creating the hold is on the critical path; both of the instructions may be off the critical path and their issue time superfluous to total algorithm timing. The critical path is first a global issue.

This critical path can change as access delays lengthens; for example, a formerly masked read or write may enter the critical path when its execution is delayed and no longer masked. An instruction awaiting two reads, as V3 in the vector sequence (in CRAY assembly language)

```
V0    ,A0,1    (vector read)
V1    ,A0,1    (vector read)
V2    S1*V1    (vector multiply)
V3    V2+FV0   (vector add)
```

could be delayed by conflicts on either V0 or V1; thus the algorithm delay is a function of delays on V0 and V1 reads, creating a "worst-case" risk situation.

In contrast, a "best-case" condition occurs when an instruction is awaiting availability of alternate identical resources. In the X-MP, the read port (of two ports) is chosen during execution. If both ports are busied with delayed reads, the first available one is used.

2. A Sensitivity Measure

In spite of the above threats to a well-behaved cause-effect relationship between memory access and algorithm delays, it is nonetheless possible to develop a meaningful sensitivity measure relating the two. Only <u>vector</u> access delays will be considered in the following discussion.

Define

T - algorithm execution time

T_{m_i} - time memory is busied during access of the ith vector in the critical path

ΔT_{m_i} - change in T_{m_i}

ΔT_i - change in T due to ΔT_{m_i}

ΔT - total change in T due to delays in vectors.

where T_{m_i} is VL+3 without conflicts, where VL is the vector length. Then if only the ith instruction is delayed

$$\Delta T_i = \Delta T_{m_i} \qquad (1)$$

and the algorithm sensitivity to a delay in the ith access is

$$S_i \triangleq \frac{\text{fractional change in } T}{\text{fractional change in } T_{m_i}} \qquad (2)$$

$$= (\Delta T_i/T)/(\Delta T_{m_i}/T_{m_i}) \qquad (3)$$

$$= T_{m_i}/T \qquad (4)$$

Thus, this normalized sensitivity is not dependent on the fraction of time a read or write is in the critical path, but, once in the critical path, on its total vector length.

If m vectors are in the critical path, and if the effects on T of each vector access delay are independent of other delays (to be tested by simulation), then

$$\Delta T = \sum_{i=1}^{m} \Delta T_{m_i} \qquad (5)$$

If all vectors are delayed by a uniform fraction of their lengths so that $\Delta T_{m_i}/T_{m_i} = D$, a constant, then define the global sensitivity

$$S_u \triangleq \frac{\text{fractional change in } T}{D} \qquad (6)$$

$$= (\Delta T/T)/D \qquad (7)$$

$$= \left(\sum_{i=1}^{m} T_{m_i}\right)/T \qquad (8)$$

$$= \sum_{i=1}^{m} S_i \qquad (9)$$

In the limiting case, then, if every vector access has one clock in the critical path in the conflict-free case, the total sensitivity would be proportional to the sum of all the read/write vector lengths!

II. SIMULATED SENSITIVITY STUDIES

A. LARGE-DELAY SENSITIVITY

The relationship of Eq. (9) merely represents the additive nature of independent delays. The practical issues are the effects of (a) large and (b) random access delays on the critical path or, equivalently, on the algorithm delay. These two effects will be measured separately by simulation.

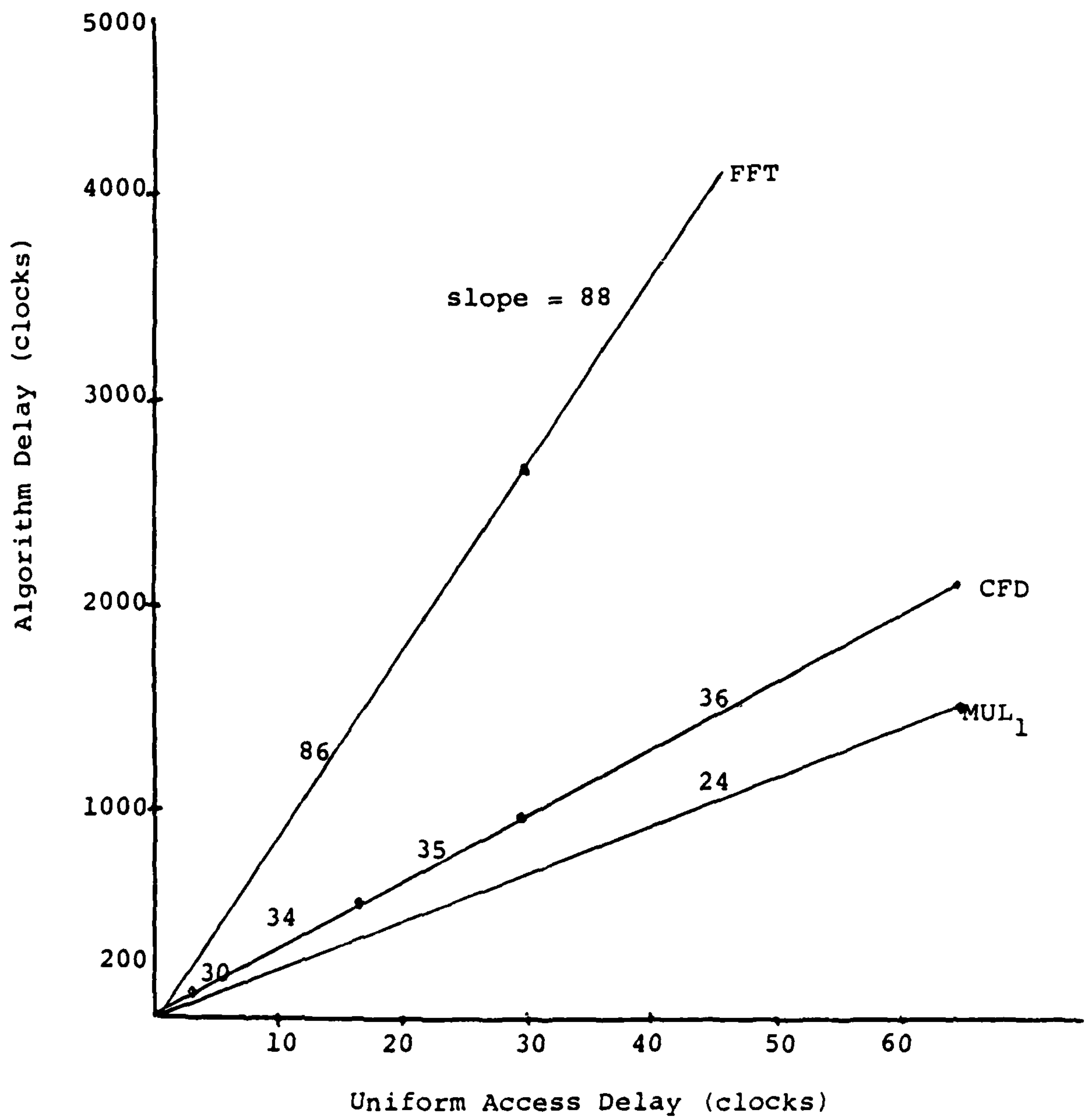
The S_u defined in Eq. (6) can be measured, irrespective of whether Eq. (9) applies as a result of independence. By disabling the X-MP conflict resolution protocol in the simulator and instead artifically delaying all accesses a uniform fraction D of their vector lengths, the delay (D_u) in algorithm execution can be measured as a function of D. This will test the dependence of the critical path on large but uniform delays.

The result (Figure 1) shows that, for the three test codes, the slope S_u remains nearly constant for delays of up to 100% of the vector length. Thus, under the assumption of uniform delays, the hazards to the critical path disruption are insignificant for access delays far greater than likely to be encountered in practice. The incremental sensitivities S_u measured at $D = 0$ are given in Table 1 for a large number of cases.

It should be noted that Eq. (9) has been verified by inspection of clock-level timing of MUL2 and CFD executions.

Fig. 1. Algorithm delay due to uniform access delay VL = 64 for all vectors

Numbers are slopes; dots at slope changes



Thus, algorithm delay seems likely to be independent of access delays for even large uniform delays.

B. EFFECTS OF RANDOMNESS

With the conflict protocol enabled, the delay in algorithm execution (D_{al}) was measured for all processors involved in a simulation, and the delays averaged ($\bar{D}_{al}$). The delays of all accesses were also recorded and averaged ($\bar{D}_{ac}$). These delays were normalized by dividing by the total algorithm execution time and by VL (=64 for all codes), respectively; this yielded $\tilde{D}_{al}$ and $\tilde{D}_{ac}$. The sensitivity

$$S_{al} \triangleq \tilde{D}_{al}/\tilde{D}_{ac}$$

is then the measure of the random, large-deviation sensitivity encountered in practice.

Table 1 indicates that S_u and S_{al} are sufficiently different that the critical path must be moderately altered in some codes. Since large uniform deviations have been shown to have nominal effects on sensitivity, one is left to conclude that it is the randomness which disrupts the critical path. This is consistent with the previous discussion of how the critical path is altered, e.g., by masking and by best-case and worst-case events.

It appears that the sensitivity to a single delayed access should be less than unity; the provision for late chaining avoids the prospect of a delayed access causing a missed chain-slot time, as in the CRAY-1. However, it is unclear whether S_{al} can be greater than unity. Nonetheless, $S_{al} < 1$ for all measured sensitivities (Table 2), with the largest being .939.

		Utilizations		Delays × 100 %		Sensitivities		
Code	Banks	$\bar{U}_m$	$\bar{U}_b$	$\tilde{D}_m$	$\tilde{D}_a$	S_{al}	S_u	$S_{al}/\bar{U}_m$
FFT	256	.653	.965	7.7	3.8	.493	.417	.755
	128	.629	.998	16.8	8.1	.482	.417	.766
CFD	256	.682	.986	6.0	2.3	.383	.336	.561
	128	.668	.999	12.3	4.3	.349	.336	.522
MUL_1	256	.690	.832	5.1	2.5	.490	.464	.710
	128	.664	.921	16.8	6.5	.387	.464	.592
MUL_2	256	1.51	.998	5.9	5.1	.864	.602	.572
	128	1.31	.998	25.4	21.9	.862	.602	.658
MUL_3	256	.932	.966	2.6	.4	.150	.147	.160
	128	.925	.989	6.5	1.1	.161	.147	.174

Table 1. Summary of simulation results for 16 processors. Sixteen samples were used to determine averages.

C. AN EMPIRICAL RELATIONSHIP

Aside from confirming intuition, Table 1 appears to show a relationship between sensitivity of Fortran codes and their memory utilization.

Define

$$\bar{U}_m = \frac{\text{average number of memory reads ad writes per processor}}{\text{algorithm execution time per processor}}$$

Then the ratio $S_{al}/\bar{U}_m$ is shown in Table 1 to range over a rather restricted set of values (.552 to .766), across different codes and in the presence of access delays $\tilde{D}_{ac}$ which vary over a 5:1 range (.051 to .254) as the number of banks is varied. An approximate sensitivity determined from

$$S_{al} = .65\ \bar{U}_m$$

would be within 18% for all cases. The range of this approximation is limited however, if S_{al} is bounded by unity. The relationship between S_{al} and $\bar{U}_m$ is felt to be indirect; possibly it is due to the number of ports rather than $\bar{U}_m$ which supply vector operands to the matrix multiply inner loop. If one of these ports is delayed, a "worst-case" delay is imposed on the loop and S_{al} increases.

D. CONCLUSIONS

In this section, two results stand out.

(a) The randomness rather than the size of the access delays have the greatest effects on the critical path.

(b) If the memory utilization per processor $\bar{U}_m$ is known, the algorithm sensitivity to access delays may be estimated from the rule-of-thumb

---

*MUL3 is a specially coded CAL routine (see Section III).

$$S_{a1} = .65\ \bar{U}_m$$

for conventional Fortran vector codes. This puts the simpler memory access simulation performed by computer architects on firm grounds, in so far as their ability to predict algorithm delay.

The above conclusions are based on three Fortran codes; this must be acknowledged as a small sample, in spite of the diversity of their access patterns. Also, the vector length was constant at 64; the above formula could also depend on VL, since for a given $\bar{U}_m$, the critical path would likely be more disrupted by short vectors.

III. ALGORITHM DELAY RESULTS

Although the above study of the two components of $\tilde{D}_{al}$ (= $\tilde{D}_{ac}$ S_{al}) may give insight, the algorithm delay $\tilde{D}_{al}$ itself is ultimately of interest. These are depicted in Figures 2 and 3 and given in Table 2.

Figure 2 gives $\tilde{D}_{al}$ when R_{bp} = 16, the most likely situation. The FFT, MUL, and CFD codes have nearly the same delay between 2% and 3% from 1 to 16 processors, corresponding to their simular $\overline{U}_m$. MUL_2, with high access delay and large S_{al}, has nearly a 5% delay. Their are seemingly no surprises here.

When the number of banks is halved, Figures 3 indicates that S_{al} of the high access MUL_2 code increases by the greatest ratio (4.1:1). Even the small differences been curves in Figure 2 are magnified in Figure 3. The implication is that, since S_{al} remains relatively constant (Table 2) as R_{bp} increases, R_{bp} = 16 is the smallest ratio which avoids the risk of high $\tilde{D}_{ac}$'s with common $\overline{U}_m$'s.

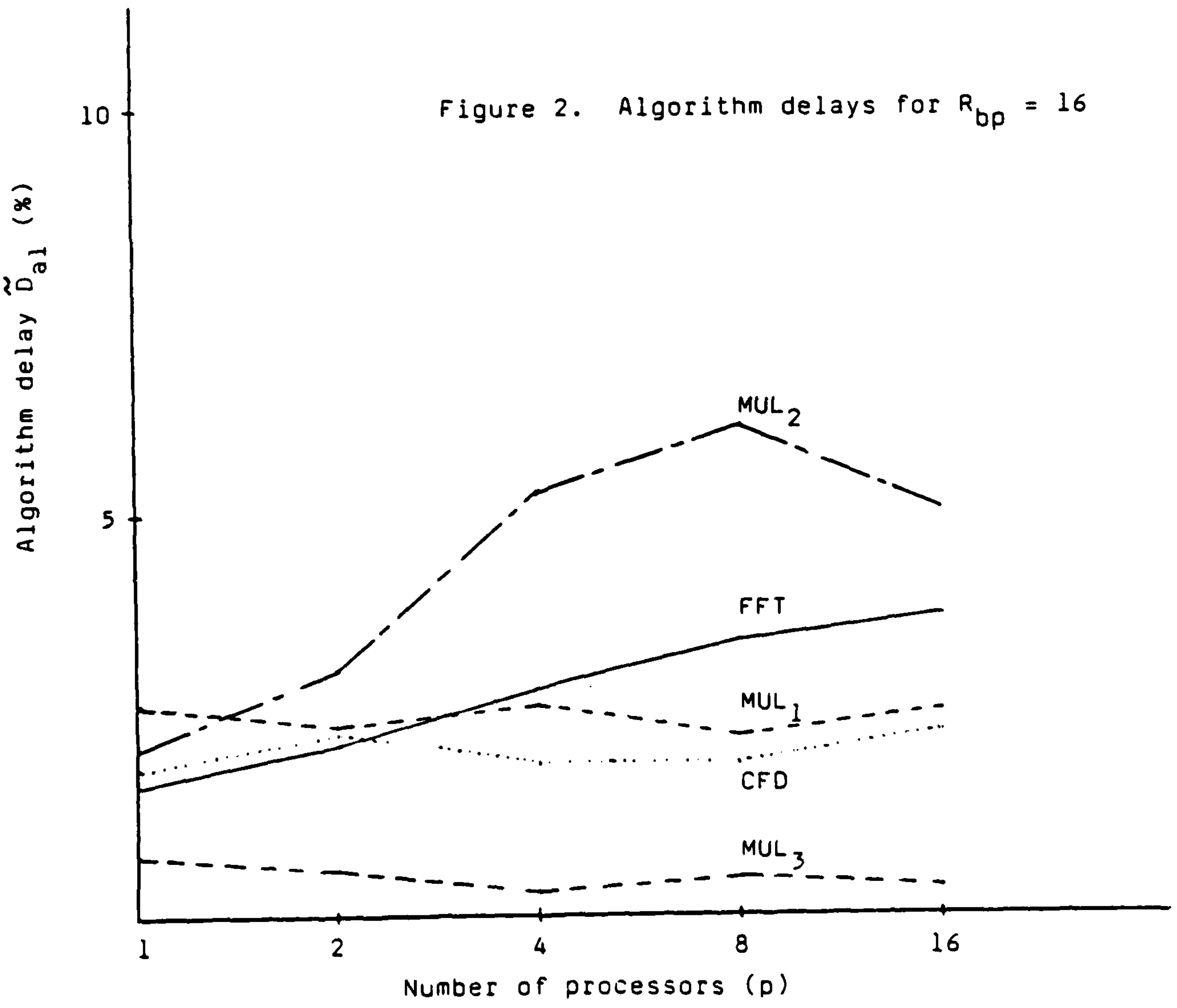


Figure 2. Algorithm delays for R_{bp} = 16
10
5
Algorithm delay $\tilde{D}_{al}$ (%)
MUL$_2$
FFT
MUL$_1$
CFD
MUL$_3$
1
2
4
8
16
Number of processors (p)

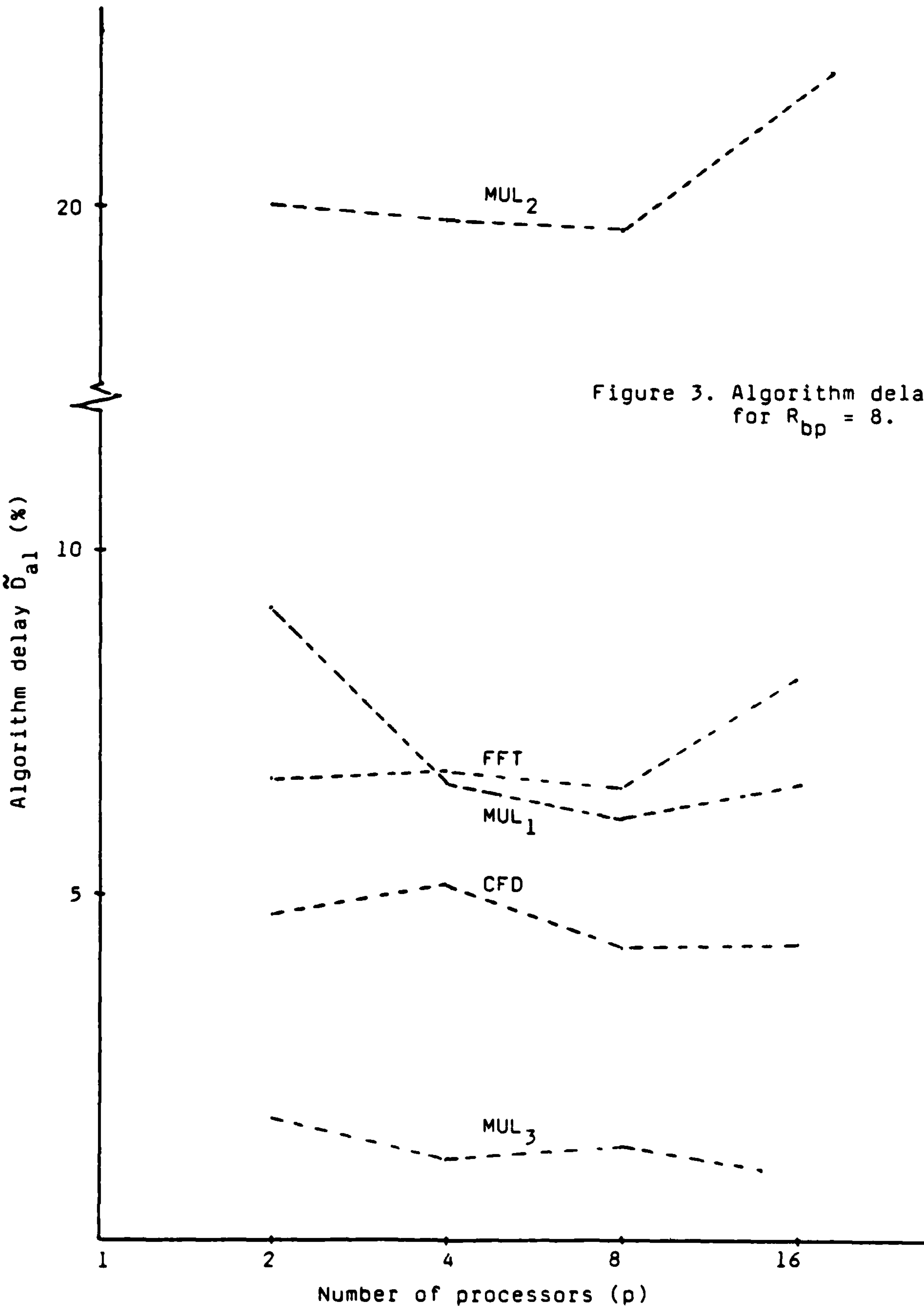


Figure 3. Algorithm delays for R_{bp} = 8.

IV. CONFLICT-RESISTANT ALGORITHMS

A. INTRODUCTION

Undoubtedly, the greatest benefit of defining an algorithm sensitivity is in algorithm design. It will be shown possible, with careful control of the data flow in each processor using assembly language (CAL), to defeat the normal relationship between D_{ac} and D_{al} and ultimately to reduce the small-delay (incremental) sensitivity to zero. Since assembly coding is a common practice for CRAY-1 and CRAY X-MP library programs, it may take a small additional effort to isolate these workhorse codes from the large delays possibly associated with many-processor architectures.

B. LOCAL MEMORY UTILIZATION

Two conditions must be met to guarantee code performance resistant to access delays.

(a) Shared memory access must be off the critical path, and

(b) Vector access must be on data in conflict-free storage.

The vector register set forms such storage on the X-MP; the former can be achieved by pre-fetching operands and post-storing results.

Prefetching is difficult to achieve for general codes, and, where possible, usually requires loop reordering and other instruction scheduling beyond commercially-available compilers. Library programs, which are often built around a small kernel, are candidates for such coding. Fortunately, CRAY-1 experience has shown that prefetching can be completely masked by floating point computation in linear algebra codes without reducing the execution rate; the vector register set is sufficiently large to act as a non-conflict buffer [2].

A matrix-vector multiply (MUL_3) has been assembly coded with these features; the resultant sensitivities are shown in Table 1. The related and S_{al}'s are a small fraction (20-25%) of those for the corresponding MUL_2 Fortran code, and considerably less than any other kernel in the table.

C. NON-UNIT STRIDE ACCESS

S_{al} of the inner loop of MUL_3 has a zero value for small delays. However, the CAL implementation that yielded the low S_{al} of MUL_3 in Tables 1 and 2 for 64 × 16 matrix-vector multiplies produced quite different results when 64 × 64 matrices were multiplied, as indicated in Table 3a. The sensitivities S_{al} increase with number of banks, although $\overline{D}_{ac}$ and $\overline{D}_{al}$ decrease individually. The orgin of the problem seems worthy of discussion.

It is a convenience in CAL coding of matrix multiplies and other linear algebra codes to implement

$$Y \leftarrow Y + MX$$

by loading the elements of X in reverse order into a vector register, and then arranging them as scalars to multply the columns of M. The related assembly code has the form

```
                  A1    64
                  VL    A1
                  V0    ,A0, -1
                         .
                         .
                         .
      inner
      loop        S1   . V0, A1
                         .
                         .
                  V3    S1*FV2
                         .
                         .
```

Banks	σ_{ac} (%)	σ_{al} (%)	s_{al}
Original code			
16	19.4	4.84	.249
32	5.32	1.64	.309
64	1.52	.842	.657
Modified Code			
16	17.6	1.47	.083
32	1.74	.391	.224
64	1.53	.271	.178

Table 3. Effect of eliminating counter-grain access in 64 × 64 multiply; p = 4.

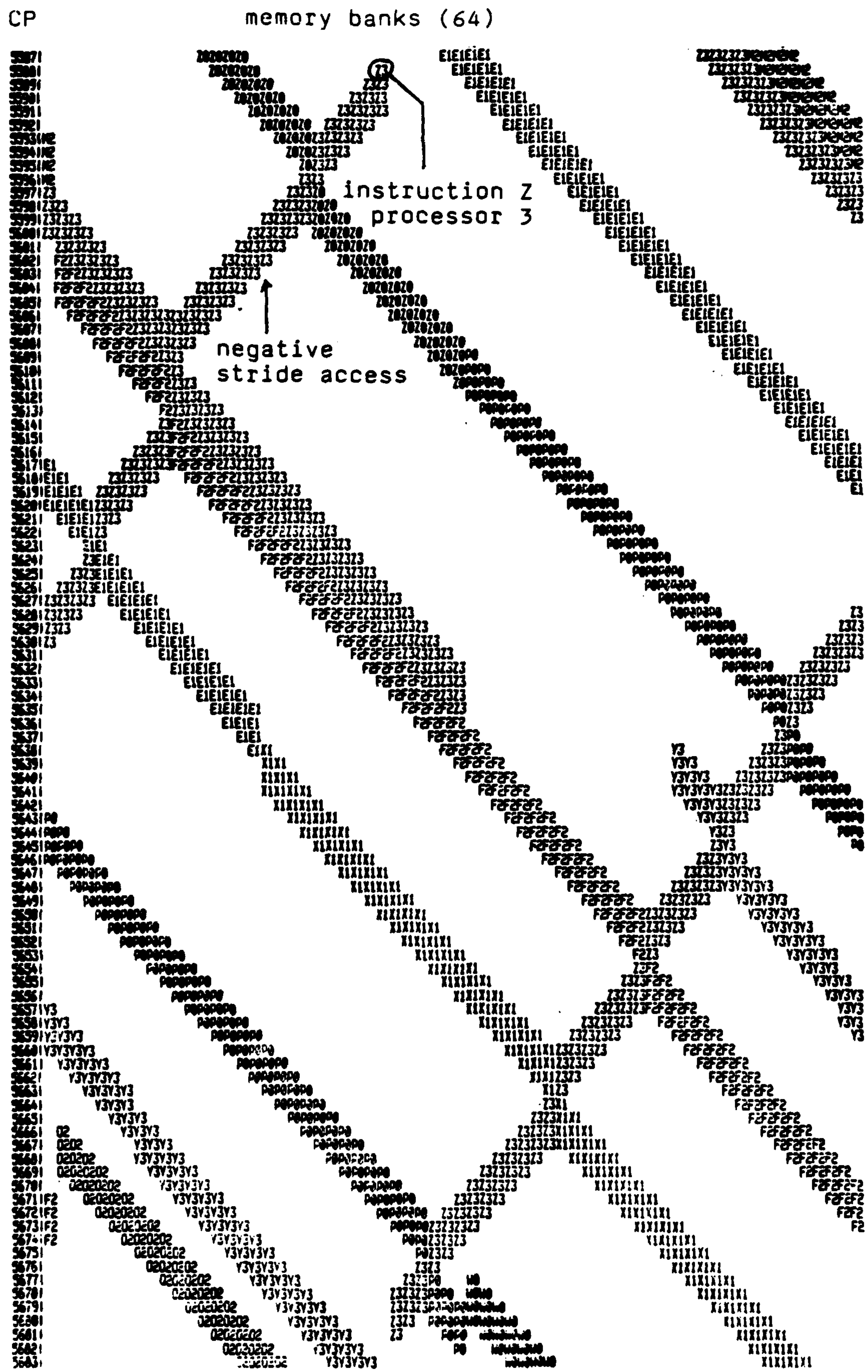


Figure 4. Effect of negative-stride access

	$R_{pb} = 8$			$R_{pb} = 16$		
1 processor	$\tilde{D}_{ac}$ (%)	$\tilde{D}_{al}$ (%)	S_{al}	$\tilde{D}_{ac}$ (%)	$\tilde{D}_{al}$ (%)	S_{al}
MUL_1				4.22	2.53	.600
MUL_2				3.75	2.19	.584
MUL_3				3.43	.76	.221
CFD				6.56	1.85	.282
FFT				4.37	1.72	.393
Average*				4.72	2.07	.464
2 processor						
MUL_1	26.5	9.25	.349	3.12	2.36	.756
MUL_2	29.6	20.0	.675	4.53	3.04	.671
MUL_3	6.87	1.81	.263	6.25	.56	.089
CFD	15.9	4.68	.294	7.19	2.21	.307
FFT	17.8	6.66	.374	4.84	2.15	.444
Average*	22.4	10.1	.423	4.92	2.44	.544
4 processor						
MUL_1	21.2	6.61	.311	6.09	2.61	.429
MUL_2	27.9	19.8	.709	5.62	5.28	.939
MUL_3	5.15	1.12	.217	1.87	.31	.166
CFD	14.3	5.15	.360	5.93	1.88	.317
FFT	15.0	6.73	.448	5.46	2.71	.496
Average*	19.6	9.57	.457	5.77	3.12	.545
8 processor						
MUL_1	16.2	6.02	.371	4.69	2.27	.484
MUL_2	25.6	19.7	.769	6.87	6.14	.893
MUL_3	7.34	1.36	.185	1.87	.49	.262
CFD	12.1	4.24	.350	5.00	1.85	.370
FFT	14.0	6.61	.472	6.56	3.48	.530
Average*	17.0	9.14	.491	5.78	3.43	.569
16 processor						
MUL_1	16.9	6.51	.385	5.00	2.51	.502
MUL_2	25.3	21.9	.865	5.93	5.09	.858
MUL_3	6.56	1.04	.158	2.65	.40	.151
CFD	12.3	4.30	.349	6.09	2.29	.376
FFT	16.9	8.16	.482	7.65	3.83	.500
Average*	17.8	10.2	.520	6.17	3.43	.559

*MUL_3 not included in averages

Table 2. Delay and sensitivity summary for two R_{bp} ratios. X-MP-2 protocol.

A delay in the load of VO may delay the first trip through the inner loop if VL is sufficiently long, since the load of S1 will not chain off the read. Worse, the read has a negative unit stride, whereas all other accesses have positive unit strides. Figure 4 shows the effects of such an access on the other highly-regular accesses. Access Z3, beginning at clock 5588, intesects and delays seven other accesses, two of them twice. It is evident that a window of accesses extending approximately -VL and +2VL clocks from the initiation of VO is potentially affected by such a counter-grain access. The access Z3 itself is delayed by 28 clocks.

When the access is replaced by a positive unit-stride access, the low sensitivity of the modified code of Table 3 is obtained. The $\tilde{D}_{al}$ is reduced to insignificant levels (.272%) for a typical R_{bp} = 16, and retains these levels (1.47%) when the number of banks is reduced by a further factor of 4!

REFERENCES

[1] Calahan, D. A., and K. E. Elliott III, "Memory Conflict Simulation of Many-Processor CRAY Architecture. Part I: A CRAY X-MP Study," Report SARL #6, Supercomputer Algorithm Research Laboratory, Department of Electrical Engineering and Computer Sciences, University of Michigan, March 1, 1985.

[2] Calahan, D. A. et al, "Sparse Matrix and Other High Performance Algorithms for the CRAY-1," Report #124, Systems Engineering Laboratory, Department of Electrical Engineering, University of Michigan, January, 1979.

APPENDIX A

EXPERIMENT DESCRIPTION

EXPERIMENTAL PARAMETERS

The codes were produced by the X-MP CFT compiler from Fortran source codes. Vector length (VL) is 64 and stride is 1 for all cases.

Distinct program and data storage was used for each of the 16 processors. Code executions were initiated at irregular intervals to further randomize accesses between processors. In general, p samples were used to produce mean values with p procesors.

Two global static measures of memory accesses were made to monitor their uniformity.

(a) Memory utilization. This is the fraction

$$\bar{U}_m = \frac{\text{Total operands and results}}{\text{Simulation time (CP's)}}$$

for the average processor; it is a measure of memory traffic for each code, and has a maximum value of 3, corresponding to the number of memory ports per processor. Table 1 shows $\bar{U}_m$ = .67 for FFT, CFD, and MUL_1.

(b) Bank utilization. Let N_b be the number of banks. There is a risk with 64-length unit-stride vectors and $N_b > 64$ that banks will not be equally utilized; this would create uncharacteristic delays in heavily-utilized banks. If $\bar{N}$ is the average number of accesses per processor across all banks, and N is the standard deviation from this average, define the bank utilization

$$\bar{U}_b = \frac{\bar{N} - N}{\bar{N}}.$$

$\overline{U}_b = 1$ indicates uniform accessing; if only 1/2 of the banks are accessed, $\overline{U}_b = 1/2$. Table 1 indicates $.832 < \overline{U}_b < .998$..

CODE DESCRIPTIONS

(a) Fluids kernel (CFD). Taken from the vectorized code of [7], this is a 32-statement single-loop Fortran kernel with an average of 3.2 64-length vector-vector operations/statement. Lack of a repetitive computational structure like FFT and MUL should make the access pattern the most random. Six buffer fetches occur in one kernel execution.

(b) FFT kernel (FFT). This code determines multiple 8-point complex-complex FFT's. Five buffer fetches occur in one kernel execution.

(c) Matrix-vector multiply kernel (MUL_1, MUL_2, MUL_3). The inner-loop of MUL_1 and MUL_2 has two vector reads and one write per execution. MUL_1 maintains low memory utilization ($U_m = .69$) with VL = 64 by multiplying 4 small (64 x 3) matrices in one kernel execution step; MUL_2 uses the same code with 512×2 matrices, which successively exercises the inner-loop 512/64 = 8 times, and achieves $\overline{U}_m = 1.58$, a value more characteristic of a large Fortran-coded matrix multiply on the X-MP. No buffer fetches occur in consecutive executions of the kernel. The inner loop of MUL_3 has one pre-fetched vector read per inner loop execution.

END

FILMED

11-85

DTIC