LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

Structure Referencing in the Tagged Token Dataflow Architecture

An Initial Investigation

Computation Structures Group Memo #268
1 October 1986

Andrew A. Chien

This report describes research done at the Laboratory for Computer Science of the Massachusetts Institute of Technology. Funding for this project is provided in part by the Advanced Research Projects Agency of the Department of Defense under the Office of Naval Research contract N00014-75-C-0661.

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Table of Contents

1 Introduction

In a high performance computer system, management of the structure storage can have significant impact on the program execution speed. For example, in a CRAY-XMP, the mapping of structures to the interleaved memory can effect the number of bank conflicts that a processor experiences. These bank conflicts can significantly reduce the effective memory bandwidth as well increase the memory latency to certain requests. Such conflicts arise when a single processor's memory requests conflict or when different processors' requests conflict.

In all multiprocessors, this problem exists in some form. Some common solutions are to use many interleaves, to use simple caches for "read only" information, or to use "snoopy" caches for everything. Use of many interleaves does not take advantage of the locality of the program (temporal or spatial), but does avoid serious contention problems. This can be viewed as a baseline solution to the problem. The "read only" caching schemes make it difficult to maintain high enough hit rates to keep the processor pipelines busy. Snoopy cache schemes do not extend easily to non-bus based systems. Thus the problem of taking advantage of locality while avoiding contention remains unsolved.

The fundamental problem is that one would like to localize the structures to that we can take advantage of spatial locality in the program. However, any such localization of structures leads to increased "contention" or demand for the memory or memories in which structure has been localized. This contention not only leads to increased latency for memory requests to the popular module, but may decrease the overall throughput of the network -- slowing the entire machine!

Recent discussion in the Computation Structures Group has raised questions about the "mapping" of structures to the machines. Varying degrees of localization have been proposed and very little experimental evidence exists to compare them. In this document, we study structure referencing patterns in the Tagged Token Dataflow Architecture. By doing so, we are able to discern some basic features of these patterns. These basic features are not likely to change except as a result of a major change in the structure referencing or work decomposition paradigms. These basic features will allow us to make some strong statements about what kinds of allocation schemes are likely to work, and how much locality we need to get in order to make an increase in contention worthwhile.

2 A Spectrum of Structure Allocation Schemes

Two major structure mapping policies are compared in the context of our experimental results. they are at opposite ends of the spectrum with respect to localization. We shall refer to any scheme which places structures (the smallest storage units allocatable by the manager) on single memory units as "vertical". The "vertical" scheme completely localizes a structure. In contrast, a "horizontal" allocation scheme is one in which each structure is spread over many (all, if possible) of the memory modules. There are many variations and hybrids of these schemes, but they only confuse the issues. We will refer to the hybrid schemes as "diagonal". The "horizontal" scheme does just the opposite. The important tradeoff in these schemes is locality (and a concomitant decrease in communication requirements) versus an increase in contention. It is interesting to note that because many small scale multiprocessors (2-20 processors) are bus based, one usually sees purely horizontal schemes (interleaved memories), because it minimizes the contention, but causes no decrease in locality. Everything is on the same bus! However, as we move to systems of more processors, the bus bandwidth becomes saturated and such solutions are infeasible.

3 Description of the Study

Because the temporal pattern of structure usage is strongly dependent on architectural details such as execution order, network speed, manager policy, etc. we felt that it would be difficult to get meaningful temporal behavior results on any of the simulation/emulation vehicles which have been constructed. Therefore, we chose to use the Graph Interpreter for the Tagged Token Architecture to study the (independent of time) structure of the cumulative data references generated by some programs. The statistics we chose to collect are the number of references per data structure in the program and the number of different contexts that refer to that structure.

3.1 Motivation for the Study

Two simple statistics over a variety of programs, and problem sizes allow us to make some interesting assertions about the eventual pattern of data structure references in our machine. The major characteristics of this pattern should not change, as they are only a function of the application program, and the basic machine level view of structures. For example, if most of the structures are only referred to by one or two contexts, then we would suspect that we could do a very good job of enhancing locality. Furthermore, this enhancement need not greatly increase contention or restrict the manager's work decomposition policy. If most of the structures are referred to by many contexts, then we might suspect that localization may result in greatly increased contention, while not enhancing locality a great deal. These are the types of gross conclusions we hope to reach from this study.

4 Results of the Study

The study can be divided into three basic investigations: "Hotness" of structures, Significance of "Hotness", and "Popularity" of structures. Each of these studies are described below. Each study consisted of a set of runs on various problem sizes for the Simple Code (a synthetic program for hydrodynamic simulations). The results are presented below. They were taken on GITA and pretty fairly represent the behavior of the Simple code, and our current architecture with the major exception of procedure call and return mechanism. Our new procedure call and return mechanism does not make use of I-structures, and would not result in any I-structure traffic. The procedure call mechanism modeled uses I-structures for both arguments and results. Some attempt was made to note the distortions due to this difference, however, the reader may draw his own conclusions.

4.1 Hotness of Data Structures

The "Hotness" study was undertaken in response to widespread speculation that I-structures encouraged a programming style which accessed data values only a few times before creating a new copy of the structure. We define the "hotness" of a data structure as the ratio of the number of references to it (memory reads) to the size of the structure. This measure will tell us whether or not the speculation was correct. Further, the distribution of the hotness of data structures will likely give us some insight as to what kinds of allocation schemes will be effective.

Simple, 1 iteration, 5 by 5

Hotness	# ofStructures	Number of each Size	% of tot. traffic
>75	2	2-5	7.47
70-75	0		
60-69	0		
50-59	0		
40-49	2	2-7	9.33
30-39	2	2-7	2.30
25-29	0		
20-24	0		
15-19	5	4-6,7,1-2	4.74
10-14	10	9-7,1-40	12.72
5-9	19	19-5,6,7	8.55
0-4	1069	474 invokes => 948 A-R Structs	54.89
		75% of the non-AR structures are "read-once"	

Detailed breakdown

Hotness	# ofStructures	Number of each Size	% of tot. traffic
5	7		2.36
4	10		2.69
3	15		3.13
2	23		3.13
1	987		44.40
0(<1)	34		1.54

Average Reads per structure:	9.07
Variance:	29.7
Standard Deviation:	882.9

Simple, 1 iteration, 10 by 10

Hotness	# ofStructures	Number of each Size	% of tot. traffic
>500	2	2-5	10.58
400-499	0		
300-399	0		
200-299	0		
150-199	1	1-12	3.83
100-149	3	3-12	10.01
50-99	3	2-11,1-40	8.90
25-49	14	8-10,11,12,6-5	8.16
20-24	2	2-11	1.09
15-19	30	30-11,12	11.90
10-14	10	10-11,12	2.83
5-9	31	31-11,12	4.41
0-4	3264	1520 invokes = >3040 AR Structs	38.28
		70% of non-AR structures are "read-once"	

Detailed breakdown

Hotness	# ofStructures	Number of each Size	% of tot. traffic
5	9		1.14
4	22		2.33
3	37		2.85
2	29		1.71
1	3113		30.29
0(<1)	63		1.10

Average Reads per structure:	14.5
Variance:	11296
Standard Deviation:	106.3

Simple, 1 iteration, 20 by 20

Hotness	# ofStructures	Number of each Size	% of tot. traffic
>2000	2	5	11.79
1000-1999	0		
500-999	0		
400-499	1	22	4.19
300-399	4	3-22,1-40	17.55
200-299	0		
150-199	2	1-22,1-5	0.88
100-149	7	3-21,4-5	4.82
50-99	9	9-21,22	6.37
25-49	4	4-21,22	1.39
20-24	18	18-22	4.13
15-19	59	59-21,22	10.56
10-14	8	8-22	0.998
5-9	62	62-21,22	4.76
0-4	11884	5709 Invokes => 11418 AR Structs	32.56
		72.6% of non-AR structures are "read-once"	

Detailed breakdown

Hotness	# ofStructures	Number of each Size	% of tot. traffic
5	26		1.33
4	36		1.55
3	102		3.47
2	24		0.595
1	11573		25.84
0	318		1.09

Average Reads per structure:	17.98
Variance:	74393
Standard Deviation:	272

Simple, 1 iteration, 40 by 40

Hotness	# ofStructures	Number of each Size	% of tot. traffic
>10,000	2	2-5	12.29
1000-1999	1	1-40	7.53
500-999	10	4-42,6-5	16.92
400-499	0		
300-399	0		
200-299	3	3-41	3.74
150-199	4	4-41,42	3.8
100-149	3	3-41	2.96
50-99	3	3-41,42	0.95
25-49	44	44-41,42	5.49
20-24	41	41-41,42	4.96
15-19	71	70-42,1-2	6.06
10-14	8	8-42	0.48
5-9	151	151-41,42	4.78
0-4	45908	22489 Invokes = >44978 AR Structs	31.03
		72.5% of non-AR structures are "read once"	

Detailed breakdown

Hotness	# ofStructures	Number of each Size	% of tot. traffic
4	76		1.58
3	222		3.66
2	44		0.535
1	45293	This is 98% of all structs!	24.15
0(<1)	273		1.10

Average Reads per structure:	21.62
Variance:	201600
Standard Deviation:	449

Analysis of Hotness Study

Several interesting observations can be made about the experimental results presented above. For instance, it is clear that for the Simple code, the number of argument result structure is vastly larger than the number of user defined structures. This ratio is well over 10 : 1 in the Simple 40x40 run. When the new procedure call schema is implemented in the compiler, we will expect to see a dramatic improvement in the number of I-structures allocated. Excluding AR structures, it seems to be true that the majority of I-structures are in fact "read once". Discounting the argument result structures, we still find that across these four runs, more than 70% of the I-structures allocated have a hotness of less than 2. However, these "read once" structures only account for a very small percentage of the overall traffic (probably less than 10%). Thus it is very important to examine the behavior of "hot" structures.

Another important point is the fact that the hotness of structures varies greatly. We found structures with hotness ratings of 0 all the way up over 10,000. Now, it is true that some of this contention was due to the way the code was written. The hottest structures were constant tables which could have been duplicated if the code were written differently or the compiler could detect them. This could reduce the contention. However, we observed hotness ratings of over 400 on some parts of the 2-D finite element arrays. This means that some structures are very likely sources of contention. The range of hotness ratings we encountered (as well as its mean and variance as we increased the problem size) gives us clear indication that certain kinds of allocation policies will not work well. For example, any scheme which works solely on the basis of structure storage available and the size of the structure allocated is not going to do a very good job of distributing the load to the I-structure controllers. The large variance of the amount of traffic headed for a given structure make it difficult for any such scheme to do well. Furthermore, the large variance makes it unlikely that any purely random scheme will do well.

4.2 Significance of Hotness

The primary statistic we're interested in is how much traffic is each structure going to experience. This was in order to determine if the hotness of structures will be an issue in our dataflow machine. For example, if an individual structure never represented more than .1% of the overall traffic, then we might conclude that increased contention and hence "hotness" is not an issue. This can be a crucial factor in how sophisticated the compilation and dynamic allocation needs to or can be. For example, if all structures had only a small amount of traffic destined for them, then perhaps simple load balancing techniques at run time would be sufficient. Conversely, if each structure could correspond to a very large or small amount of traffic, we might require some sophisticated compiler analysis in order to do a decent allocation. The object here is to determine where we lie in that problem space.

Structure Histogram

Simple Size	Percentage of total traffic 0	1	2	3	4	5	6	7	8
5x5	1101	2	2	4	1				
	153 (excluding AR structures)								
10x10	3350	3	0	4	0	2	1		
	310 (excluding AR structures)								
20x20	12050	3	0	3	1	2	0	1	
	632								
40x40	46250	3	0	3	1	0	2	1	
	1272 (excluding AR structures)								

Analysis of Hotness Significance Study

The results this study are quite interesting. There are about 10 structures that account for a very significant amount of the overall traffic. After a closer examination, one discovers that the three structures accounting for the largest amount of traffic are all constant tables. Some provision in the language or the machine to allow duplication of such structures could have major benefits. That would relieve any contention due to these structures. However, those three structures aside, there are still six or seven structures accounting for over 1% of the traffic each. And in fact, the numbers for the temporal mix will probably be worse than what is shown in the above tables (due to statistical fluctuations and the fact that the data structure lifetime is less than the program execution time). These structures are easily "hot" enough to cause the "hot spot" degradation that Pfister and Norton reported. This leads us to believe that contention for single structures may still be a problem in a dataflow context, even though we don't have the synchronization locks of RP3 or the NYU Ultracomputer.

4.3 Popularity of Data Structures

We define the "popularity" of a structure as the number of contexts (they may be different invocations of the same code block) that make use of it. This measure addresses a different side of the localization issue and should begin to give us some indication of how much we can gain by localizing structures. That is to say, "How often are we going to able to cleverly place a structure in the same place as most of the people who use it?" Clearly if a structure is very popular, trying to do

so would strongly constrain a resource manager's work distribution decisions. Conversely, if most structures aren't very popular, then we may be able to keep most of their contexts close by. Here are the statistics on structure popularity for several different sizes of Simple. The structures are categorized according to how many contexts referred to them -- the right column represents the number of structures that fell into each category. No Simple 5x5 stats are included because of the small number of contexts produced by execution of that problem size.

Simple 10 by 10

Number of Contexts	Number of structures	
0	0	
1	5	
2	3112 (74 without AR structures)	
3	69	
4	24	
5	8	
6	6	
7	16	
8	2	
9	20	
10	27	
11	12	
12	4	Size 10,11,12
13	5	
14	1	
16	1	
17	0	Size 11,12
18	18	
22	2	
23	14	
24	1	Size 11,12 probably upper
68	2	level structure of 2-D
91	2	arrays.
130	2	
131	2	5 elt. const. structs
194	2	
323	2	
516	1	40-element environment

Total structures: 3360

Total without AR structures: 322

55 Hottest Structures

These structures represented (26532/48700) = 54.5% of the overall traffic. A larger percentage if you exclude AR structures.

Number of Contexts	Number of structures
1	1
7	10
9	4
10	2
11	1
12	2
14	1
16	1
18	3

..... The most popular 30 structures were all present in this group of the 55 hottest.

22	2
23	14
24	1
68	2
91	2
130	2
131	2
194	2
323	2
516	1

Simple 20 by 20

Number of Contexts	Number of structures
0	0
1	5
2	11572 (154 without AR structures)
3	149
4	54
5	18
6	16
7	36
8	2
9	4
10	0
19	36
20	47
22	22
23	25
24	1
26	1
38	38
42	2
43	34
44	1
328	2
381	2
650	2
651	2
974	2
1623	2
2596	2

Total structures: 12060

Total without AR structures: 642

47 Hottest Structures

These structures represented (112569/220160) = 51.1% of the overall traffic. A larger percentage if you exclude AR structures.

Number of Contexts	Number of structures
20	3
21	2
22	2
23	1
24	1
26	1
38	4
42	2
43	17

-------- The most popular 14 structures were all present in this group of the 47 hottest.

44	1
328	2
650	2
651	2
974	2
1623	2
2596	1

Simple 40 by 40

Number of Contexts	Number of structures
0	0
1	5
2	45292 (314 without AR structures)
3	309
4	114
5	38
6	36
7	76
8	2
9	4
39	76
40	87
41	42
42	4
43	5
44	1
46	1
78	78
82	2
83	74
84	1
1448	2
1561	2
2890	2
2891	2
4334	2
7223	2
11556	1

Total structures: 46260

Total without AR structures: 1282

37 Hottest Structures

These structures represented (465129/939880) = 49.5% of the overall traffic. A larger percentage if you exclude AR structures.

Number of Contexts	Number of structures
39	1
40	9
41	2
42	2
43	1
44	1
46	1
78	4
82	2

-------- The most popular 14 structures were all present in this group of the 37 hottest.

84	1
1448	2
1561	2
2890	2
2891	2
4334	2
7223	2
11556	1

Analysis of Popularity of Structures Study

There are several interesting facts to note in the above study. We found that many of the structures are referenced by multiple contexts. If we remove the Argument Result structures from the count (as they will be removed when the new procedure call schema is implemented), this characteristic is even more pronounced. This would lead us to believe that unless the way the structure is shared by contexts is very simple, trying to keep the structure close to all of the contexts that use it is going to be very difficult.

We also found that a small number of structures are accounting for most of the traffic (around 50%) and those structures are used by a lot of contexts. If we wanted to reduce the network traffic by localizing data structures, these very "hot" ones would be prime candidates. However, since they are used by many contexts, it appears that getting much benefit from localizing the very "hot" contexts will be quite difficult.

The most popular structures were the global constant structures in the program. As we discussed

in the "hotness" study, these structures could be copied and distributed. Thus, we're not terribly interested in how many contexts use them. However, in this study we see that certain other structures are often used by more than 10 contexts. Localization of such structures could lead to memory contention.

5 Summary and Conclusions

This study has only begun to scratch the surface of the structure reference characterization problem. Despite that fact, the study has made several things quite clear. Supporting procedure invocation via I-structure interfaces is prohibitively expensive in terms of the number of I-structures used. With large problem sizes of Simple, the AR structures accounted for 97% of the structures. This confirms the general feelings which led to the development of a new procedure invocation schema that does not use I-structures.

In the course of the study we encountered a wide range of hotness ratings. This tells us that the amount of traffic a structure will incur may not be very strongly related to the size of the structure. Hence allocation schemes which only take free storage and structure size into account are not likely to do significantly better than random schemes.

The average number of reads per structure was quite low. We presume that this was largely due to huge number of AR structures. When the new procedure call schema is implemented, this number should increase dramatically. The variance of the number of reads per structure (which increased rapidly as we increased the problem size) should also increase. If that is the case, the large variance makes it unlikely that purely random "vertical" schemes are going to do very well. To balance the load on each memory unit, these schemes depend on the statistical effect of averaging a large number of random variables to reduce the overall variance. This only is successful if the number of random variables is very large, or the initial variance is small. Of course, these two things are characteristics of the program, so they are difficult to control them. However, the numbers collected in this study would lead us to believe that such random schemes will not do well.

There are several structures in each run of Simple that each account for a substantial percentage of the overall I-structure traffic. This is only considering cumulative references. These percentages seem large enough to cause hot spots in any medium scale (20-100 processors) multiprocessor. In fact, because most structures are considerably shorter lived than the execution of the overall program, one would expect many more structures to cause hot spots than the ones we found. One result clear from this study, unless we reduce the contention for structures (by distributing them, or some other means) we will not be able to fully utilize our machine.

The results of the Popularity study allow us to make the following argument. It seems that the number of contexts referencing the "hot" structures is relatively large. Ideally, the "hot" structures are the ones that you would like to localize in order to reduce latency and network traffic. However, the results of this study tell us that this may be very difficult. The structures accounting for the majority of the references are referenced by many contexts. Unless very few of these contexts are in existence at any time, or the relationship between them is very simple, it is unlikely we will be able to gain much locality by clever allocation. Furthermore, such allocation is likely to strongly constrain the resource manager's work distribution policy.

6 Implications for a Structure Mapping Policy

The results of the various parts of this study seemed to all point to one conclusion: doing anything clever in terms of structure mapping is going to be very difficult. The concrete evidence to back up this fact is as follows: First, the variance in the amount of traffic each structure accounts for is large, so we need some information about the traffic a given structure will draw. Second, the hottest structures tend to be accessed by many contexts. This means that any attempt to enhance locality will interact strongly with the work distribution policy. Finally, there seems to be little room for error, a few structures draw such a large percentage of the overall traffic that unless the hot spot problem is solved, localizing them could be disastrous.

The large percentage of traffic attracted by some structures coupled with the large variance in the amount of traffic to each structure seems to rule out any random vertical schemes. We would expect that some memories would be overloaded, and others greatly underutilized. The small margin for error (due to "Hot Spot" degradation) also leads us to discount these schemes.

Diagonal allocation schemes seem to offer a reasonable compromise between localization and contention. They offer perhaps the most promising approach in the long run. However, the interconnection of the network is likely to be very dense. Hence the most we could hope to gain here is a small logarithmic factor in latency. Because of the TTDA's ability to tolerate memory latency, this is not likely to be a crucial performance factor in our machine.

Horizontal allocation seems to be the safest scheme. It distributes the load of the "hot" structures over many memory modules, and thus increases the potential memory bandwidth each structure can have. These two characteristics go a long way towards addressing the concerns raised above. However, it seems unsatisfying as a solution because it relegates the memory latency seen by a processor to mediocrity. On the other hand, the strong point of the TTDA is that it can live with this

mediocrity and still provide high performance. Thus this scheme seems to be taking advantage to the TTDA's greatest strength.