South African
Computer
Journal
Number 2
May 1990

Suid-Afrikaanse
Rekenaar-
tydskrif
Nommer 2
Mei 1990

Computer Science and Information Systems

Rekenaarwetenskap en Inligtingstelsels

The South African Computer Journal
An official publication of the South African Computer Society and the South African Institute of Computer Scientists

Die Suid-Afrikaanse Rekenaartydskrif
'n Amptelike publikasie van die Suid-Afrikaanse Rekenaarvereniging en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

Subscriptions

	Annual	Single copy
Southern Africa:	R32-00	R8-00
Elsewhere:	$32-00	$8-00

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

Editorial

Information Systems Research: A Teleological Approach?

The request to write this editorial came at a very opportune time, coinciding as it did with an intense examination of the development of the field of information systems and an analysis of the progress of IS research. I have therefore used this opportunity to focus my thoughts and outline some of my conclusions. By doing so I don't pretend to answer any questions, merely perhaps to stimulate thought amongst those SACJ readers involved in IS research.

The last fifteen years has seen a tremendous growth in the study of information systems. During this period a number of journals devoted to IS research appeared such as *MIS Quarterly, The Journal of MIS, Information and Management* and *Data Base.* There are now many research-based activities: the International Conference on Information Systems; the annual IS doctoral dissertation colloquium; and various awards for IS research contributions. Hundreds of universities world-wide have formed information systems departments with (reasonably) standard curricula.

Yet with all this, what has *really* been achieved from a research viewpoint? Are we any closer to understanding the true nature of information systems? Is there a general unified theory of information systems? Is there even an accepted, unique body of IS knowledge? The answer to all of these must surely be no.

We have, I believe, achieved precious little. Yes, we do understand something of IS development approaches. We understand a little more now than we used to about how users interact with systems. But to get back to the first question, do we really understand what information systems *are* and how they work? No. Which begs the question: Why not?

There are, again I believe, a number of reasons, but the foremost must be that the majority of people in the IS research community either reside in the business schools of the USA or are drawn from other disciplines. These people, it would appear, are researching for research's sake; to publish in order to secure tenure or develop a research track record, not to further the body of knowledge of the subject. There seems an almost frantic zeal to generate and test hypotheses, trying to adopt and pursue what is seen to be a "scientific approach". But there is very little focus - there can't be, or the answers to my questions earlier would be yes rather than no!

Let me hasten to add that there is nothing unique about these IS researchers. "Publish or perish" is still very much alive and well! But also they are really not all that different from other social scientists. As Nagel [3] observed:

> "... in no area of social enquiry has a body of general laws been established, comparable with outstanding theories in the natural sciences in scope of explanatory power or in capacity to yield precise and reliable predictions ..."

Why should this be the case? Is it because the great intellects gravitate to the natural sciences and the social sciences pick up the second best who are incapable of generating these general laws? I hope not! The answer may well be that we have become locked into a particular research approach which is inappropriate to developing a body of social science, and more particularly, IS knowledge. Maybe we should be learning from our own source discipline (systems theory) and be developing a real research approach which complements our field of study.

To explore this further let me go back to the roots of information systems. What is an information system? Do we really have an accepted definition? Probably the most widely referenced is that provided by Davis and Olson [2]:

> "an integrated, user-machine system for providing information to support operations, management and decision-making functions in an organization. The system utilizes computer hardware and software; manual procedures; models for analysis, planning, control and decision making; and a database".

Note how this emphasizes the man-machine interrelationship and underscores computers as a core component when they are not even necessarily a part of the information system. The worst aspect is that it does little to describe what a system is, and this may well be one of the causes of our research dilemma. Again, if we draw on systems theory then a more appropriate definition might well be: "a hierarchical set of procedures utilizing information to monitor and control organizational performance". Note that this definition fits with general systems theory that *all systems* have four basic foundations: cybernetics, hierarchy, control and information [1].

An additional aspect not apparently recognised by IS researchers is that the information system, just like any other system, biological or otherwise, suffers from the problem first identified by our own Jan Christiaan Smuts [4]: that of holism. Simply put, this says that the whole is greater than the sum of the parts. This means that information systems, unlike science, cannot be reduced to simple isolated fields of enquiry and then analyzed or tested using hypotheses and laboratory experiments from which elaborate generalizations may be inferred. They have levels of complexity with new factors emerging at each level. The problem with most of the current research is that it starts out with a reductionist approach and then focuses on the highest (or lowest) level. Thus the majority of the topics have as their target the interaction between user and computer or the management or application of technology. There is very little research that is taking place at fundamental level, that of developing a general theory of information sytems. This is the teleological approach, searching for the natural laws and developing the theory based on deduction and logical development. Until we can advance *that* area of knowledge and, from a basis of these fundamental laws, develop a hierarchy of hypotheses that can *then* be tested, we will have little focus to our IS research. It will remain a fragmented, uncohesive smattering of the work of individuals who are merely grasping at tenure. There are few people who would today argue against the inclusion of information systems as a field of study at a university or as a fruitful research area. But until such time as we focus on the foundation theory, it will remain unstructured and immature.

References

[1] P Checkland, [1984], *Systems Thinking, Systems Practice*, John Wiley and Sons, New York.

[2] Davis and Olson, [1985], *Management Information Systems: Conceptual Foundations, Structure and Development*, 2nd Ed: MacGraw-Hill, New York.

[3] E Nagel, [1962], *The Structure of Science*, Routledge and Hegan Paul, London.

[4] J C Smuts, [1929], *Holism and Evolution*, MacMillan, London.

Prof Peter Lay
Assistant Editor: Information Systems

This SACJ issue is sponsored by
Department of Computer Science
University of Cape Town

Predicting the Performance of Shared Multiprocessor Caches

H A Goosen and D R Cheriton
*Computer Science Department, Stanford University, Stanford CA 94305, USA**

Abstract

We investigate the performance of shared caches in a shared-memory multiprocessor executing parallel programs, and formulate simple models for estimating the load placed on the bus by such a shared cache. We analyze three parallel program traces to quantify the amount of sharing that takes place during program execution. These results indicate that shared caches can substantially reduce the load placed on a bus by a large number of processors. Finally we propose a new metric of group locality to aid in predicting the performance of a parallel program running on a machine with shared caches.

Computing Review Categories: H.I.4, I.2.6
Keywords: shared-memory multiprocessors, parallel processing

Presented at Vth SA Computer Symposium

1 Introduction

There is considerable interest in the design of scalable shared memory multiprocessors. The problem of building such machines is largely that of building a memory system that is fast enough to supply the multiple processors with the data they need to execute programs and communicate with each other.

A bus is an attractive option for connecting multiple processors to a shared memory since it is cheap, reliable, and has low latency. Unfortunately, the bus is a bottleneck between the processors and the memory, which means that only a small number of processors can be connected to the bus before it saturates. This problem is solved by a tree-structured memory system consisting of a hierarchy of shared buses and caches. In this case only a small number of processors are connected to each bus, easing the bandwidth requirement of each individual bus. Several such machines have been proposed, e.g., the Encore Gigamax [13], and the VMP-MC multicomputer [3].

The VMP-MC architecture exploits shared caches to increase the scalability of the machine. Preliminary results indicate that substantial reductions are possible in the shared memory bandwidth required per processor when the machine is executing parallel applications. This enables more processors to be used in the system, and therefore increases the potential performance of the machine.

In the next section we motivate the use of shared caches for a large-scale multiprocessor running parallel programs. Next we describe simple models for estimating the load placed on the bus by a shared cache. We present the results of an analysis of parallel program traces, which provide an indication of the potential benefit that can be gained from using shared caches. Finally we propose a new metric of group locality which can assist in predicting the performance of a parallel program running on a machine with shared caches.

*This work was sponsored in part by the Defense Advanced Research Projects Agency under Contract N00014-88-K-0619, and by an IBM graduate fellowship.

2 Background

Processor cycle time is decreasing much faster than memory cycle time. Since the performance of a machine is usually measured by the number of instructions executed per second, the average memory access time has to be close to the processor cycle time for the performance of the machine to approach its peak performance.[1] This usually requires a multi-level cache design [8]. In addition, it has become feasible to integrate large instruction and data caches on-chip (12 Kbytes in the Intel i860). In such a multilevel cache hierarchy, the majority of the traffic (90% to 99% in uniprocessors with large cache blocks [9]) is absorbed by the first-level cache, which means that the higher level caches are idle most of the time.

In a multiprocessor system one can increase the utilization of a higher-level cache by sharing it among several procesors. In addition, sharing a cache between several processors executing the same parallel program can improve the hit ratio of the shared cache, and so increase the scalability and performance of the machine.

Figure 1 shows an example of a shared memory multiprocessor. The bus has sufficient bandwidth to support four processors (each with an on-chip cache) connected to external caches. If a group of four processors are connected to each external cache, the load presented to the bus by the group will be lower than the load presented by four processors, each with an external cache, even if the caches are of the same size [3]. This means that the bus can now support more than four processors, where each group of four processors shares a cache, as shown in Figure 2. It is also worth noting that the memory access latency is not increased (to first order[2]), since both systems have 2 levels of caching.

We previously investigated the performance of a shared board-level cache [3]. Trace driven simulations were run of several different 16-processor machines. The basic

[1] For high-performance microprocessors the peak performance is approximately (cycle time)$^{-1}$, since these processors execute most instructions in one cycle.

[2] This assumes that the utilization of all shared resources are low enough that queueing delays are not a factor.

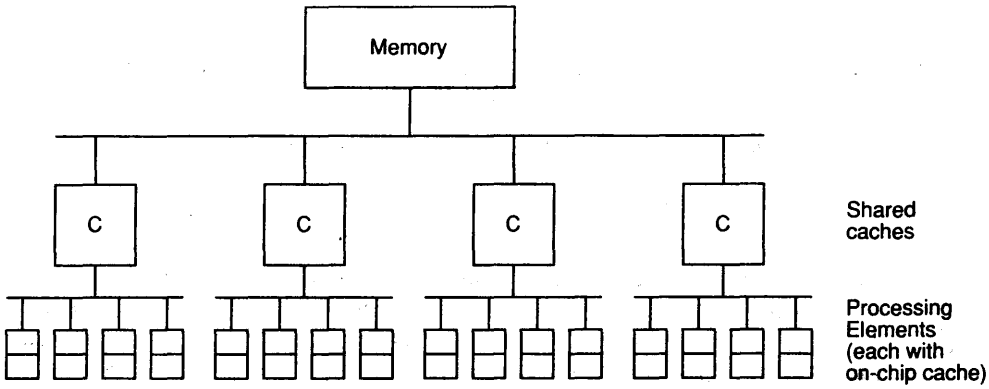

Figure 2: Cache sharing.

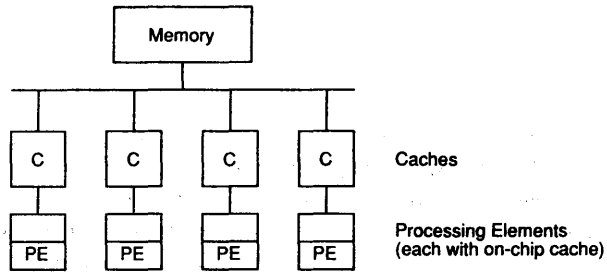

Figure 1: No cache sharing.

machine structure is similar to that shown in Figure 2, which shows a 16-processor machine with four processors (each with an on-chip cache) sharing each board-level cache. In these simulations, the on-chip caches were 16 Kbyte unified caches with 32 byte blocks, while the board-level caches were 512 Kbyte unified caches with 128 byte blocks. These basic parameters were chosen to represent a "good" design,[3] and were not varied in the simulation runs. The experiment consisted of running the three 16 processor traces against machines which had different numbers of processors sharing a board-level cache. These three applications are described in the Appendix. We consider these applications to be representative of general-purpose computer-aided design tools and simulators. The programs were not optimized for this architecture.

Table 1 shows the decrease in the shared cache miss ratio as we increase the number of processors sharing the cache from 1 to 8. The miss ratio decreases by 55% for *dcsim*, 57% for *mp3d*, and 61% for *locusroute*. The lower miss ratios imply a reduction in the average memory access time, and a reduction (of about 50% to 60% as well) in traffic on the global shared bus.

These results are quite promising, and indicate that a machine which has 2 groups of 8 processors sharing a board-level cache would have up to 60% less global bus traffic than a system with only one processor per board-level

[3]The on-chip cache parameters represent our guess as to what will be commercially available on microprocessors in 1990. The shared cache parameters were chosen after making a few runs to find the maximum block size that did not dramatically increase the number of coherence invalidations required.

	Processors per shared cache				
Name	1	2	4	6	8
mp3d	77	67	54	43	33
dcsim	20	17	14	11	9.3
locusroute	3.3	3.1	2.2	1.7	1.3

Table 1: Shared cache miss ratio (% of references to the shared cache)

cache, with the same size caches. This means that we can put roughly twice as many processors on the bus when sharing the caches by 8 processors, compared to the case where we do not share the caches. This should enable us to double the performance of our system.

These results do not explain where the reduction in traffic is coming from, or how to further reduce the global bus traffic. It is also not clear how these results will apply to much larger systems. In order to answer these questions, we will now present simple models for shared cache behavior, and use it it to estimate the performance of such a cache.

3 Dynamic Behavior of Shared Caches

We shall discuss cache sharing in terms of the terminology introduced in Figure 3. Several processing elements (PEs) are attached to a bus, which is connected to the input port of the shared cache. Each PE issues a stream of memory references that flows in the direction of the shared memory. We shall refer to subsystems which are closer to the source of the references than the shared cache is as *upstream*, and anything closer to memory as *downstream*.

The reference streams of all the PEs are merged, and this is the input to the shared cache. The shared cache, in turn, generates output references to the downstream system. Block transfers occur in response to the memory references: from the shared cache to upstream PEs, and from a downstream cache or memory module to the shared cache.

The n processors in the system are divided into m *groups*, where each group of $p = n/m$ processors shares a cache. This is shown in Figure 4. The nodes labeled $P_1, P_2, \ldots, P_n$ are the processors, the nodes $S_1, S_2, \ldots, S_m$ are the shared caches, and the node labeled *Bus* is the down-

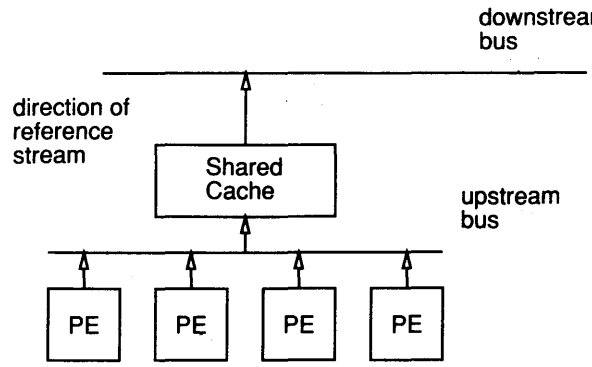

Figure 3: Terminology for shared caches.

stream bus. The shared caches S_i are the same size, irrespective of the number of PEs sharing the cache. In the special case where there is no sharing ($m = n, p = 1$), the shared caches will be denoted by $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_n$.

The amount of data that has to be transferred from the downstream memory system into a PE or cache will be called the *load* placed by the PE or cache on the downstream bus. The load placed on the bus by S_i is denoted by l_i. The load is measured in number of block transfers per second.[4] We will use $\hat{l}_i$ to represent the load of $\hat{S}_i$.

The main purpose of a shared cache is to reduce the load that is placed on the downstream system. For a bus with a fixed capacity, a reduction in the per-processor load means we can connect more processors to the bus without saturating it. Provided that the latency does not increase too much, the result is that the performance of the system will increase (for programs which can use the extra processor cycles).

In this study, the performance of a shared cache configuration will be measured by the ratio R between the load placed on the downstream bus by the shared caches, and the load placed on the downstream bus if caches are not shared:

$$R = \frac{\sum_{i=1}^{m} l_i}{\sum_{j=1}^{n} \hat{l}_j} = \frac{L}{\hat{L}}$$

L is the total load placed on the downstream bus by the m shared caches, while $\hat{L}$ is the total load placed on the bus by the n caches when only one processor shares each cache. If $R = 1$ for a given system, it means that it performs the same as a system with private caches. If $R < 1$, the system performs better than a system with private caches. In the next sections we examine the performance of a shared cache system executing a parallel program.

3.1 Parallel Program Behavior

A parallel program is considered to execute in a single address space, with multiple lightweight processes (threads). Two processes (threads) share a block simply by referring to it with the same virtual address. The programming paradigm we consider here uses a number of processes executing identical code on different items of shared data [5]. Data items are obtained from one or more queues. Barrier synchronization is used to separate different phases in the computation.

We consider references to a given cache block of data, and determine the amount of bus traffic that results from a reference to that block. Program references can be divided into three groups on the basis of block behavior: references to (1) private blocks, (2) shared read-only blocks, and (3) shared read/write blocks.

3.1.1 Private blocks

Private blocks interfere with our goal of reducing the downstream bus traffic, since a block which is referenced by one processor, and brought into the shared cache, will not subsequently be referenced by another processor. Although these references form a substantial fraction of the data references (60% to 70%) in the parallel programs we examined, most of these references hit in the caches, and do not cause any bus traffic.

Most of the code in a shared program is shared by all the processors (at least in the paradigm we consider), so there are few instruction fetches from private code. Private data is dominated by references to the stack and dynamic temporary variables.

Finally, the shared caches are large and block replacement is rare. Private blocks tend to stay in the cache for a long time, which means that reference to private blocks is largely a cold-start phenomenon. Therefore we will not further consider private data in this paper.

3.1.2 Shared read-only blocks

The *degree of sharing* (k) of a shared block is defined as the number of different processors that access the block during a specified window of duration w cycles.

For shared read-only blocks, the reduction of traffic will depend on k, with w equal to the program execution time (assuming no context switches). A simple example will illustrate this. Suppose we have two systems, each with four processors. In the one case, each group of two processors share a cache, while in the other case, each processor has its own private cache. If all four processors reference a specific block, $k = 4$. The bus traffic in the non-shared case will be four block transfers (i.e., $\hat{L} = 4$), while in the shared case it will be two (i.e., $L = 2$). The shared case therefore requires 50% of the bus bandwidth required by the non-shared case (i.e., $R = 0.5$). If the degree of sharing was lower, this will change. If only processors 1 and 4 read the data, there would be no reduction in bus traffic for the shared case compared to the non-shared case, and therefore $R = 1$.

Although R depends on the exact sequence of references made by each processor, we can estimate the value of R under a set of reasonable assumptions. We assume infinite caches, and that blocks will never be replaced once they are in a cache. If a block is shared by k processors, then there are $\binom{n}{k}$ ways to assign the block to n processors. Each of these combinations c, where $1 \leq c \leq \binom{n}{k}$, places a load L_c on the downstream bus. We can now find the expected total

[4] In this model we consider only the data transfers. In reality there are also other bus transactions, e.g., invalidation signals, but most of the bus bandwidth is consumed by the block transfers.

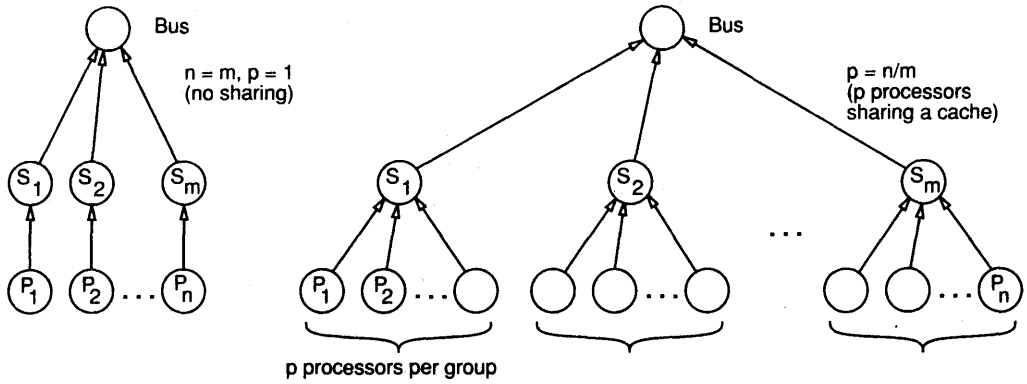

Figure 4: Sharing structure.

load C on the downstream bus by summing the products of $P\{c\}$ (the probability of c) and L_c. We assume that each combination c is equally likely, so that $P\{c\} = 1/\binom{n}{k}$:

$$C = \sum_{c=1}^{\binom{n}{k}} L_c P\{c\} = \frac{1}{\binom{n}{k}} \sum_{c=1}^{\binom{n}{k}} L_c \qquad (1)$$

The load L_c can be calculated by noticing that one block transfer is required into every shared cache which contains at least one processor referencing that block. If there are no processors referencing the block in a particular shared cache, no block transfer is required into that cache.

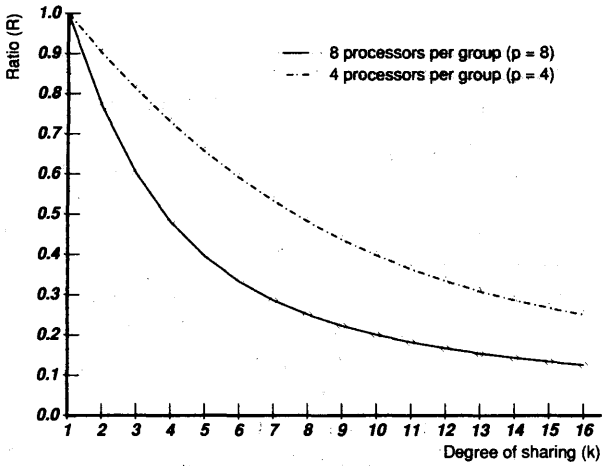

Figure 5: Ratio R for $p = 4$ and $p = 8$ configurations, $n = 16$.

Figure 5 shows R for two different configurations of a 16 processor system, calculated using equation 1. In the first configuration, the system consists of four groups of four processors each ($m = p = 4$), and in the second it consists of two groups of eight processors each ($m = 2, p = 8$). We observe that R decreases rapidly for small values of k (as k change from 1 to 2, R decreases 10% in the $p = 4$ case and 20% in the $p = 8$ case). As k increases, the reduction in R becomes less dramatic. If $k = n$, then $R = 1/p$, which means the downstream traffic out of a shared cache is independent of the number of processors sharing the cache.

To get an idea of the performance improvement one can expect from real programs, we measured k for all the read-only shared data in the three programs (*mp3d*, *dcsim*, and *locusroute*) described in Appendix. The histograms in Figures 6 through 8 show the number of reads to all read-only blocks for each value of k.

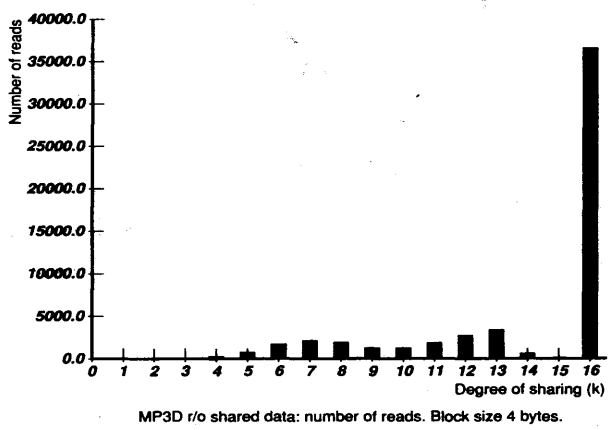

Figure 6: Number of references to shared read-only data for *mp3d*.

From these histograms we can calculate an average value for k to use in estimating R. For *mp3d*, $k = 14$, for *locusroute* $k = 4.5$, and for *dcsim* $k = 8.5$. These values were obtained by weighting each value of k according to the number of references it received. R can be read off the graph in Figure 5.

With finite caches, replacement interference becomes a factor. If the shared cache is the same size as the non-shared cache, it is likely that the interference in the shared cache will be higher than that in the non-shared cache, simply because the combined working sets of the PEs connected to the shared cache will be larger than the working set of the single PE connected to the non-shared cache. However, with very large caches this will be a small effect, since replacement is very rare in large caches. In addition, it is always possible to increase the size of the shared cache to the point where the interference will be comparable to that

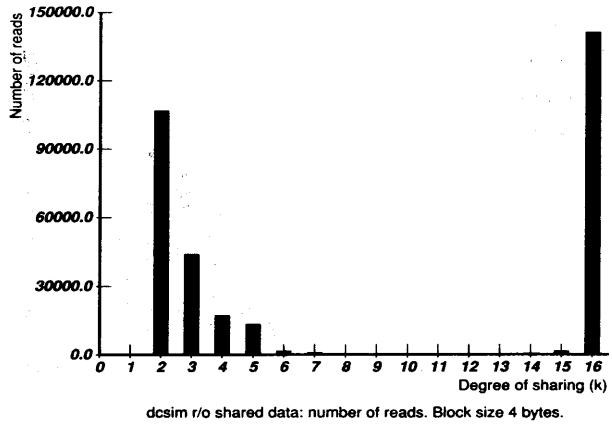

Figure 7: Number of references to shared read-only data for *dcsim*.

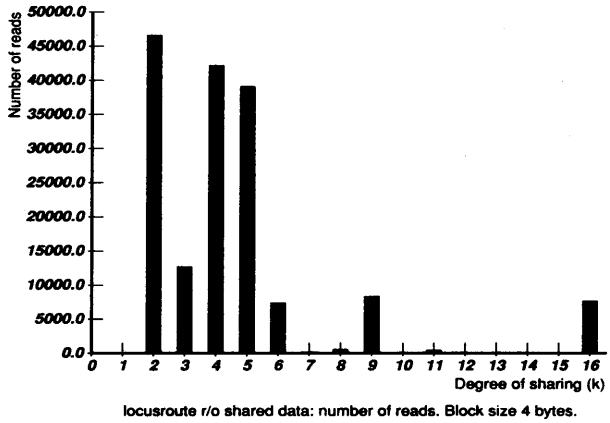

Figure 8: Number of references to shared read-only data for *locusroute*.

of the non-shared case.

3.1.3 Read/write shared blocks

The behavior of read/write shared blocks is more complex, since consistency has to be maintained between all the caches. The details of this behavior will depend on the cache coherence protocol that is used. As a reference point, this discussion will be carried out in terms of the ownership protocol used in the VMP multiprocessor [4].

The VMP ownership protocol works as follows: a cache block can be in one of two states, shared or private. In shared mode, only read access is allowed. If a processor wants to write to a block, the block has to be in private mode. In shared mode, a block can be present in more than one cache at a time, while in private mode it may only be in one cache at a time. When a block is read in private mode, all copies of the block in other caches have to be invalidated. The protocol also allows a shared block to be converted to a private block, but we will not consider that case here.

k, with w equal to the execution time for the program, is not a useful parameter for predicting the performance of a shared cache on read/write shared data. The reason for this is that read/write shared blocks typically do not stay in a cache for very long, but are invalidated whenever another processor wants to write to that block. To take this into consideration, we propose the *readership* metric (d):

Readership This is defined as the average number of different processors (excluding the owner of the block) which read a block between the start of one write run and the start of the next write run[5] on the block. The readership measures the extent of read-sharing of read/write shared blocks. If the readership of a block is high, a shared cache can reduce the traffic on the downstream bus as we discussed earlier for read-only shared blocks.

For example, consider the reference stream shown in Table 2, generated by processors in a four-processor system (all references are to the same block x). Pn refers to processor n, and Cn refers to the private cache of processor n. In this example $d = 2$, since two other processors (processors 2 and 3) read the block between the two write runs.

Reference	Description
P1 writes x;	x moved into C1, private
P2 reads x;	x writeback, moved into C2 shared
P1 reads x;	no action, x still in C1 shared
P2 reads x;	no bus traffic, x still in C2 shared
P3 reads x;	x moved into C3 shared
P4 writes x;	x invalidated in C1, C2, and C3, moved into C4 private

Table 2: Example reference stream.

Since the general behavior of these blocks is complex, we will investigate the behavior for two simple systems of n processors. Assume that $k = n$, and consider the ratio R for two extreme cases: $d = n - 1$ and $d = 0$. Again, we consider infinite caches, fully associative, with uniformly distributed references from all processors which share a block.

1. If $d = n - 1$, all processors are expected to read the block before somebody writes it. To calculate $\hat{L}$, we notice that there will be $n - 1$ transfers of the block when everybody reads it, followed by 1 transfer when it is written back. This means there will be n transfers on average for every write. In the case where p processors share a cache, there will be m transfers for every write, using the same arguments as for read-only shared blocks. Therefore, $R = m/n = 1/p$.
2. When $d = 0$, all references to the shared block are writes (this serves to show the other extreme of behavior). The distribution is uniform, so the probability that any P_i will read the block is $1/n$. To calculate $\hat{L}$, we observe that there are two cases: if P_i has a block in private mode, the next write will be from P_i with probability $1/n$ and cost 0, or it will be from one of the other processors with probability $(n - 1)/n$ and

[5] A write run [5] is defined as a sequence of write references to a shared block by a single processor, uninterrupted by any accesses by other processors.

cost 2 block transfers (one for the writeback, one for the read). It follows that $\hat{L} = 2(n-1)/n$. L is calculated in a similar way by observing that, if P_i has the block in private mode, it will also be in the shared cache S_j, where $j = \lfloor i/p \rfloor + 1$. The next processor to read the block will be in the same group with probability $1/m = p/n$ and cost 0, or in another group with probability $(m-1)/m$ and cost 2 block transfers. Therefore, $L = 2(m-1)/m$, and

$$R = L/\hat{L} = \frac{n(m-1)}{m(n-1)}$$

For the case where we have 16 processors, with 2 groups, each with 8 processors, $R = 8/15 = 0.53$, and with 4 groups of 4 processors each, $R = 4/5 = 0.8$.

We can expect the behavior of a real program to lie somewhere in between these extremes, assuming that references to the shared blocks are uniformly distributed across processors.

The preceding discussion assumed a uniform distribution of references from processors sharing a block. However, in the case of read/write shared data, there is the potential to achieve very small values of R by appropriately structuring the algorithm and data structures. For example, if we can ensure that all processors in one group will write to the block before a processor from another group writes to it, R will improve to $m/n = 1/p$, the same as for shared read/only data. We can do even better than this if we write programs so that read/write data is partitioned so that only one group accesses a partition most of the time. In this case, R can approach 0.

This is of course predicated on the degree of sharing that actually exists in parallel programs: if it turns out that k is small, it will not help very much to organize data for maximum sharing. To get an idea of the values of k that one can expect in real programs, we collected the same type of histograms for read/write shared data that we earlier showed for read-only data. These are shown in Figures 9 through 11.

Calculating average values from these histograms (using the same method as for read-only blocks) yield $k = 12$ for mp3d, $k = 7$ for dcsim, and $k = 6$ for locusroute. These values suggest that it may indeed be possible to achieve significant improvements in R by proper data placement and the best assignment of processes to processors. This is especially important since our simulations indicate that read/write shared data accounts for most of the traffic in a shared-memory multiprocessor executing parallel programs.

We are developing a measure of this group locality, so that we can evaluate how successful our efforts at program restructuring are. Such a metric must be a good predictor of shared cache performance, and must also be computationally feasible. For a system with n processors, m groups, and p processors per group, where the memory is divided into blocks of equal size, and we are executing a parallel program, we define the following:

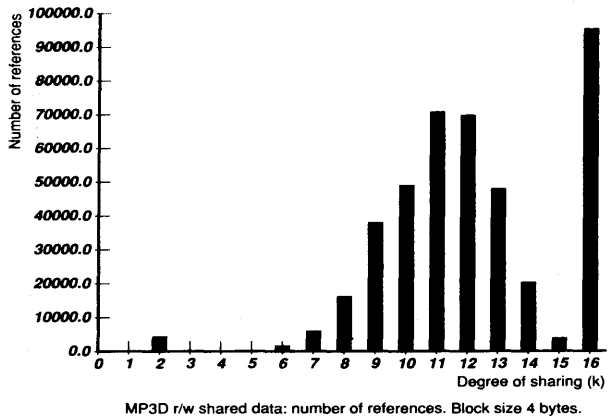

Figure 9: Number of references to shared read/write data for *mp3d*.

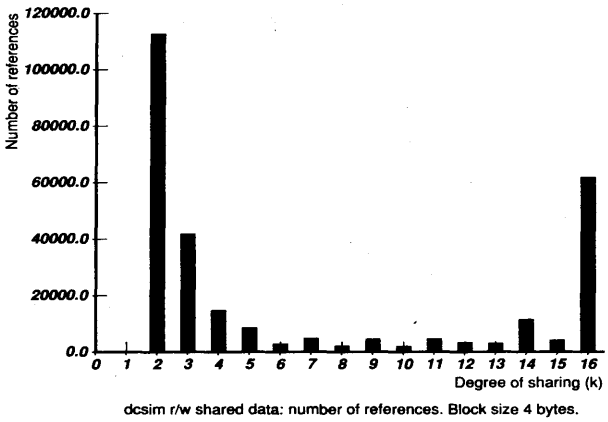

Figure 10: Number of references to shared read/write data for *dcsim*.

Definition 1 (Trace) *A trace T is a sequence of references $\{r_1, r_2, \ldots\}$, where each reference r_i is a tuple (processor, block, reference type). The reference type is either read or write.*

Definition 2 (Block trace) *For a given block b and trace T, the block trace T_b is a sequence of references $\{b_1, b_2, \ldots\}$, where the b_i are all the references to b in T, and where the order of the references in T are preserved.*

Definition 3 (P-group) *A P-group is any set of cardinality p of processors.*

A P-group corresponds to one possible assignment of processors to a shared cache. We will now define a measure of the quality of such an assignment. Since the order of the processor assignment is not important, there are $\binom{n}{p}$ P-groups for this system.

Definition 4 (G-subsequence) *For a given block trace T_b and P-group g, a G-subsequence is any maximum length subsequence from T_b such that (1) the subsequence contains no writes by processors which are not elements of g, and (2) the subsequence contains at least one reference by each processor which is an elements of g.*

Definition 5 (Weight of G-subsequence) *For a given G-subsequence s and P-group g, the weight of s is the number*

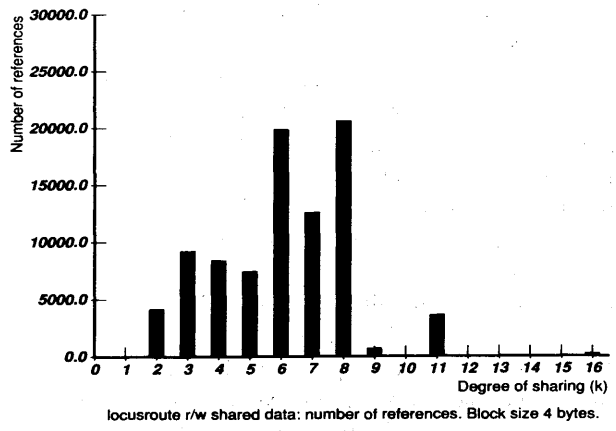

Figure 11: Number of references to shared read/write data for *locusroute*.

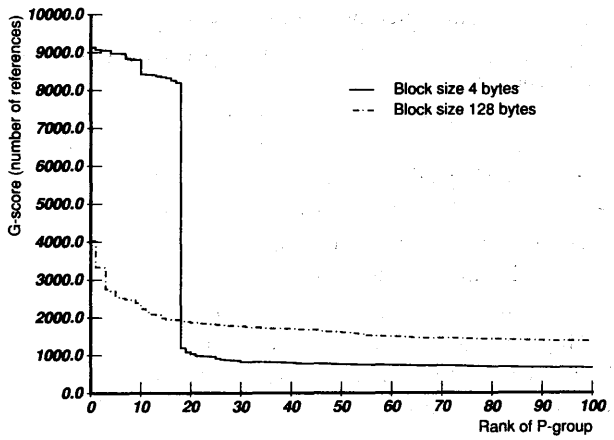

Figure 12: G-scores for the first 100 P-groups.

of references in s by elements of g.

Definition 6 (G-score) *For any P-group g, and all blocks b in memory, the G-score (measured in number of references) is the sum of the weights of all G-subsequences in all block traces T_b.*

The G-score counts the number of references that are made during only those subsequences which reference *all* the processors in a P-group. The G-score should therefore be an indication of the quality of the processor assignment associated with the P-group.

We measured the G-score for all the P-groups that occur in the *mp3d* trace, and ranked the P-groups according to G-score. For this 16-processor system with 2 groups of 8 processors per shared cache, there are $\binom{n}{p} = \binom{16}{8} = 12870$ P-groups. Figure 12 shows the G-score for those 100 P-groups with the highest G-scores.

The results suggest that the 4 byte blocksize has greater group locality than the 128 byte blocksize. Examination of the program revealed that most of the data were in fact word-aligned, and accessed on a word basis (4 bytes) by each processor, so that the 128 byte blocksize resulted in large amounts of false sharing, and hence poor group locality. It is also interesting to note that there is a relatively sharp rolloff in the G-score plot, suggesting that it may be advantageous to assign processes to processors according to the P-group with the highest G-score.

4 Comparison with Related Work

Multi-level bus structures were first proposed in the Cm* [10] architecture, which was a distributed-memory message-based system. [7] describes a dual processor machine which has a multilevel cache structure. Each processor has a private 64 Kbyte cache, and the two caches are connected to a shared 512 Kbyte second level cache. The main motivation seems to have been economy, and not an increase in performance. Yeh [15, 14] suggested the use of shared caches for avoiding the cache consistency problem. They recognized that the sharing of operating system code would probably improve cache behavior. The Alliant FX series machines [2] employ shared caches at different points in their multiprocessor machines, including shared first-level caches. To our knowledge they have not published any performance analysis of their shared caches. The Sequent Balance 21000 has two processors connected to a single first-level cache on every board. This seems like a bad idea, since the processors will compete for cache access on *all* references, and not just on first-level cache misses.

A comprehensive overview of multi-level cache structures for multiprocessors is given by Wilson [13] He concentrated more on the multiple bus aspect than on shared caches. He also proposed a local memory system to get around the problem of downstream bus bandwidth, with a shared cache to take care of the rest of the traffic. He estimated the required shared cache sizes as an order of magnitude larger than the sum of all the lower level caches, which is not supported by our analysis or measurements.

It is interesting to compare our approach with the context switching approach [12]. In this design, the processor absorbs the high memory access latency expected by switching contexts. A suggested implementation of this idea with RISC processors is to have several processors on one board, connected to the cache. Each processor is loaded up with process state, and one executes until it misses in the cache, at which point the next one is started. The advantage of the shared cache approach is that one effectively gets the same behavior when a processor misses, except that all the processors can execute concurrently while they hit in their on-chip caches.

Eggers [5, 6] proposes the notion of a write run, which we use in our definition of readership. The length of write runs is a measure of processor locality. This is useful for comparing cache coherence schemes that differ in the way they treat write sharing. However, it does not provide much insight into the behavior of shared caches.

Weber and Gupta [11] counted the number of invalidations on a write to a block. By averaging their data from 1 to infinity, an estimate of the readership can be obtained. They provided distributions of the number of invalidations per write, so one can actually calculate the readership distribution from their data. Agarwal and Gupta [1] defined and measured clings and pings for shared data, and also pro-

vided some initial data on write invalidation distributions for small numbers of processors. This was done primarily to analyze the relative performance of different cache coherence protocols, and to evaluate the feasibility of directory schemes with small numbers of pointers to processors storing copies of a block.

5 Future Study

This paper describes work in progress, and we are currently busy validating the models against trace-driven simulations. Specific efforts include: measuring the effect of data restructuring on the group locality measure and the extent to which that is a predictor of R, measuring the effect of different process-to-processor assignments on the locality measure and R, comparing different group locality metrics in terms of computational cost and quality, and measuring the effect of block size on locality.

6 Summary and Conclusions

We presented simple models that can be used to arrive at rough estimates of the expected performance of a shared cache under various circumstances. These models show that a shared cache can reduce the load placed by multiple processors on a global bus, provided that cache blocks are shared by the processors connected to the shared cache. We presented initial results of a study of parallel program traces, and gave values for the degree of block sharing for these programs. This showed that, in the traces we examined, both read-only and read-write shared data were shared by a significant fraction of the processors in the system. Finally we introduced a new measure of group locality which can be used to study the the performance of a shared cache.

References

[1] A. Agarwal and A. Gupta. Memory reference characteristics of multiprocessor applications under MACH. In *Proc. SIGMETRICS*, 1988.

[2] Alliant. FX/Series product summary. Technical report, Alliant Computer Systems Corporation, June 1985.

[3] D.R. Cheriton, H.A. Goosen, and P.D. Boyle. Multilevel shared caching techniques for scalability in VMP-MC. In *Proc. 16th Int. Symp. on Computer Architecture*, pages 16–24, May 1989.

[4] D.R. Cheriton, G. Slavenburg, and P. Boyle. Software-controlled caches in the VMP multiprocessor. In *Proc. 13th Int. Conf. on Computer Architecture*, pages 366–374, June 1986.

[5] S.J. Eggers and R.H. Katz. Sharing in parallel programs. In *Proc. 15th Int. Symp. of Computer Architecture*, pages 373–382, June 1988.

[6] S.J. Eggers and R.H. Katz. The effect of sharing on the cache and bus performance of parallel programs. In *Proc. ASPLOS-III*, pages 2–15, April 1989.

[7] A. Hattori, A Koshino, and S. Kamimoto. Three-level hierarchical storage system for FACOM M-380/382. In *AFIP*, pages 253–262. AFIP, June 1982.

[8] S. Przybylski, M. Horowitz, and J.L. Hennessy. Performance tradeoffs in cache design. In *Proc. 15th Int. Symp. on Computer Architecture*, pages 290–298, May 1988.

[9] A.J. Smith. Cache Evaluation and the Impact of Workload Choice. In *Proc. 12th Int. Symp. on Computer Architecture*, pages 64–73, June 1985.

[10] R.J. Swan, S.H. Fuller, and D.P. Siewiorek. Cm*: a modular multi-microprocessor. In *AFIPS Conf. Proc.*, volume 46. National Comp. Conf., 1977.

[11] W.D. Weber and A. Gupta. Analysis of cache invalidation patterns in multiprocessors. In *Proc. ASPLOS-III*, pages 243–256, April 1989.

[12] W.D. Weber and A. Gupta. Exploring the benefits of multiple hardware contexts in a multiprocessor architeture: Preliminary results. In *Proc. 16th Int. Symp. on Computer Architecture*, pages 273–280, May 1989.

[13] Andrew W. Wilson, Jr. Hierarchical cache/bus architecture for shared memory multiprocessors. In *Proc. 14th Int. Conf. on Computer Architecture*, pages 244–253, June 1987.

[14] P.C.C. Yeh. *Shared cache organization for multiple-stream computer systems*. PhD thesis, Univ. Illinois, 1982. Coordinated Science Lab., Rep. R-904.

[15] P.C.C. Yeh, J.H. Patel, and E.S. Davidson. Shared cache for multiple-stream computer systems. *IEEE TC*, C-32(1):38–47, January 1983.

Acknowledgments: We are grateful to Wolf Weber and Anoop Gupta for providing the traces used in this study. Thanks are also due to Joe Pallas, Ed Sznyter, Kieran Harty and John Hennessy for comments on earlier drafts of this paper.

Appendix The Program Traces

The traces used in these studies were all multiprocessor traces of 16-processor machines, obtained by running a multiprocessor simulator using the VAX T-bit mechanism to step the processes through their references in round-robin fashion. The traces do not include operating system references. Each trace consists of more than 7 million references, of which about half are data references. For a more detailed description of the programs see [11].

Locusroute: This is a global router for VLSI standard cells. Each processor removes a wire from the task queue and selects the best route for that wire. The cost data structure on which the routing is based is shared by all the processors, and updates to this global structure is made without locking.

Mp3d: This is a three-dimensional particle simulator for rarefied flow. Particle properties are stored in separate arrays. During each time step, the particles are moved one at a time. One lock protects an index into the global particle array. Because of the poor data layout, this program will not perform well on a cache-based multiprocessor.

Distributed Csim: This is a distributed logic simulator which does not rely on a global time during simulation. The trace does not include references to locks.

NOTES FOR CONTRIBUTORS

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as a Communications or Viewpoints. While English is the preferred language of the journal papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted **in triplicate** to the editor.

Form of Manuscript

Manuscripts for review should be prepared according to the following guidelines.

- Use double-space typing on one side only of A4 paper, and provide wide margins.
- The first page should include:
  - title (as brief as possible);
  - author's initials and surname;
  - author's affiliation and address;
  - an abstract of less than 200 words;
  - an appropriate keyword list;
  - a list of relevant Computing Review Categories.
- Tables and figures should be on separate sheets of A4 paper, and should be numbered and titled. Figures should be submitted as original line drawings, and not photocopies.
- Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin. Distinguish clearly between such cases as:
  - upper and lower case letters;
  - the letter O and zero;
  - the letter I and the number one; and
  - the letter K and kappa.
- References should be listed at the end of the text **in alphabetic order** of the (first) author's surname, and should be cited in the text in square brackets. References should thus take the following form:
  [1] E Ashcroft and Z Manna, [1972], The translation of 'GOTO' programs to 'WHILE' programs, *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255.
  [2] C Bohm and G Jacopini, [1966], Flow diagrams, Turing machines and languages with only two formation rules, *Comm. ACM*, **9**, 366-371.
  [3] S Ginsburg, [1966], *Mathematical theory of context free languages,* McGraw Hill, New York.

Manuscripts *accepted* for publication should comply with the above guidelines, and may provided in one of the following three formats:

- in a **typed form** (i.e. suitable for scanning);
- as an **ASCII file** on diskette; or
- in **camera-ready** format.

A page specification is available on request from the editor, for authors wishing to provide camera-ready copies.

Charges

A charge per final page, scaled to reflect scanning, typesetting and reproduction costs, will be levied on papers accepted for publication. The costs per final page are as follows:

Typed format:	R80-00
ASCII format:	R60-00
Camera-ready format :	R20-00

These charges may be waived upon request of the author and at the discretion of the editor.

Proofs

Proofs of accepted papers will be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Note that, in the case of camera-ready submissions, it is the author's responsibility to ensure that such submissions are error-free. However, the editor may recommend minor typesetting changes to be made before publication.

Letters and Communications

Letters to the editor are welcomed. They should be signed, and should be limited to about 500 words.

Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book reviews

Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertisement

Placement of advertisements at R1000-00 per full page per issue and R500-00 per half page per issue will be considered. These charges exclude specialized production costs which will be borne by the advertiser. Enquiries should be directed to the editor.

South African Computer Journal

Number 2, May 1990
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 2, Mei 1990
ISSN 1015-7999

Contents