Vol. 36 No. 7 Transactions of Information Processing Society of Japan July 1995

Regular Paper

Performance Analysis of a Data Diffusion Machine with High Fanout and Split Directories

HITOSHI OI †

The Data Diffusion Machine is a virtual shared memory architecture which has the advantage that data migrates from node to node when needed. However its disadvantages compared with other shared memory architectures such as CC-NUMA are higher miss penalties due to its hierarchical structure in interconnection and contention of the transactions at higher level directories. One way to alleviate these disadvantages is by increasing fanout and splitting directories. We analyze the performance improvement of the DDM by adopting these two schemes by extending the experimental results obtained from the DDM emulator. From the emulation result of mp3d running on 3×3 configuration, the performance of a DDM with flat 9-node configuration has been estimated. Its execution time is 1.3 times faster than 3×3 configuration. To see the accuracy of our estimation method, we have compared the actual execution time and the estimated execution time in the case of a DDM with flat 4-node, which can be configured with current DDM emulator with minimum modifications. The relative error was 3%. We also discuss about possible sources of errors in our method.

1. Introduction

The Data Diffusion Machine (DDM)[1] (also referred as Cache Only Memory Architecture (COMA) in Ref. 2)) is an instance of virtual shared memory architecture. In the DDM data has no fixed home location; instead it migrates from node to node when it is needed. Therefore when a *data item* (the unit of data storage and migration in the DDM) is not found in the processor's local memory, (which has a set-associative structure, and is called *Attraction Memory* in Ref. 2)), there has to be a functionality to locate the data. In the DDM we achieve this by a tree structure and directory nodes. **Figure 1** shows the structure of a 16-node DDM.

When a read miss to a data item occurs at the processing node, a *read* transaction is sent to the directory node above the processing node. If any processing node below the directory node has the item, the node sends back the item (by a *copy* transaction), but if none of the nodes below has the item, the directory node sends a *read* transaction to the directory one level higher. In the case of write, if the data item exists only in the processing node (i. e. the item is in the *exclusive* state), the write is achieved locally and immediately. If copies of the item exist in other nodes, an *erase* transaction is sent to each node having a copy of the item, and the write operation is suspended until each erase has been acknowledged in order to provide a strong consistency model.

A prototype DDM is being built at the Swedish Institute of Computer Science (SICS)[3]. This prototype DDM uses Motorola 88100 processers as leaf node processors and hierarchical buses for the interconnection. At the University of Bristol, a DDM emulator based on a network of Transputers has been implemented[4]. These two instances of the DDM have relatively small fanouts (the number of subsystems below the directory node at each level of hierarchy). These restrictions come from the physical limitation of the shared bus in the former case, and the number of communication links available in the current Transputer in the latter case. These factors lead to a large number of levels in the hierarchy of the DDM. In Ref. 6) it is pointed out that the disadvantage of COMA (DDM) is its higher miss penalty (compared with CC-NUMA) which comes from COMA's hierarchical structure. Recent semiconductor technologies make large crossbar switch devices, such as the Inmos C104[7], available. With these devices, it is possible to have large fanouts in the DDM, and hence the disadvantage of the higher miss penalty can be reduced. In this paper, we first describe configurations of the DDM based on an interconnection network using such switching devices. Then we will estimate how such configurations (with high fanout and split direc-

† Department of Computer Science, University of Bristol, U. K.

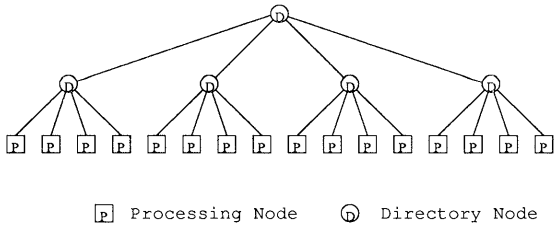


Fig. 1 Structure of 16 nodes DDM.

tory) will improve the performance of the DDM, by extending the statistical results obtained from the emulator. In Section 4, estimated execution time is compared with actual execution time to see the accuracy of our method. And then we discuss possible source of errors in our method.

2. Increase Fanout and Split Directory

Figure 2 shows how hierarchy levels are reduced, and how directory modules are dis tributed among leaf nodes in the case of converting 4×4 into 16 (flat) DDM.

Figure 2(a) is the original 2-level, 4×4 structure. First remove the level-2 (top) directories and collect all the level-1 directory entries into one node (Fig. 2(b)). Then split directory entries into 16 pieces using part of the item identifier. Note that all the leaf nodes may look up all the split directories. Therefore full 16×16 interconnection is needed (Fig. 2(c)). In Fig. 2(d), split directory nodes are embedded in leaf nodes, and they look physically flat. But we still consider the processing nodes are virtually placed under the directory nodes (we call this "*virtual hierarchy*").

Having high fanout allows a small scale DDM to have a one-level virtual hierarchy. However, as we will discuss in the section 5, we think a large scale DDM should have multi-level structure for practical reasons. **Figure 3** shows a block diagram of a leaf node in a virtual hierarchy DDM, which has a processor, a DDM memory, which has a set-associative structure, a part of directory entries, and a network interface which connects the node to the switching network. In this example the DDM system is assumed to have a two level structure, hence we have two directory modules. For simplicity, components in Fig. 3 are connected by a single bus, but this is not necessary. When necessary, a better interconnection scheme such as crossbar switch should be used to avoid traffic conflicts between components.

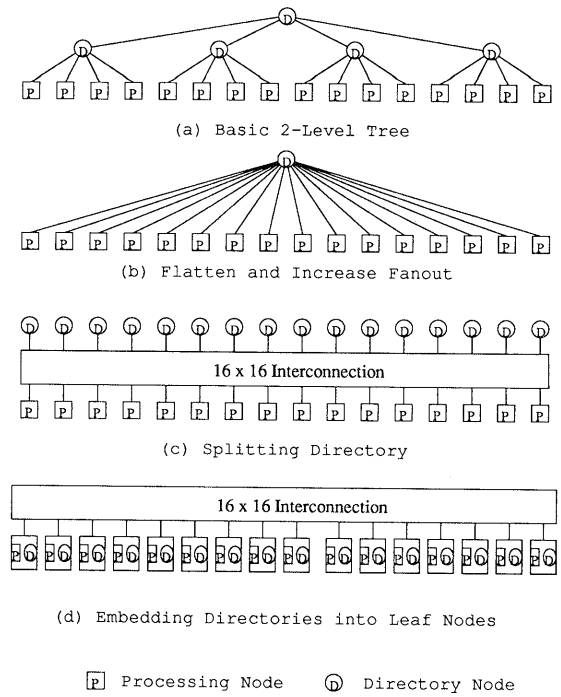


Fig. 2 Distributing directories.

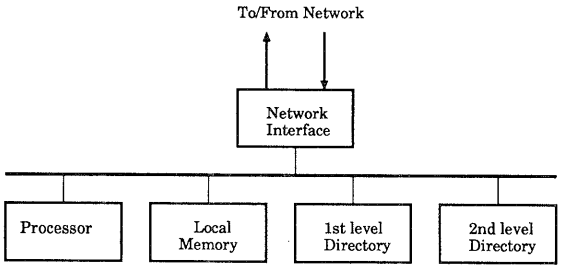


Fig. 3 A leaf node in virtual hierarchy DDM.

Adopting the virtual hierarchy into the DDM has both advantages and disadvantages. Since a leaf node has a part of the directory entries, look-ups to those entries can be achieved locally, hence it reduces network traffic and the average latency for look-up becomes shorter. This property of non-unique distance to directory module can be exploited for optimizing data placement, however some people may think one of DDM's policies is lost; although data items can still migrate among processors, directory entries have some fixed places to reside (In this sense a DDM with virtual hierarchy should be placed between original DDM and CC-NUMA). The network interface is shared by local memory and directory modules;

this can be both an advantage (cost reduction) and a disadvantage (contention of transactions destined to the different modules in a same node).

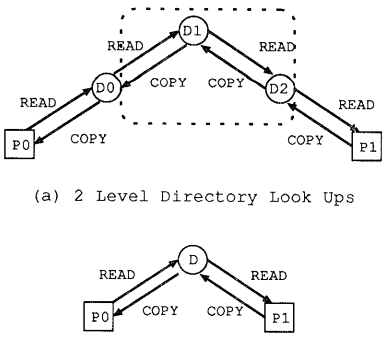
Assume that we have M nodes in the DDM system, and each node has memory for N data items. Then we need to keep track of the locations of at most $M \times N$ different data items in the system, that is, the directories have $M \times N$ entries for each level. We distribute these entries evenly to M nodes, resulting in each node having N entries. A part of the *Item Identifier* (which corresponds to the Physical Address in ordinary shared memory machines, minus k bits, where the item size is 2^k Bytes) is used for this directory entry distribution. In the case of 16×16 nodes DDM (256 nodes in total), each 16 leaf nodes shares one level-1 directory, and the entries of the directory with item identifier of which lower 4 bits are 0000 are stored in the node 0 in the group, 0001 are stored in the node 1, and so forth (For this distribution method, the number of nodes should be a power of two). The top directory (i. e. second level) is shared by all the 256 nodes in the system. Directory entries with item identifier whose lower 8 bits are 00000000 are stored in the node 0, and 11111111 are stored in the node 255, for example.

3. Performance Analysis

Although the experimental results obtained from the DDM emulator provide us accurate and reliable measures for the DDM performance[5], the configuration of the emulator is limited to having a fanout of three (except for four at the top node). It is possible to record all the transactions at all nodes in the emulator and analyze them with some program, but it would take huge computing resources to precisely analyze this data. In this section, we analyze the possible performance improvement in the DDM with higher fanout and split directories, by converting the statistical data obtained from the emulator. Its results could be less precise, but quick and does not require development of a software program to process transaction records.

The introduction of high fanout and a split directory scheme into the DDM will mainly affect the performance in the following points:

1. Reduction of the average miss penalty due to fewer directory look-ups.
2. Number of directory entries per node is reduced. This means each directory node received fewer transactions, which lead to less contention.



(a) 2 Level Directory Look Ups

(b) Single Level Directory Look Up

Fig. 4 Reduction of directory look-ups.

Table 1 Timing parameters for latency estimation.

Parameter	Description	Latency
T_H	Header Message	400 ns
T_F	Full Message	1040 ns
T_P	Protocol Transaction	200 ns

It is expected that increasing fanout may change the traffic pattern of the transactions in a DDM system. Also there exist certain types of contention which cannot be avoided by splitting directories. We assume these effects are negligible unless the fanout becomes too big. Below we investigate each factor independently, and then evaluate overall performance improvement by combining these factors.

3.1 Reduction of Directory Look-Ups

Increasing the fanout (i. e. number of subsystem under a directory node at each level) reduces the number of levels in the DDM hierarchy, and results in a reduction of directory look-ups.

Figure 4 shows the case where two level directory look-up (*DLU*) for read and copy transactions is reduced to one level. In Fig. 4 (a), a leaf node P0 sends a read transaction, and the transaction takes 4 hops to get to P1, the leaf node having a copy of the requested data item. Also the transaction needs to be processed at each directory node that the transaction passes. It is similar in the case where copy transaction is sent back from P1 to P0; however, since copy transaction contains the body of data item, it takes longer than read, which is made of only header (Full Message

and Header Message in the table 1 respectively). On the other hand, in the case of one level DLU, it takes two hops and one protocol processing each way. The area surrounded by dashed line in Fig. 4(a) shows these differences.

The **Table 1** shows the timing parameters used in the DDM emulator. Using these parameters, a DLU reduction from two levels to one level is converted to the time reduction of 3680ns in a miss penalty. In actual executions of application programs, transactions from different processors may be combined as they pass the directory nodes. Hence its calculation has to be done at each hop (e.g. in figure 4 from P0 to D0, from D0 to D1, etc). Another factor to be taken into consideration is that a read miss at a local memory could produce multiple read* transactions due to pre-fetching in the emulator, and a write miss at a local memory could produce multiple erase transactions (up to number of local memories minus one). However these multiple transactions are ideally processed simultaneously. We assume that the relation between reductions of directory look-up and execution time is proportional to the ratio (*Number of Misses at Local Memory*)/(*Number of Transaction Directly Generated by Local Memory Misses*). The following formula gives the benefit of fewer directory look-up in the execution time of application:

$$\Delta ExecTime_1 = \frac{((T_P+T_H)\times N_H + (T_P+T_F)\times N_F)\times Miss_{LM}}{Miss_{L1D}\times N_{Nodes}} \quad (1)$$

Where:

$\Delta ExecTime_1$	Reduction of Execution Time by Fewer DLU
N_H	Number of Level 2 Header Messages
N_F	Number of Level 2 Full Messages
$Miss_{LM}$	Number of Misses at Local Memory
$Miss_{L1D}$	Number of Messages Triggered by Misses at Local Memory
N_{Nodes}	Number of Leaf Nodes

3.2 Transaction Contention

Increasing fanout and splitting directory will change the traffic pattern of transactions in a DDM system. As stated earlier in this section, it reduces the number of transactions per directory node. When directory entries in a module are distributed among N modules, ideally the number of transactions per directory node is reduced to $1/N$. This reduction of transactions results in reduction of average latency at queues of directories.

From the execution of the DDM emulator, we can obtain the number of transactions received and the average latency at each level of directory. The average latency would be affected by the following factors in the DDM with large fanout and split directories:

Factor 1 Number of transactions per directory node is reduced to $1/N$, where N is the split factor, of the original hierarchy structure. Moreover if a directory node becomes top in the hierarchy, it will not receive transactions from above and again the number of transactions is reduced.

Factor 2 Lower average miss penalty caused by fewer DLU's makes average interval time between memory accesses shorter. This results in higher access rate and more contention, and causes longer average latency at directories.

We use the following symbols:

L_{Ave}	Average Latency (Original)
L'_{Ave}	Average Latency (Improved)
TR_{Total}	Number of Total Transactions
TR_{Above}	Number of Transactions from Above
T_{Org}	Original Execution Time
T_{DLU}	DLU Time Reduction**
N_{Split}	Split Factor
$\Delta ExecTime_2$	Reduction of Exec. Time by Less Contention

When a miss occurs at a leaf node, a transaction will be sent to the first-level directory above the leaf node. At the moment, if any outstanding transaction(s) is in the directory node, the transaction from the leaf node is not processed until the outstanding job is finished. Similarly, if several leaf node processors submit transactions at the same time, they are processed sequentially, and hence all but one are delayed. Since now we have the same number of first-level directories as processors, if no two transaction were destined to the same directory (i.e. transactions were completely distributed among the directories) there would be no delay. However there are some conflicts of transactions in actual applications; we

* Up to four reads in the experiments done in this paper.

** Miss penalty reduction due to fewer DLU in section 3.1.

assume the delay is linearly reduced as the number of transactions is reduces. The effect of this Factor 1 is expressed as:

$$Factor\ 1=\frac{TR_{Total}-TR_{Above}}{N_{Split}\times TR_{Total}} \quad (2)$$

Factor 2 above can be considered as a reduction of the average interval between transactions issued from a processor. There is uncertainty in estimating this effect precisely, i. e. if the delay at a queue is caused by transactions issued at the same time from different processors, Factor 2 does not make a big difference. On the other hand, if the delay is mainly caused by receiving new transactions at a directory which has outstanding transactions, the average latency is directly affected by Factor 2. Again we simply take the effect of Factor 2 as proportional to the total execution time:

$$Factor\ 2=\frac{T_{Org}}{T_{Org}-T_{DLU}} \quad (3)$$

Combining these two factors, the following equation gives the improved average latency:

$$L'_{Ave}=L_{Abe}\times Factor\ 1\times Factor\ 2$$
$$=\frac{L_{Ave}\times(TR_{Total}-TR_{Above})\times T_{Org}}{N_{Split}\times TR_{Total}\times(T_{Org}-T_{DLU})} \quad (4)$$

$$\Delta ExecTime_2=\frac{(L_{Ave}-L'_{Ave})\times Miss_{LM}}{N_{Nodes}} \quad (5)$$

3.3 Estimation Based on Emulation Results

In this subsection, we apply the formula presented in the previous subsections to the actual data obtained from the emulator and show the performance improvement. We chose *mp3d*, a fluid flow simulation from the SPLASH benchmark suites, as the application program to be run on the emulator, because in m3pd the miss penalty is dominant in its total execution time. In other words, in applications with low miss penalty (e.g. *Water* in SPLASH benchmark), having high fanout and split directory does not profit very much.

We estimate a case where 3×3 configuration is converted to 9, that is, the data obtained from the DDM emulator having 3×3 configuration is converted to an estimation of the performance of the one-level (flat) 9-node DDM. The execution time of 3×3 configuration time was 411ms. As explained in the section 2, actual implementations of split directory will have nodes of power of two to distribute entries using a part of item identifier (i.e. we find the node where the directory entry for the missed item exists by decoding some bits of item identifier). In the configuration used for performance estimation here, it is assumed that distribution of directory entries to non power of two nodes can be done without timing overhead due to hardware complexity.

Table 2 3×3 level 2 DLU transactions.

N_H	1.53×10^6
N_F	5.59×10^5
$Miss_{LM}$	4.64×10^5
$Miss_{L1D}$	9.38×10^5
N_{Nodes}	9
$\Delta ExecTime_1$	89 *ms*

Table 3 3×3 queue latency calculation.

L_{Ave}	279 ns
TR_{Total}	2.71×10^6
TR_{Above}	1.65×10^6
T_{Org}	412 ms
T_{DLU}	179 ms
N_{Split}	3
L'_{Ave}	64.3 ns
$Miss_{LM}$	4.64×10^5
N_{Nodes}	9
$\Delta ExecTime_2$	11 *ms*

Table 4 Execution time 3×3 and flat 9 configurations.

3×3 (Actual)	Flat 9 (Estimate)
412 *ms*	312 *ms*

3.3.1 DLU Reduction

Table 2 shows the total number of second level DLU's. With timing parameters in the Table 1, we estimated the time for DLU's will be reduced by 89ms.

3.3.2 Latency in Queue

Table 3 shows the data from the emulator and the estimation of the improved average latency obtained by applying the the formula 5.

In the case of flat 9-node DDM, its average latency at queues is reduced by 11ms.

Finally, we estimate the execution time of the DDM with flat 9-node configuration as 312ms, which is 1.3 times faster than the 3×3 configuration, and about 3 times faster than a single processor (**Table 4**).

4. Comparison

A Transputer T800, by which current DDM

emulator is implemented, has four communication links. Therefore we could implement a flat 4-node DDM with split directory scheme with minimum modifications (**Fig. 5** (a)).

We obtained the data from the execution of mp3d on a two level (2×2, Fig. 5(b)) DDM, applied the scheme in section 3 to those data, and compared its execution time with the case of a flat 4-node DDM (a).

The results of the DLU reduction and the queue latency reduction are shown in **Tables 5** and **6**, and the comparison between the estimated execution time and the execution time of a flat 4-node DDM is shown in **Table 7**.

We over-estimated the performance improvement by 3%. Although this seems to be acceptable level of inaccuracy, the possible sources of the error in our method, which would be the topics of the future study are:

- The modification of network configuration changed temporal characteristic of memory accesses in the application program (mp3d).
- The linear approximation of the queue latency reduction in section 3.2 was not appropriate, especially in small configurations, in which transaction contention might have been small.
- Increasing fanout caused hot spot contention.

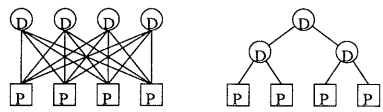


Fig. 5 Four nodes DDM's for comparison.

Table 5 2×2 level 2 DLU transactions.

N_H	1.03×10^6
N_F	3.48×10^5
$Miss_{LM}$	6.19×10^5
$Miss_{L1D}$	6.43×10^5
N_{Nodes}	4
$\Delta ExecTime_1$	253 *ms*

Table 6 2×2 queue latency calculation.

L_{Ave}	115 ns
TR_{Total}	1.88×10^6
TR_{Above}	6.88×10^5
T_{Org}	766 ms
T_{DLU}	263 ms
N_{Split}	2
L'_{Ave}	55 ns
$Miss_{LM}$	6.19×10^5
N_{Nodes}	4
$\Delta ExecTime_2$	9 *ms*

Table 7 Comparison between estimation and emulator result.

Two Level (2×2)	Flat 4 (Estimate)	Flat 4 (Actual)
766 ms	504 ms	520 ms

5. Limitation of Flat Configuration

In Ref. 6), COMA-F, a COMA which has lost hierarchy structure completely, is proposed. High fanout and split directory scheme makes a small scale DDM have one level structure, and it can be seen as an instance of COMA-F. However we think COMA-F cannot be adopted for constructing a large scale machine; rather we should have hierarchy level. For a COMA, we need a method to find a missed item. For this we use bitmap for the nodes having the item. If we build a flat 4096 nodes machine, a bitmap for an item has 4Kbits. To reduce false sharing we set item size relatively small (64Bytes) for the emulator; in this case the hardware overhead for a bitmap is 8 times larger than the data item itself. This seems to be impractical. On the other hand if the machine has three level hierarchy (16×16×16), the overhead reduces to 48 bits (less than 10% of the data item size).

Another reason that makes COMA-F impractical is that, in some applications a number of processors try to read an item at the same time. In the previous section we assumed that the load of directory modules are balanced well. However if a large number (say ten's or hundred's) of transactions are destined to a single directory node and they have no chance of combining due to a flat structure, the latency caused by this contention may be too large to be ignored. In the execution of the emulator we observed that in some applications the probability of combining is very high (e.g. in mp3d more than half of transactions are combined at each level of directory). The optimal values for fanout and hierarchy levels are affected by many factors, e.g. hit/miss ratio for applications, size and latency of the switching device, etc, and need further study.

6. Conclusions and Future Works

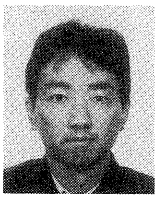
We explained the architecture of the DDM with high fanout and split directories. We can expect significant performance improvement of the DDM with this architecture, and estimated this improvement by extending the statistical data from the DDM emulator. The model used was less precise than the DDM emulator results; however by comparing our estimation with a flat DDM emulator with small number (four) of nodes we found its error was acceptable. We estimated the potential performance of the DDM. When a Transputer with C104 is available we will be able to increase fanout and split directories of the emulator, and then we can compare the estimation presented here with the result from the extended emulator. In the performance analysis, a possible delay due to contention of transactions (so called hot spot) was not taken into consideration. We will be working on the model including this effect to increase the accuracy of the analysis.

Acknowledgment The author would like to appreciate his colleagues, Henk Muller, Paul Stallard, and Jorge Buenabad, and his supervisor Professor David Warren for their comments and discussions on this paper. This work was partly supported by ESPRIT P7249 OMI/HORN.

References

1) Warren, D. H. D. and Haridi, S.: The Data Diffusion Machine—A Scalable Virtual Memory Multiprocessor, *Proc. 1988 Int. Conf. Fifth Generation Computer Systems*, pp. 943-952 (Dec. 1988).
2) Hagersten, E., Landin, A. and Haridi, S.: DDM—A Cache-Only Memory Architecture, *Computer*, pp. 44-54 (Sep. 1992).
3) Hagersten, E.: Toward Scalable Cache Only Memory Architecture, PhD Dissertation Swedish Institute of Computer Science (1992).
4) Muller, H. L., Stallard, P. W. A. and Warren, D. H. D.: An Evaluation Study of a Link-Based Data Diffusion Machine, *Proc. Int. Workshop on Support for Large Scale Shared Memory Architecture*, pp. 115-128 (Apr. 1994).
5) Muller, H. L., Stallard, P. W. A., Warren, D. H. D. and Raina, S.: Parallel Evaluation of a Parallel Architecture by Means of Calibrated Emulation, *Proc. 8th Int. Parallel Processing Symp.*, pp. 260-267 (Apr. 1994).
6) Stenström, P., Joe, T. and Gupta, A.: Comparative Performance Evaluation of Cache-Coherent NUMA and COMA Architecture, *Proc. 19th Annual Int. Symp. Computer Architecture*, pp. 80-91 (May 1992).
7) May, M. D., Thompson, P. W. and Welch, P. H. eds.: *Networks, Routers & Transputers*, IOS Press (1993).

(Received October 7, 1994)
(Accepted January 12, 1995)



Hitoshi Oi is a postgraduate student in computer science at University of Bristol, U. K. He received his bachelor's degree (in electronics and communication engineering) and master's degree (in electrical engineering) from Meiji University in 1985 and 1987, respectively. He worked for Digital Equipment Corporation Japan, ASURA project at ASTEM RI/KYOTO, and OMI/HORN project at Science Research Foundation PACT, U. K. His research interests include computer architecture, logic design and verification, and non-standard logic. He is a member of IPSJ.