Memory Characterization of a Parallel Data Mining Workload

Jin-Soo Kim[*], Xiaohan Qin, and Yarsun Hsu

IBM T. J. Watson Research Center
P. O. Box 218, Yorktown Heights, NY 10598
{jinsoo, xqin, hsu}@watson.ibm.com

Abstract

This paper studies a representative of an important class of emerging applications, a parallel data mining workload. The application, extracted from the IBM Intelligent Miner, identifies groups of records that are mathematically similar based on a neural network model called self-organizing map. *We examine and compare in details two implementations of the application: (1) temporal locality or working set sizes; (2) spatial locality and memory block utilization; (3) communication characteristics and scalability; and (4) TLB performance.*

First, we find that the working set hierarchy of the application is governed by two parameters, namely the size of an input record and the size of prototype array; it is independent of the number of input records. Second, the application shows good spatial locality, with the implementation optimized for sparse data sets having slightly worse spatial locality. Third, due to the batch update scheme, the application bears very low communication. Finally, a 2-way set associative TLB may result in severely skewed TLB performance in a multiprocessor environment caused by the large discrepancy in the amount of conflict misses. Increasing the set associativity is more effective in mitigating the problem than increasing the TLB size.

1 Introduction

The performance of microprocessors or systems is highly dependent upon the characteristics of applications for which the hardware supports. A number of studies have been performed to evaluate commercial as well as scientific applications [1, 2, 3, 4]. Recently, several classes of new applications, such as multimedia applications, web-based software, data warehousing and data mining, have emerged to gain market share rapidly. Understanding these new applications is crucial to successful design and optimization of future products. In this paper, we study a representative of an important class of emerging applications—data mining workloads.

Data mining is the process of extracting valid, previously unknown information from large databases to support critical business decisions [5, 6]. Its application and importance have been recognized in retail, marketing, banking, insurance, etc. For example, data mining can be used to classify "loyal" customers or to identify the set of people that are the most likely candidates to buy a new product. It can also be used by insurance companies to predict property or casualty losses for a given set of policy holders. It is expected that data mining will become one of the major applications in the near future.

The specific data mining workload we choose is a parallel implementation of self-organizing map (SOM) neural network model used in IBM Intelligent Miner [7]. Since the input data of the application often contain a large percentage of zero entries, the original SOM algorithm has also been optimized for sparse data sets. This paper examines the memory characteristics of the two SOM implementations (for dense and sparse data sets respectively). First, we study the temporal locality or working set sizes of the application and how they may be influenced by the application's parameters. Second, we measure the spatial locality by varying the cache block size and computing the memory block utilization. Third, we characterize the communication and scalability of the application with the increasing number of processors. Finally, we investigate the TLB performance in a 4-way SMP environment.

The rest of the paper is organized as follows. Section 2 describes the self-organizing map algorithm. Section 3 explains the methodology and application and simulation parameters. Section 4 presents workload characteristics. Finally, we conclude in Section 5.

[*] Jin-Soo Kim is a graduate student in department of computer engineering at Seoul National University, Korea, and was supported in part by the Korea Research Foundation.

2 A Parallel Data Mining: Self-Organizing Map

The self-organizing map (SOM) [8] is a neural network model used in IBM Intelligent Miner for database segmentation and clustering. The objective is to find records that share similar properties. An important application of this operation is in database marketing, which is to drive promotional campaign through the mining of corporate databases that record customer product preferences and public information about customer demographics and lifestyles. It is an area that data mining has been applied with singular success. Different from conventional neural network models that train the network against input data with known outputs, SOM identifies "clusters" of input records that have similar characteristics with no knowledge of their classes or outcomes *a priori*.

The input of the SOM algorithm is a set of records $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, each of which has F *fields* or *attributes*, i.e., $|\mathbf{x}_i| = F$. The main output of the algorithm is a set of prototypes organized as a 2-D lattice $W = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K\}$ ($K \ll N$ and $|\mathbf{x}_i| = |\mathbf{w}_k| = F$) and a classification of the input records based on the prototypes. Initially, the value of the prototypes are set to the first K input records. Then, for each input record, the algorithm iteratively computes its distance to every prototype, determines the closest prototype $\mathbf{w}_{c_i}$, and updates the fields of the prototypes.

There are two ways to update the prototypes: (1) online update (*online SOM*), where updates are performed after processing each record, and (2) batch update (*batch SOM*), where updates are performed at the end of each iteration (or *epoch*) by Eq.(1):

$$\begin{aligned}\mathbf{w}'_k &= \mathbf{w}_k + \frac{\sum_{i=1}^{N} h(c_i,k)\cdot[\mathbf{x}_i - \mathbf{w}_k]}{\sum_{i=1}^{N} h(c_i,k)} \\ &= \frac{\sum_{i=1}^{N} h(c_i,k)\cdot \mathbf{x}_i}{\sum_{i=1}^{N} h(c_i,k)} = \frac{\mathbf{T}_k}{B_k} \qquad (1)\end{aligned}$$

$h(c_i, k)$ is the neighboring function that controls the extent to which $\mathbf{w}_k$ is allowed to adjust in response to an input record based on the distance of $\mathbf{w}_k$ and $\mathbf{w}_{c_i}$ in the 2-D lattice. Figure 1 shows an outline of the batch SOM (BSOM) algorithm. Note that vector variables ($\mathbf{x}_i$, $\mathbf{w}_k$, and $\mathbf{T}_k$) are printed in bold face, while scalar variables (B_k, d_k, etc.) in italic in Eq.(1) and Figure 1.

A big advantage of the batch SOM over the online SOM is that the former facilitates the development of data-partitioned parallel methods. We use a parallel implementation of the BSOM algorithm developed by Lawrence *et al.* [9]. The input data of the parallel BSOM algorithm are partitioned across processors, while $\mathbf{T}_k$, B_k, and the prototype array $\mathbf{w}_k$ are shared by all processors. In addition, each processor p has private copies of $\mathbf{T}_k$ and B_k, namely $\mathbf{T}_k(p)$ and $B_k(p)$, to store partial summation calculated based on its own input records. At the end of each epoch, these values are collected and used to update the shared prototypes.

```
1:  Initialize $\mathbf{w}_k$ with $\mathbf{x}_k$ ($k = 1, \ldots, K$)
2:  for each epoch $e$ ($e = 1, \ldots, E$) do
3:    for each input record $\mathbf{x}_i$ ($i = 1, \ldots, N$) do
4:      for each prototype $\mathbf{w}_k$ ($k = 1, \ldots, K$) do
5:        Compute distance $d_k = \| \mathbf{x}_i - \mathbf{w}_k \|^2$
6:      end for
7:      Find the closest prototype $\mathbf{w}_{c_i}$ for $\mathbf{x}_i$
8:      for each prototype $\mathbf{w}_k$ ($k = 1, \ldots, K$) do
9:        Accumulate $B_k \leftarrow B_k + h(c_i, k)$,
                     $\mathbf{T}_k \leftarrow \mathbf{T}_k + h(c_i, k) \cdot \mathbf{x}_i$
10:     end for
11:   end for
12:   for each prototype $\mathbf{w}_k$ ($k = 1, \ldots, K$) do
13:     Update $\mathbf{w}'_k \leftarrow \mathbf{T}_k / B_k$
14:   end for
15: end for
```

Figure 1. Outline of the BSOM algorithm

Computing the distance (lines 4–6 in Figure 1) to find the closest prototype is one of the most time consuming computations in the BSOM algorithm. Since the input data of the application often contain a large fraction of zero entries [9], we can accelerate the distance computation by pre-computing $\sum_{f=1}^{F} {w_{kf}}^2$ (between line 2 and 3) and adjusting for the non-zero fields. We will compare and contrast the two implementations: the original (BSOM-D) and the optimized version (BSOM-S) for sparse data sets. For BSOM-S, only the non-zero fields are stored in memory.

3 Methodology

We use the front-end of an execution-driven simulation tool to generate memory reference traces. The simulation tool has been adapted from the Augmint toolkit [10] for the PowerPC architecture. To instrument the application, we modified BSOM-D and BSOM-S using M4 macros [11]. We select a set of values as the default application parameters. To analyze the working set sizes and their relations to the application parameters, we perform a series of experiments by changing one application parameter at a time. Table 1 displays the application parameters that have potential impact on the workload characteristics and their values, with the default values

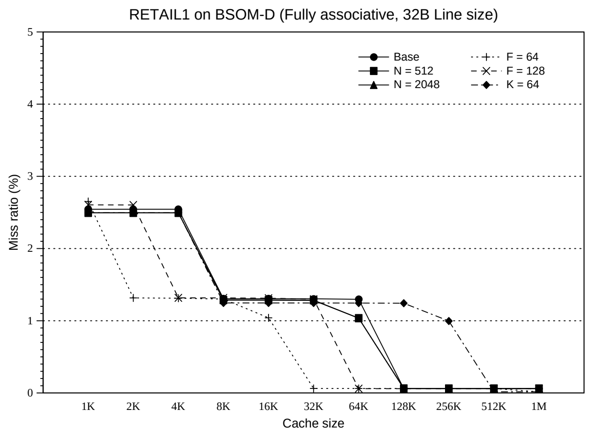
Table 1. Application parameters

parameters	values
Number of records (N)	512, **1024**, 2048, 4096
Number of fields (F)	64, 128, **272**
Number of prototypes (K)	**16**, 64

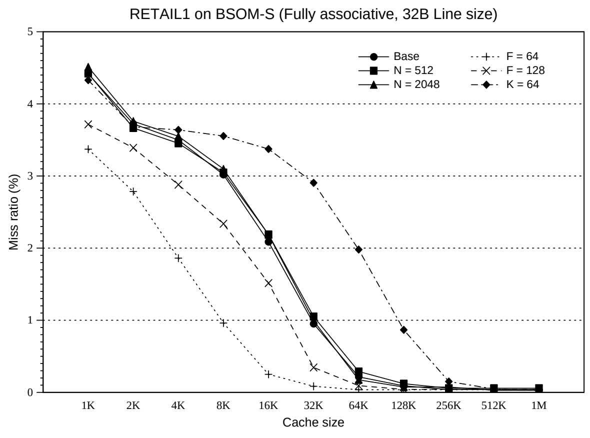
highlighted in bold face. Unless otherwise stated, the results presented in this paper are for the first two epochs.

To identify the working set size, we calculated the stack distance [12] for each memory reference by implementing a variation of stack simulation algorithm [13]. The stack distance corresponds to the number of distinct memory blocks referenced between two successive memory references to the same memory block. We use the stack simulation because it is capable of generating miss ratios for various sizes of fully-associative LRU caches in one pass. This method is particularly effective in the identification of working set sizes, where the cache sizes of interest are unknown. We consider cache sizes from 1KB to 1MB.

For the spatial locality, we measure cache miss ratios and average memory block utilizations under various block sizes, where the utilization of a memory block is defined as the percentage of the block being touched when it is selected for replacement. The block sizes we simulated range from 8B to 256B. To study the communication characteristics, we measure the communication volume per processor using 1 to 8 processors. Finally, TLB miss ratios are measured by simulating 2-way, 4-way, and fully-associative TLBs with either 128 or 256 entries. We assume that the page size is 4KB.



(a) BSOM-D



(b) BSOM-S

Figure 2. The working sets and the impact of application parameters

4 Workload Characteristics

This section presents four aspects of memory characteristics for the data mining application: (1) temporal locality or working set sizes; (2) spatial locality and memory block utilization; (3) communication characteristics and scalability; and (4) TLB performance.

4.1 Temporal locality and working set sizes

Figure 2 displays, for the dense and sparse implementations, the cache miss ratios of fully-associative LRU caches using the default and altered application parameters.

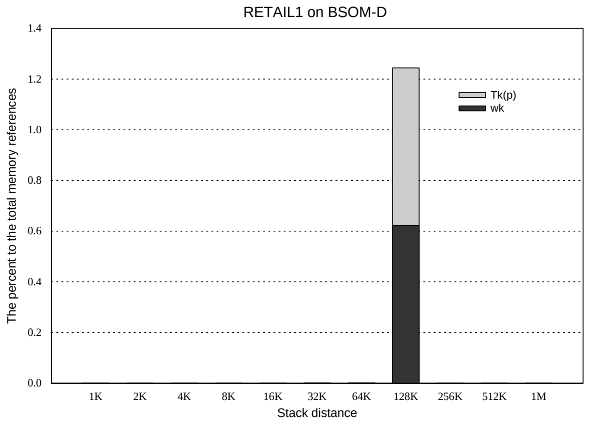
First we notice from Figure 2(a) that BSOM-D has two working sets at 8KB and 128KB for the default application parameters (the solid line). Recall that the innermost loops of the dense BSOM algorithm are computing the distance between one input record and the prototype array and accumulating partial summation (lines 4–6 and 8–10 in Figure 1), in which the input record is reused for each prototype. Therefore the first working set size hinges on the size of one input record which is proportional to the number of fields (F). As we reduce F to 128 (the dashed line) and 64 (the dotted line), i.e., 1/2 and 1/4 of the default value (272), the first working set size is reduced to 4KB and 2KB respectively.

The second working set size appears to be determined by the size of the prototype array $O(KF)$. Thus, when the number of prototypes (K) is quadrupled, the second working set size becomes four times as large (the dotted dash line in Figure 2(a)). And when the number of fields changes from 272 to 128 and 64, the second working set size is reduced to 1/2 and 1/4 of that for the default parameters. However, when we increase the number of input records (N), neither of the two working set sizes is affected. This is because all input records are referenced in each epoch. Unless the cache is big enough to hold the entire input records, a particular record will not find itself in the cache when it is reused in the next epoch.

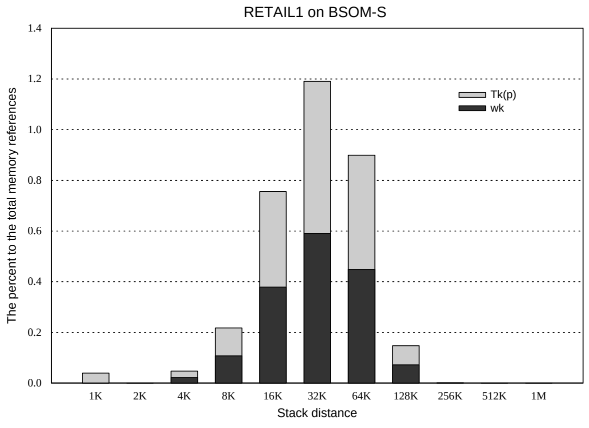
Now we analyze the results for BSOM-S. Figure 2(b) shows that BSOM-S has higher miss ratios than BSOM-D for caches smaller than 16KB. This is because cache lines are less efficiently used in BSOM-S due to its worse spatial locality (cf. Section 4.2). In contrast to BSOM-D, BSOM-S has only one well-defined working set size, which, similar to the second working set of BSOM-D, is also governed by the size of prototype array ($O(KF)$). Therefore, when varying the number of fields or the number of prototypes, the point at which the miss ratio curve turns flat shifts accordingly. The reason that BSOM-S does not have a clear-cut first working set size is because, in computing the distance, BSOM-S only accesses non-zero fields whose numbers vary from one record to another.

Notice that the working set size of BSOM-S is only one-half of the second working set size of BSOM-D. For example, using the default application parameters, BSOM-D has the second working set at 128KB, while the corresponding working set of BSOM-S occurs at 64KB. To identify the data structures that are responsible for this working set, we collect statistics on memory references directed to important data structures. BSOM-D and BSOM-S employ three major data structures whose sizes are all proportional to $O(KF)$: $\mathbf{w}_k$, $\mathbf{T}_k$, and $\mathbf{T}_k(p)$ (cf. Section 2). The shared data structure $\mathbf{T}_k$ is used only at the end of each epoch, thus its accesses do not contribute much to the overall miss ratio. The remaining two data structures are heavily used in the innermost loops, hence have the greater impact on the miss ratio. We plot the distribution of their stack distances (for the default application parameters) in Figure 3. Since we only care about cache sizes that are power of 2, we classify a reference with a stack distance in the range of $(2^{k-1}, 2^k]$ into the bin of 2^k. In the figure, the y-axis displays the percent of references of a certain stack distance to the total number of memory references.

Figure 3(a) shows that, in BSOM-D, all the memory references to $\mathbf{w}_k$ and $\mathbf{T}_k(p)$ have the stack distance of 128KB in the window of 1KB to 1MB. This is because every element in $\mathbf{w}_k$ and $\mathbf{T}_k(p)$ is referenced and they each are of 34KB. When the two data structures are reused to compute the distance for the next input record, more than 64KB but less than 128KB memory have been touched. In the case of BSOM-S, only the fields corresponding to the non-zero fields in the input record are accessed, and the non-zero fields vary among input records. As a result, the stack distances of the references to $\mathbf{w}_k$ and $\mathbf{T}_k(p)$ are distributed over the range of 4KB to 128KB, and most references (95.5%) have distances less than or equal to 64KB. Notice that the distribution of the stack distances of the references to the two data structures corresponds very well with the derivative of the overall miss ratio vs. log(cache size) for caches between 8KB and 1MB. In other words, references to $\mathbf{w}_k$ and $\mathbf{T}_k(p)$ are directly responsible for the changes in the miss ratio when the cache size increases in the range. They constitute the second working set.



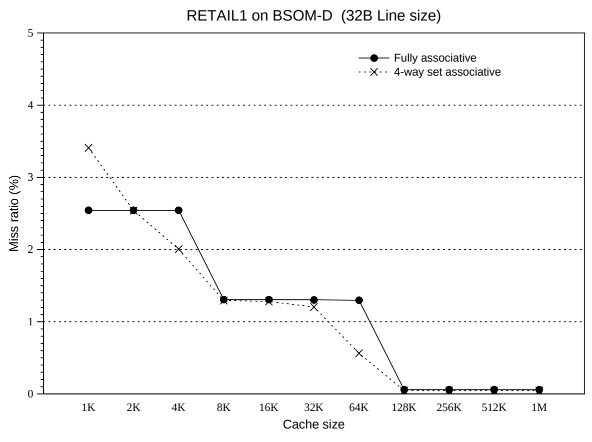
(a) BSOM-D



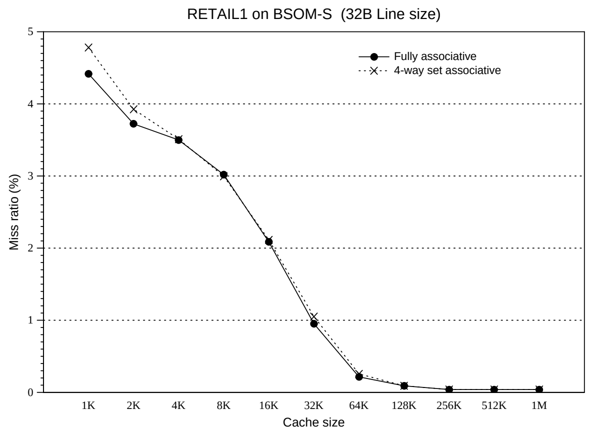
(b) BSOM-S

Figure 3. The distribution of the stack distances for $\mathbf{w}_k$ and $\mathbf{T}_k(p)$ using the default application parameters

To compare the stack simulation results with those of realistic caches, we simulated 4-way set associative caches using the default application parameters. Figure 4 shows the comparative cache miss ratios for BSOM-D and BSOM-S. When the cache size is very small ($<$ 4KB), conflict mapping results in higher miss ratios for the 4-way set associative cache. An interesting point



(a) BSOM-D



(b) BSOM-S

Figure 4. Comparison of miss ratios in fully and 4-way set associative caches

is that, in the dense BSOM implementation, the fully-associative cache performs worse at 4KB and 64KB, where the cache size is close to but smaller than the working set size. This is because the fully-associative cache constantly replaces the frequently-used cache lines due to the limited capacity, while the set associative cache divides the cache lines into sets and exercises the LRU policy for each set independently. Overall, the two sets of results match very well. For that reason, we only present the results of fully-associative caches in the remaining of the paper.

4.2 Spatial locality and memory block utilization

Figure 5 depicts the miss ratios for different cache block sizes and different cache sizes. An application that has perfect spatial locality reduces the miss ratio by half when doubling the cache block size due to prefetching effects. From Figure 5, we can see that BSOM-D has almost perfect spatial locality, while BSOM-S has slightly worse spatial locality mainly because of its irregular memory access pattern in the innermost loops. In order to measure the spatial locality quantitatively, we define *average memory block utilization* U as follows:

$$U = \frac{1}{M} \sum_{m=1}^{M} \frac{1}{n(m)} \sum_{i=1}^{n(m)} \frac{u(m,i)}{B} \qquad (2)$$

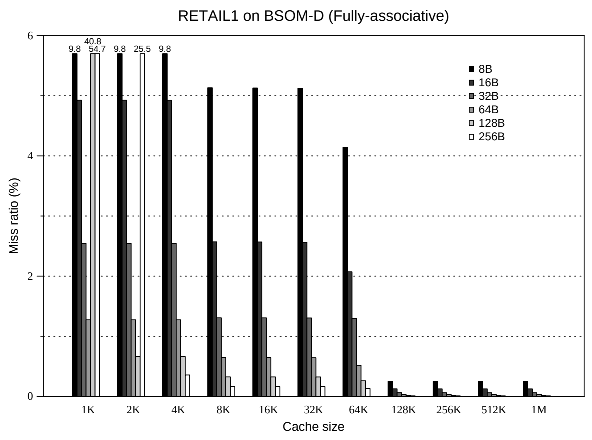
where B and M denote the cache block size and the total number of memory blocks touched by the application respectively. $u(m,i)$ is the number of bytes touched in the memory block m since the block is brought in the cache the i-th time until it is expelled from the cache. $n(m)$ is the number of cache misses due to the block m, i.e., the number of times the block is transferred from memory to the cache.

We plot the average memory block utilization for four cache sizes (8KB to 64KB) in Figure 6. It shows that BSOM-D has almost 100% memory block utilization, while the corresponding number for BSOM-S is slightly lower. The very good memory block utilization is due to the sequential access pattern in the shared-memory region, which accounts for a large portion of memory blocks used by the application (98.3% for BSOM-D and 83.7% for BSOM-S).

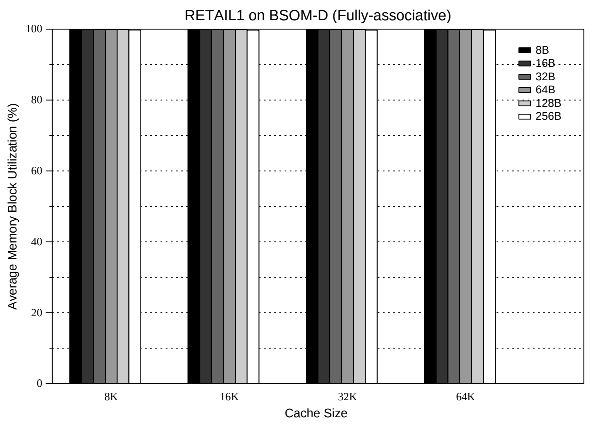
The benefit of using larger cache block sizes not only depends on that the working set has good spatial locality, also the cache needs to be large enough to hold application's working set. If the cache size is too small ($<$ 4KB), a larger cache block may result in high capacity misses (shown in Figure 5(a) and (b)). When the cache is at least 4KB, the miss ratios for BSOM-D monotonically decreases as the block size doubles (cf. Figure 5(a)). On the other hand, BSOM-S still suffers from lacking of sufficient cache blocks up to 32KB. As shown in Figure 6(b), for a given block size, increasing the cache size brings a slight improvement in the memory block utilization for 8KB to 32KB caches. However, once the cache reaches 64KB, the capacity to hold the critical working set (cf. Figure 2(b)), the memory blocks stay in the cache long enough to benefit from the spatial locality, hence, the block utilization is increased significantly.

4.3 Communication and Scalability

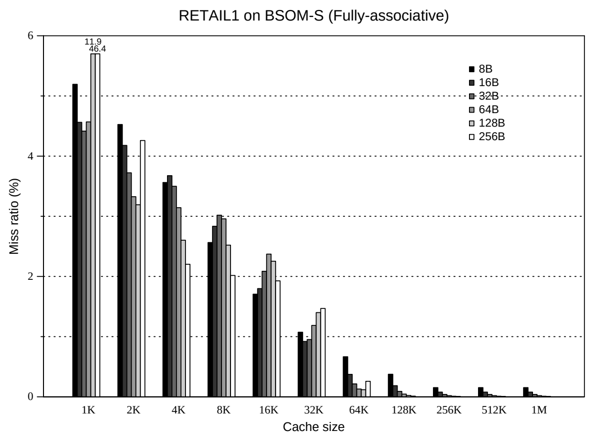
In this section, we consider the communication characteristics as we increase the number of processors. We assume that the problem size or the number of input data records will increase at the same rate as the number of



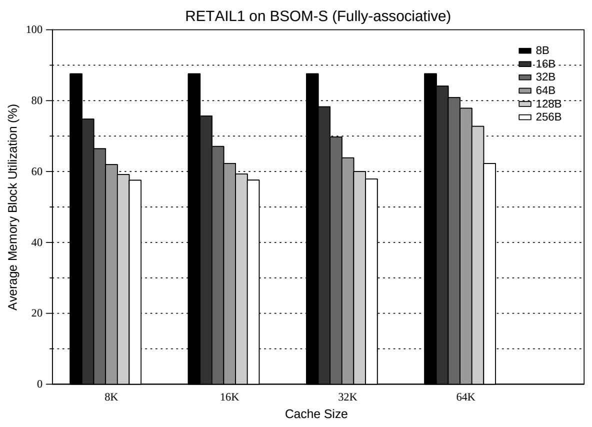
(a) BSOM-D



(a) BSOM-D



(b) BSOM-S



(b) BSOM-S

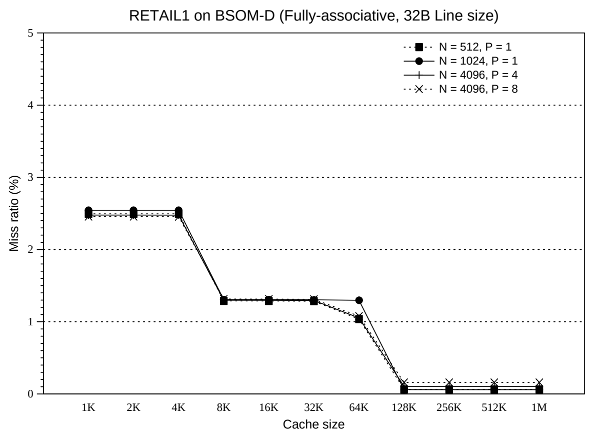
Figure 5. The impact of cache block size on miss ratios for the default application parameters

Figure 6. The average memory block utilizations for the default application parameters

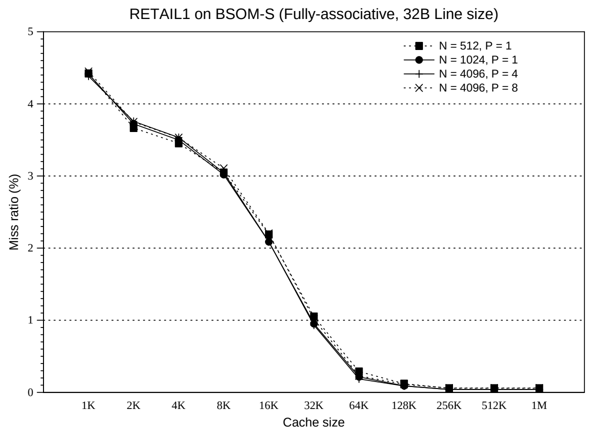
processor increases. In other words, the data assigned to each processor remain the same. In section 4.1, we have observed that the miss ratios and the working set sizes are independent of the number of input records. Therefore, increasing the number of processors has little impact in those regards (cf. Figure 7).

We now examine the communication volume as a function of the number of processors. We define the communication volume by the average amount of data (cache blocks) exchanged between one processor and the other processors. Figure 8 illustrates the communication volume normalized to a 2-processor system. In the 2-processor system, we observe that the communication accounts for only 2.3% of the misses for caches larger than 128KB. The very low communication volume can be attributed to the batch update scheme. As the number of processor increases, the communication volume monotonically increases, however, the amount quickly converges.

In the BSOM algorithm, communication occurs at the end of each epoch when processors update the prototype array in parallel and when processors read the updated prototypes for the distance computation. Each processor is responsible for updating a portion, i.e., $1/P$, of the prototype array using the partial summation computed by itself and the other processors (cf. Eq.(1)). Therefore, the amount of communication incurred by each processor is $O(\frac{P-1}{P})$, which is bounded by some constant value no matter how many processors are used. The same communication scheme is employed in the two BSOM implementations, which explains why they have the similar communication characteristics.



(a) BSOM-D



(b) BSOM-S

Figure 7. The impact of the number of processors on the working sets

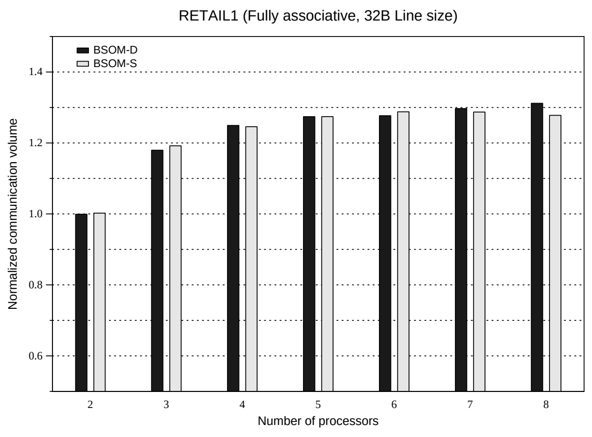


Figure 8. The normalized communication volume for the default application parameters

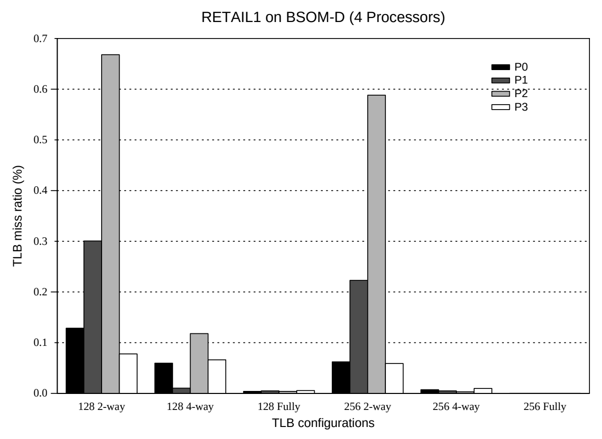
4.4 TLB Performance

The performance of TLB is crucial to the overall system performance especially if the L1 cache is physically addressed since the TLB access would be on the critical path of every memory reference. In this section, we examine the TLB behavior of BSOM-D and BSOM-S running under a 4-way SMP environment. We consider six TLB configurations by varying the number of TLB entries from 128 to 256 and by varying the associativity from 2-way to 4-way and fully-associative. As in the previous experiments, we also vary the application parameters. However, only when $F = 272$ and $K = 64$, the applications yield considerable TLB misses. Therefore, we present and discuss the results using the above application parameters. Figure 9 shows the TLB miss ratios for the six TLB configurations. In each configuration, the four bars represent the miss ratios of the four processors.

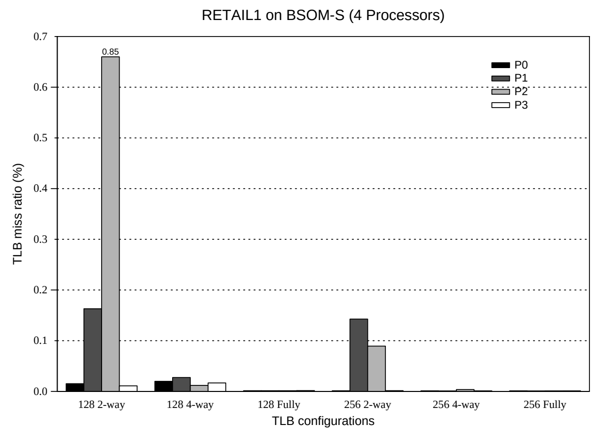
When a fully associative TLB is used, the miss ratios are very small or negligible regardless of the TLB size. It appears that the TLB performance benefits more from increasing the set associativity than from increasing the TLB size. For instance, when a 2-way set associative TLB is employed, increasing the number of entries from 128 to 256 reduces the TLB miss ratio for BSOM-D slightly, while employing a 4-way set associative TLB with 128 entries, the TLB miss ratio decreases significantly.

The most surprising result is that the TLB miss ratios of the four processors differ substantially in a number of configurations. For example, when using the 2-way 128-entry TLB, $P2$ has 8.6 times more TLB miss ratios than $P3$ in the case of BSOM-D. For BSOM-S, it is even worse that $P2$ has 77.6 times more misses than $P3$ does. The discrepancy in the TLB performance among the four processors introduces severe skew in what was perfectly balanced (per processor) workload. Since all processors have to barrier-synchronize to update the prototypes, the uneven TLB performance is propagated and eventually limits the overall speedup.

To understand what causes the significant difference in the TLB performance, we collect detailed statistics on the TLB entries. Figure 10 displays the number of replacements occurred in each set of the 2-way 128-entry TLB for $P2$ and $P3$ (the processors that have the highest and lowest miss ratios respectively) for BSOM-D. The picture for BSOM-S looks similar. In the figure, y-axis is displayed in log scale. The replacements have been classified into two groups: *shared* for replacements due to

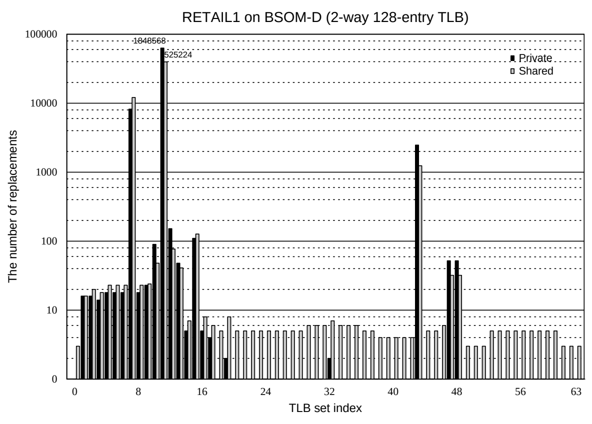


(a) BSOM-D

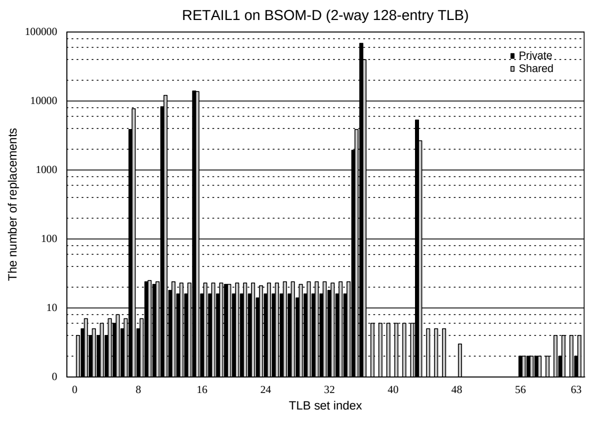


(b) BSOM-S

Figure 9. TLB miss ratios for various TLB configurations (4 processors, K=64)



(a) $P2$



(b) $P3$

Figure 10. The number of replacements in 2-way 128-entry TLB for BSOM-D (4 processors, K=64)

shared references (*shared pages*) and *private* for replacements due to private references either to stack variables (*local pages*) or to static variables and dynamically allocated structures (*global pages*). Usually, (local) stack variables consume a very small amount of memory. The majority of the memory pages are either shared or global pages.

We observe that the TLB misses are distributed unevenly among the sets. In $P2$, the number of replacements in one set (set 11) accounts for 99% of the total TLB misses of that processor. In-depth analysis reveals that, on $P2$, there are 1 local page, 3 global pages, and 4 shared pages simultaneously mapped into the set 11. On $P3$, however, the set that incurs the highest replacement has only 2 global pages and 4 shared pages. The plausible cause for the difference is that the application employs the thread model in which threads share the same virtual address space. Although the addresses of the shared data are the same on the four processors, the addresses of the private data (local and global pages) will be different. If a processor happens to map a local page and several global and shared pages into the same set, its TLB is likely to suffer from a large number of conflict misses. This is the case with $P2$. Since most of the TLB misses are caused by the conflicting mapping, increasing the set associativity eases the problem significantly.

5 Concluding Remarks

This paper studies the memory characteristics for a representative of an important class of emerging applications—data mining workloads. The particular workload, chosen from the IBM Intelligent Miner, implemented the self-organizing map neural network model in parallel, and employed batch update to minimize the communication. The core algorithm has been implemented in two versions with one suitable for dense data sets and the other optimized for sparse data sets.

We examine and compare in details four characteristics of two implementations of the BSOM algorithm: (1) temporal locality, or more specifically, working set sizes and their relations to the application's parameters; (2) spatial locality and memory block utilization; (3) communication characteristics and scalability with varying number of processors; and (4) TLB performance.

First, we have found the working set hierarchies of BSOM-D and BSOM-S are governed by the size of an input record and the size of prototype array. Neither of the working sets is sensitive to the number of input records. Second, BSOM-D has almost 100% of memory block utilization, i.e., good spatial locality, while BSOM-S has slightly worse spatial locality. Third, the data mining application appears to have very low communication. The amount is bounded by some constant value no matter how many processors are used. Finally, the 2-way set associative TLB can result in skewed TLB miss ratios in a multiprocessor environment and increasing the set associativity is more effective than increasing the TLB size in the improvement of the TLB performance.

Due to the diversity and complexity of data warehouses, a single data mining technique is not sufficient to reveal all the relationships among the data. It is thus necessary that several data mining techniques are available to the business and data analysis. We plan to investigate other data mining techniques to fully understand the behavior of this class of applications.

6 Acknowledgments

The authors would like to thank George Almasi and Richard Lawrence in IBM T. J. Watson Research Center for providing the data mining application studied in this paper.

References

[1] A. M. G. Maynard, C. M. Donnelly, and B. R. Olszewski, "Contrasting Characteristics and Cache Performance of Technical and Multi-User Commercial Workloads," in *Proc. of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 145–156, 1994.

[2] L. A. Barroso, K. Gharachorloo, and E. Bugnion, "Memory System Characterization of Commercial Workloads," in *Proc. of the 25th International Symposium on Computer Architecture*, pp. 3–14, 1998.

[3] D. C. Lee, P. J. Crowley, and J.-L. Baer, "Execution Characteristics of Desktop Applications on Windows NT," in *Proc. of the 25th International Symposium on Computer Architecture*, pp. 27–38, 1998.

[4] S. C. Woo *et al.*, "The SPLASH-2 Programs: Characterization and Methodological Considerations," in *Proc. of the 22nd International Symposium on Computer Architecture*, pp. 24–36, 1995.

[5] IBM, "IBM's Data Mining Technology." White Paper, available at http://www.software.ibm.com/data/iminer/, 1996.

[6] D. S. Tkach, "Information Mining with the IBM Intelligent Miner Family." An IBM Software Solutions White Paper, available at http://www.software.ibm.com/data/iminer/, 1998.

[7] IBM, "Intelligent Miner Family." http://www.software.ibm.com/data/iminer/, 1998.

[8] T. Kohonen, "The Self-Organizing Map," *Proc. of the IEEE*, vol. 78, pp. 1464–1480, Sep. 1990.

[9] R. D. Lawrence, G. S. Almasi, and H. E. Rushmeier, "A Scalable Parallel Algorithm for Self-Organizing Maps with Applications to Sparse Data Mining Problems." Submitted for publication, 1997.

[10] A.-T. Nguyen, M. Michael, A. Sharma, and J. Torrellas, "The Augmint Multiprocessor Simulation Toolkit for Intel x86 Architectures," in *Proc. of 1996 International Conference on Computer Design*, 1996.

[11] J. Boyle *et al.*, *Portable Programs for Parallel Processors*. Holt, Rinehart and Winston, Inc., 1987.

[12] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation Techniques for Storage Hierarchies," *IBM Systems Journal*, vol. 9, pp. 78–117, Feb. 1970.

[13] Y. H. Kim, M. D. Hill, and D. A. Wood, "Implementing Stack Simulation for Highly-Associative Memories," in *Proc. of 1991 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, pp. 212–213, 1991.