

Efficient Data Packet Compression for Cache Coherent Multiprocessor Systems

Baik Song An*, Manhee Lee[†], Ki Hwan Yum* and Eun Jung Kim*

**Department of Computer Science and Engineering*

Texas A&M University, College Station, Texas 77843-3112

Email: {baiksong,yum,ejkim}@cse.tamu.edu

[†]National Security Research Institute, Daejeon, Republic of Korea

Email: manheelee@ensec.re.kr

Abstract: Multiprocessor systems have been popular for their high performance not only for server markets but also for computing environments for general users. With the increased software complexity, networking overheads in multiprocessor systems are becoming one of the most influential factors in overall system performance. In this paper, we attempt to reduce communication overheads through a data packet compression technique integrating a cache coherence protocol. Here we propose Variable Size Compression (VSC) scheme that compresses or completely eliminates data packets while harmonizing with existing cache coherence protocols. Simulation results show approximately 23% of improvement on average in terms of overall system performance when compared with the most recent compression scheme. VSC also improves performance by 20% on average in terms of cache miss latency.

I. INTRODUCTION

With the current trend in information technology, multiprocessor systems have been widely adopted to achieve better performance in diverse computing environments. Relying on traditional methods such as instruction-level parallelism (ILP) in uniprocessor environments has a limitation on performance improvement. Most software in recent days adopts multithread program structures, which were mainly used for high-performance computing environments exclusively. Under a high degree of parallelism, networking performance is getting more dominant than computation power in determining overall system performance. Since a traditional approach using shared buses is not a scalable solution, switch-based interconnection networks are regarded as a promising alternative in large-scale multiprocessor systems. Providing an effective solution to reduce communication overheads has been one of the major goals in the multiprocessor system design.

There has been plenty of research carried out to diminish overheads of interconnection networks in multiprocessor environments. This goal can be achieved in two ways: adopting a faster network with shorter latency and wider bandwidth, or decreasing the amount of network workloads. Some studies focus on the latter through data compression to reduce data packet overheads. Data compression used in memories and interconnects of multiprocessor systems helps to decrease the size of cache block data that is transferred in the form of data packets through the network. Recently, a static data compression scheme [1] has been proposed to compresses data based on predefined fixed patterns, which fails to exploit dynamic communication behavior. To overcome this drawback, an adaptive compression scheme [2] uses tables containing frequently used patterns and updates them dynamically. However, both schemes simply reduce the packet size through compression and never attempt to incorporate the compression with underlying cache coherence protocols, which limits the amount of performance improvement at a certain level.

This research tries to take a step forward to overcoming the limitation. Here we attempt to reduce communication overheads through data packet compression that is aware of an

intrinsic cache coherence mechanism. In the best case, the compression scheme enables data packets to be completely eliminated, which contributes to simplifying the miss handling procedure in coherent caches and reduces cache miss latency significantly. Data packets can be eliminated if the data pattern is known to the requestor without having to actually send and receive packets. It must be guaranteed that eliminated packets do not make any conflicts or errors with the existing cache coherence protocols. Also, the compression scheme used in multiprocessor systems must show good performance in handling a cache block whose size is small, not larger than 64 bytes in general.

We propose Variable Size Compression (VSC) scheme that is scalable and efficiently eliminates or compresses data packets. Packet elimination in VSC is done by managing matching status bits in the directory that denote whether each cache block data matches the most frequent pattern in the system. A reply message from the directory to the requestor carries the bit to notify that the requestor does not need to wait for a data message from the sender if the bit is set. If VSC fails to eliminate data packets, it divides the cache block into multiple units for compression. Unlike the existing schemes, VSC provides data compression with various unit sizes in parallel for more efficient compression. Simulation results show that the proposed scheme can reduce the message size by 54% on average and the overall performance by 23%, compared with the most recent compression scheme for interconnects proposed in [2].

The remainder of this paper is organized as follows. We discuss related work in Section II. In Section III, we explain the data compression technique in detail. Section IV presents simulation results and analysis, and finally Section V summarizes our work and makes conclusions.

II. RELATED WORK

Various data compression techniques have been introduced to improve performance by increasing the capacity or reducing the latency [3], [4], [5], [6], [7], [1], [8], [2], [9]. Among them, a large number of hardware-based compression schemes implement dictionary-based compression algorithms [8], [6]. Dictionary-based schemes depend on building a dictionary which contains data words appearing in a message, and use the entries to encode repeated words. Significance-based schemes compress data by removing redundant information such as the high-order portion of address values or sign extension bits [3], [5]. Frequency-based mechanisms are based on the observation that small sets of values are found in load/store operations more frequently than other values [7], [9].

Data compression mechanisms have been adopted in diverse system domains. Cache or memory compression makes better use of the limited memory by compressing and storing data in a smaller space so that two or more compressed blocks can fit in a single block space [3], [8], [6]. Buses also can be utilized more efficiently through compression. Bus-Expander is proposed in [4], where frequent values of the most significant bits of the bus are stored in a table and used to efficiently cut the size of data in half. Even in on-chip interconnects, compressed messages can improve network utilization and system performance by reducing packet latency [1], [2].

III. VARIABLE SIZE COMPRESSION (VSC)

In this section, we explain how to eliminate packets and to compress and decompress data values in our framework. We propose a new compression scheme, which is called

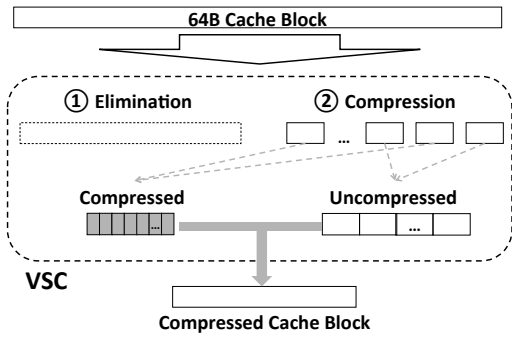


Figure 1: Compression procedure of a cache block

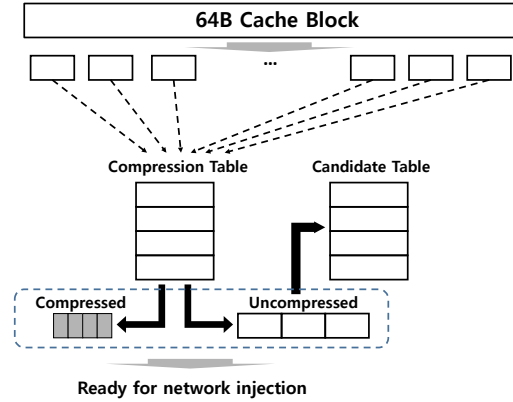


Figure 2: Compression and candidate tables

Variable Size Compression (VSC). Figure 1 summarizes the compression procedure of our scheme. With VSC, a cache block can be eliminated or divided into multiple small units to be compressed. After the compression, there can be compressed and uncompressed units, which will be explained in the following section. We assume that only data messages are compressed.

A. Efficient Data Compression Utilizing Variable Size Pattern Frequency

VSC uses two types of tables, compression and candidate tables, to compress or eliminate a cache block. Compression tables used in VSC contain most frequent data patterns in the network data traffic. After a cache block is divided into multiple compression units, each compression unit is compared to each entry in the compression table. If it matches one of the entries in the compression table, it is encoded into its corresponding index. If no entry is a hit, the unit is transferred uncompressed. Each entry needs to count the number of hits to be used in the table update. In this manner, compression units become ready to be encrypted either in a compressed or uncompressed form. After the packet arrives at the receiver and is decrypted, compressed units are converted into original units by looking up the compression table at the receiver's side. Uncompressed units are directly delivered as they are. All nodes have the same contents in the compression tables so updating the compression tables is performed regularly in a centralized manner reflecting dynamic behavior of data patterns.

Candidate tables contain the most frequent values among uncompressed units and they are managed and updated dynamically and independently per node. Figure 2 describes how compression and candidate tables are used to compress cache block data. After compression, each uncompressed unit is compared to candidate table entries. If it matches one of the entries, the corresponding hit count of the entry increases. If not, one entry is evicted and replaced with the uncompressed unit data based on the Least Frequently Used (LFU) policy. Since accessing candidate tables is done after compression, the access latency is not in the critical path.

Updating compression tables is done as follows. First, hit counts of compression table entries and values/hit counts of candidate table entries in all nodes are gathered. Then hit counts of compression table entries for all nodes are summed up and all candidate table

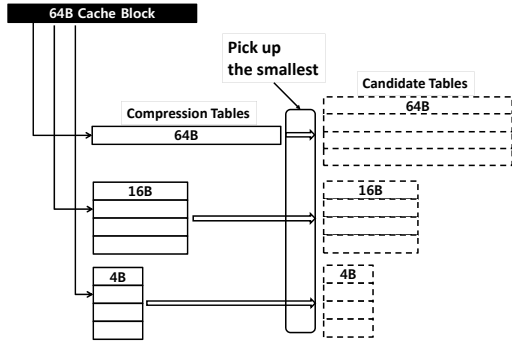


Figure 3: Data compression with various unit sizes

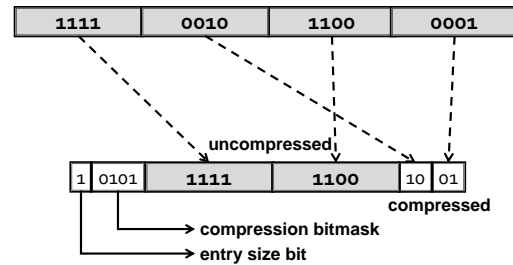


Figure 4: Compressed data block structure

entries are merged and sorted in the descending order of total hit counts. Hit counts of the most highly ranked entries in the merged table are compared to those of compression table entries. If the hit count of a merged table entry is bigger than any of compression table entries' hit counts, the compression table entry is evicted and updated with the merged table entry. Finally, all hit counts of a newly updated compression table are initialized to zero to prevent aging effect and then the new compression table is broadcast to all nodes. Note that the update procedure is off the critical path, meaning that it can be done without affecting cache miss handling.

But in reality, the broadcast messages cannot reach all nodes at the same time due to the variation of packet transmission time. So there may be a synchronization issue between the two adjacent periods using different pattern sets, meaning that some nodes already receive the new pattern set while others do not. Here we handle this issue by using one-bit information, which is called a period bit (PB). A PB differentiates two adjacent periods by flipping the bit every time a new pattern set is ready. For example, if current period is represented by zero, the next period using a new pattern set can be represented by one. Each node maintains its own PB, updating the bit when it gets a new pattern set. Since the current period and the period after next are represented by the same bit value, some might think we should consider how to make difference between the two. However, it is a reasonable assumption that each period is long enough so that we do not need to be concerned about the issue. Also the amount of data traffic during the time when two pattern sets are used mixedly is quite minimal, which will be explained in detail in Section IV. Every time a cache coherence packet is injected into the network, the packet carries a sender's PB to let a receiver know the sender's status. Suppose that a sender of a cache block already got a new pattern set while a receiver did not. Then the sender must not use the new pattern for compression because the receiver cannot restore compressed data to its original one.

To minimize the packet size, we investigate how VSC can be applied, mostly the number of entries, each entry size, and the new cache-to-cache coherence packet structure. We ran 49 combinations of simulations with seven different entry sizes and seven different numbers of compression table entries in 8 and 16-processor systems. The results show that a four byte-long entry provides a very good compression ratio only with four entries in a 64-byte cache block architecture. Since the entry size is four bytes and the index is

just two bits long, assuming four entries per table, the unit size is reduced by a factor of 16. However, it would be beneficial to achieve a better compression rate if we try different unit sizes in every data compression and choose the best one. It is reasonable because VSC is free from the scalability issue, and adopting extra compression and candidate tables with different entry sizes is not problematic. So we use three different compression entry sizes: 4-byte, 16-byte and 64-byte as shown in Figure 3. For 4-byte and 16-byte, both compression and candidate tables have four entries. But, in the case of 64-byte that is the same as a cache block size, a compression table has only one entry, even though a candidate table has four entries as in the other two cases. The reason is that we do not need to use even index values once the cache block is compressed with 64-byte unit size, which means data could be completely eliminated. We will make a more detailed description of how it works in the next section. Also, each node maintains two sets of compression tables to handle two different pattern sets used in the adjacent periods.

To integrate VSC into a 64-byte block system, we divide a 64-byte block into a number of compression units, and compare each unit against the compression table. Then, each data packet should have additional information about which units are compressed. Figure 4 describes the structure of a compressed data packet, which depicts four 16-byte units belonging to one cache block. The content in each unit is displayed as a 4-bit number for simplicity, even though it should be actually 16 bytes long. An entry size bit that indicate the unit size used for compression is located at the head. If the bit is set, it means a 16-byte unit size is used. If it is clear, a 4-byte unit size is used. We do not have to adopt an extra bit for 64-byte unit size since a data packet itself could be completely eliminated and does not need to be sent. Note that all units in a cache block are compressed with the same unit size denoted by the entry size bit that shows the best compression rate. Then it is followed by a compression bitmask, which indicates whether each unit is compressed. The compression bitmask will be used by the receiver to restore the original data packet. Uncompressed and compressed units are located in order at the tail. In Figure 4, the receiver restores the original data block by getting the first and third uncompressed units from the packet and reading the second and fourth units from the compression table.

Since we divide one data block into multiple units, it takes much time to process each compression unit sequentially. Therefore, we adopt a number of duplicated compression tables to compress/decompress multiple units in parallel for each compression unit size. Latencies are overlapped through parallel compressions using duplicated tables. Note that we do not need to duplicate candidate tables because accessing candidate tables could be done off the critical path.

B. Selective Update in VSC

As explained in the previous section, updating the compression tables is performed in a centralized way. However, it may cause the scalability problem as the system size grows because all the information of compression and candidate tables must be collected from all nodes regularly to update the compression tables. It is already observed and well-known that the communication behavior in the interconnects of multiprocessor systems exhibits the high level of temporal locality. Also it is verified that only a few pattern dominates the total data traffic in overall [3], [1]. Therefore, we gather the information

not from all nodes, but from some selected nodes that aggressively communicate with other nodes. The communication pattern between nodes looks asymmetric in many cases, meaning that some nodes send more packets than received packets and other nodes receive more packets than sent packets. Since each data packet is shown to both a sender and a receiver, it is a good way of reducing overheads to select only one of the two groups, aggressive senders or aggressive receivers. Here we get the information from aggressive senders transmitting data packets more than receiving them.

Selective update in VSC can be performed as follows. Each node i can be chosen to provide the information for update if

$$s_i > k \cdot r_i,$$

where s_i and r_i are the numbers of sent and received packets in node i , respectively. k is a constant for adjusting how selective VSC should be in choosing the nodes. As k becomes bigger, VSC becomes more selective and the number of chosen nodes decreases. It can reduce the update overheads, but it may worsen the compression ratio since the amount of gathered information also decreases. We will show a detailed analysis of the selective update in Section IV-B.

C. Data Packet Elimination Using Temporal Locality

While compression facilitates to reduce the data packet size, we can further eliminate data packets exploiting the frequent data patterns. Data packets can be completely eliminated using the compression table with only one 64-byte entry. Here we explain in detail how it is done. In previous section, we already described how the 64-byte compression table entry is shared by all nodes and how it is updated. We use extra one-bit information per cache block to denote whether each up-to-date cache block data matches the current table entry or not and it is assumed to be stored in a directory. We call it matching status bit (MSB) hereafter. In the case of fetching from disks, the memory becomes the owner. Since the directory is usually located along with memory, it is not a problem to keep track of most recent patterns for data blocks. However, if a store operation occurs, the processor becomes the owner. The processor compares the previous data pattern with the newly written one. If the two are different and one of them matches the compression table entry, it means the corresponding MSB should be updated and the processor notifies the directory. Note that the notification is needed only when the matching result changes after a store operation, and it is normally only a small fraction of total store operations. Detailed experimental results regarding this will be provided in Section IV. Using this method, the directory is able to keep track of the matching status of most recent data values for all cache blocks.

When a cache miss occurs, the processor sends a request message to the directory. Then the directory looks up the MSB of the corresponding requested block. If the MSB in the directory is clear, meaning that the block does not match the compression table entry, the directory follows the normal cache miss procedures to provide data to the requestor by forwarding the request to another owner node or sending the data block by itself if the memory is an owner. However, if the MSB is set, the directory sends a reply message to the requestor to notify that the data packet has been eliminated. Once the requestor receives the reply message from the directory, it does not need to wait for another message but only has to copy the compression table data to the corresponding

cache line. The cache miss handling finishes at this moment, reducing the miss latency significantly.

As in the case of compression tables, the directory contains two MSBs for each cache block to handle two different pattern sets. The two bits are used alternately for each period by switching to the other bit every time a new pattern set is ready. Also, a valid bit should be assigned to each MSB to invalidate all MSBs used for the previous period when a new pattern set appears in the system. Thus, space overheads of MSBs and their valid bits are 4 bits per cache block, which corresponds to 0.7% of cache block size assuming 64-byte block. All control packets used for packet elimination such as request, reply or notification messages must carry the sender's PB to handle two different pattern sets that may be used mixedly.

IV. PERFORMANCE EVALUATION

A. Simulation Framework

We measure the performance of the proposed schemes using Simics full-system simulator [10]. In order to simulate a shared memory model for a multiprocessor system, we also use General Execution-driven Multiprocessor Simulator (GEMS) [11] which is implemented in the form of a module used by Simics simulator. We use MOESI_SMP_directory as a cache coherence protocol, which has five cache block states and provides directory-based cache coherency. Table I shows the system parameters used in the simulation. Since we assume that each processor has its own off-chip memory, the total number of nodes in the system is double the number of processors.

Benchmarks used in this paper are three SPLASH-2 [12] (*radix*, *fft*, *lu*), four SPEC OMP2001 (*equake*, *fma3d*, *swim*, *mgrid*) and two PARSEC [13] (*streamcluster*, *swaptions*) benchmarks for scientific workloads. We also use SPECjbb2000 server benchmark for commercial workloads.

B. Simulation Results

First, we clarify how much VSC is useful in data compression by measuring average compression ratios, which denotes the average of original block size divided by compressed block size. The performance of VSC is compared to the most recent two compression schemes [2], [1] for interconnects. Figure 5 shows the average compression ratios for three different compression schemes: table-based adaptive compression called Frequent Value Table (FVT) [2], significance-based static compression called Frequent Pattern Compression (FPC) [1] and VSC. VSC performs as good as the other two schemes in overall, while greatly surpassing them in *mgrid-8p* benchmark.

In Figure 6, we show the overall system performance in terms of instructions per cycle (IPC) of three different schemes: baseline, FVT and VSC. The baseline system does not use compression techniques for data packets, while FVT and VSC try to compress data packets to reduce their sizes. VSC shows better performance than the others, by 36% compared to the baseline and 23% to FVT on average. This enhancement is achieved because of the significant reduction of cache miss latency by the compression and packet elimination scheme in VSC. To support our analysis, we measure average cache miss latency of the three schemes as shown in Figure 7. Again, VSC outperforms the other two schemes, by 34% compared to the baseline and 20% to FVT on average. By comparing the results of FVT and VSC, we can find out that the amount of enhancement looks quite

Table I: System parameters

Parameters	Values
CPU	1GHz UltraSPARCIII+, 8/16 processors
L1 I & D cache	4-way, 16KB, 3 cycles
L2 cache	4-way, 4MB, 12 cycles
Cache block size	64B
Memory	4GB, 480 cycle access time
Number of memory modules	same as the number of processors
Directory access time	80 cycles
Network topology	hierarchical switches (fanout degree of 4)
Network link bandwidth	3G bytes/sec.
OS	Sun Solaris 9
Compression/decompression latencies	2 cycles

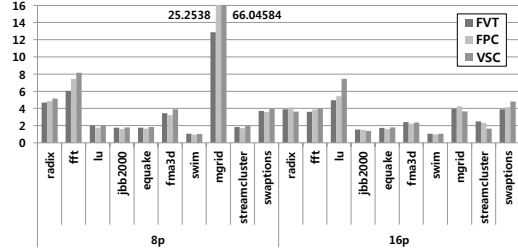


Figure 5: Compression ratios

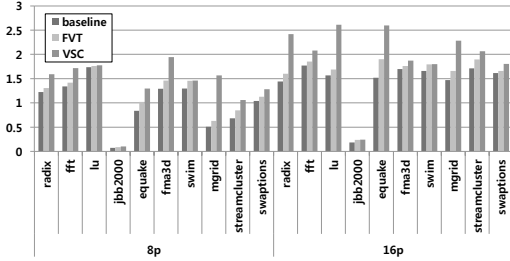


Figure 6: Overall system performance

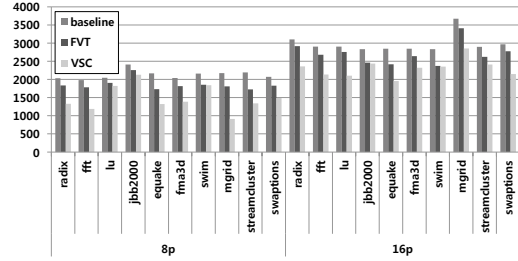


Figure 7: Average cache miss latency

larger than that in Figure 5. This is because the packet elimination in VSC greatly helps to simplify a miss handling procedure in addition to packet size reduction, by cancelling request forwarding and data packet transmission. FVT compresses a data packet through encoding the data into the index of a table entry it matches, which reduces data packet latency but cannot totally hide it. But in VSC, the most frequent 64-byte data pattern is known to all nodes in the system, and data packet transmission is completely cancelled if the requested data matches the pattern because the data is already available to the requestor.

In order to analyze the results in more detail, we examine how much VSC affects network latency compared to the other two schemes. Figure 8 shows average data packet latency, which is a network delay of a data packet. Here we assume that the latency of eliminated packets in VSC is zero. VSC achieves approximately 64% of enhancement from the FVT scheme, even decreasing the latency by more than half on average compared to the baseline. By eliminating and compressing data packets, VSC makes a significant contribution to reducing network overheads, enhancing the overall performance.

We measure the amount of eliminated data packets in VSC to clarify the effect of packet elimination. As shown in Figure 9, up to 96% of data packets can be eliminated depending on the workload and 41% of packets are removed on average. Table II illustrates numbers of MSB updates by owner nodes. As explained in Section III-C, the processor must notify the directory if it needs to update MSB when it becomes the owner after a store operation. Since it could incur considerable network overheads if the notification occurs frequently, we measure the percentage of notifications over the total number of store operations. Results show that the average percentage is approximately 2.8%, which is marginal in overall.

Table III shows how many data packets are injected during the time when two pattern sets are mixedly used between two adjacent periods. As seen in the table, the amount

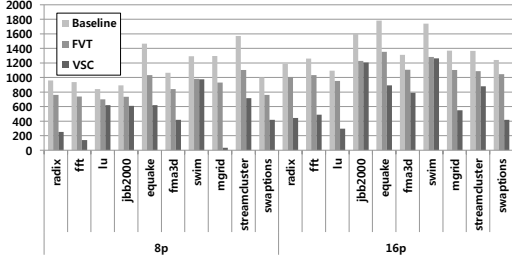


Figure 8: Average data packet latency

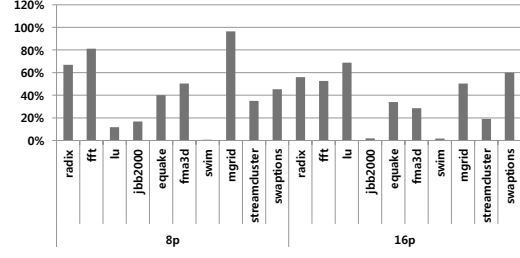


Figure 9: Eliminated data packets

Table II: The percentage of notifications from owner nodes

Processors	Benchmark	# of Stores	Notifications	Percentage
8p	radix	128857	1536	1.192019%
	fft	102629	1657	1.614553%
	lu	33893	330	0.973652%
	job2000	408706	13229	3.236801%
	equake	848430	8015	0.944686%
	fma3d	1546842	149257	9.649143%
	swim	2401947	61208	2.548266%
	mgrid	13677	483	3.531476%
	streamcluster	9259	25	0.270008%
	swaptions	24932	332	1.331622%
16p	radix	98063	1193	1.216565%
	fft	102629	1657	1.614553%
	lu	312470	4752	1.520786%
	job2000	114726	1207	1.052072%
	equake	320219	5172	1.615145%
	fma3d	675611	98320	14.552753%
	swim	713277	1511	0.211839%
	mgrid	21951	1354	6.168284%
	streamcluster	6578	31	0.471268%
	swaptions	9588	127	1.324572%

Table III: Mixed packets in two adjacent periods

Processors	Benchmark	Total	Mixed Pkts	Percentage
8p	radix	239156	115	0.048086%
	fft	658918	274	0.041583%
	lu	73898	33	0.071125%
	job2000	492910	129	0.026171%
	equake	4330889	1099	0.025376%
	fma3d	2635812	930	0.035283%
	swim	6430531	2307	0.035876%
	mgrid	16424012	4172	0.025402%
	streamcluster	2102753	1192	0.056688%
	swaptions	4001181	1273	0.031816%
16p	radix	234829	104	0.044288%
	fft	124391	66	0.053059%
	lu	568546	310	0.054525%
	job2000	487468	75	0.015386%
	equake	1054687	691	0.065517%
	fma3d	997459	577	0.057847%
	swim	1070339	615	0.057458%
	mgrid	152513	38	0.024949%
	streamcluster	503260	285	0.056631%
	swaptions	3210054	1256	0.038504%

of those packets among total number of injected packets is less than 0.05% on average, which is an almost negligible result.

Finally, we measure the compression ratios and the number of selected nodes with different k values to clarify how the selective update is beneficial compared to the non-selective one. Figure 10(a) shows the compression ratios normalized to the result with a non-selective method. As shown in Figures 10(a) and 10(b), we can verify that the number of selected nodes reduces significantly by 70% on average with the sacrifice of less than 1% in terms of a compression rate. It means that the network overheads can be considerably reduced by 70% for gathering information from nodes in VSC update, with almost no penalty in compression rates.

V. CONCLUSIONS

In this paper, we have proposed VSC to alleviate network overheads by reducing the number of packets as well as the packet size. The packet compression technique, VSC, coupled with an underlying cache coherence mechanism, achieves significant performance improvement by cancelling packet transmission for the most frequent data known to all nodes. Eliminating data packet transmission tremendously reduces cache miss latency, which enhances overall system performance. In simulation results, VSC outperforms the existing FVT compression scheme [2] by 23% on average in terms of overall execution time, and by 20% in cache miss latency. Compared to the baseline system with no compression techniques, the amount of performance improvement in VSC significantly increases up to 36% in terms of overall execution time and 34% for cache miss latency.

Our work can be explored further by investigating the performance effect when using different topologies for multiprocessor systems. Also, we plan to analyze the performance

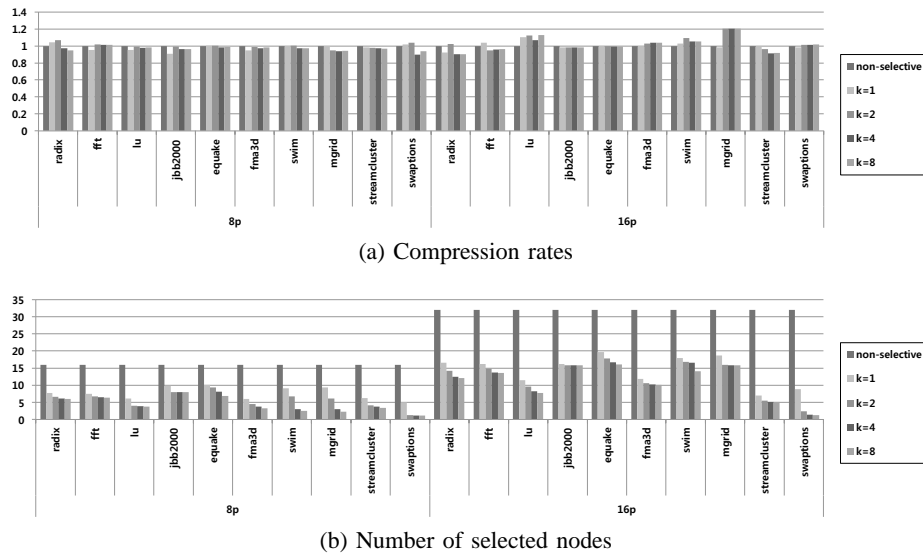


Figure 10: Effect of selective update

of VSC under mixed workloads or different system domains such as chip multiprocessor (CMP) systems.

REFERENCES

- [1] R.Das, A.K.Mishra, C.Nicopoulos, D.Park, V.Narayanan, R.Iyer, M.S.Yousif, and C.R.Das, "Performance and power optimization through data compression in network-on-chip architectures," in *Proceedings of HPCA*, 2008.
- [2] Y.Jin, K.H.Yum, and E.J.Kim, "Adaptive data compression for high-performance low-power on-chip networks," in *Proceedings of MICRO*, 2008.
- [3] A.R.Alameldeen and D.A.Wood, "Adaptive cache compression for high-performance processors," in *Proceedings of ISCA*, 2004.
- [4] D. Citron and L. Rudolph, "Creating a wider bus using caching techniques," in *Proceedings of HPCA*, 1995.
- [5] M.Farrens and A.Park, "Dynamic base register caching: a technique for reducing address bus width," in *Proceedings of ISCA*, 1991.
- [6] J.-S.Lee, W.-K.Hong, and S.-D.Kim, "Design and evaluation of a selective compressed memory system," in *Proceedings of ICCD*, 1999.
- [7] M.H.Lipasti, C.B.Wilkerson, and J.P.Shen, "Value locality and load value prediction," in *Proceedings of ASPLOS*, 2007.
- [8] R.B.Tremaine, T.B.Smith, M.Wazlowski, D.Har, K.-K.Mak, and S.Arramreddy, "Pinnacle: Ibm mxt in a memory controller chip," *IEEE Micro*, 2001.
- [9] Y.Zhang, J.Yang, and R.Gupta, "Frequent value locality and value-centric data cache design," in *Proceedings of ASPLOS*, 2000.
- [10] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, B. Werner, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.
- [11] M.M.Martin, D.J.Sorin, B.M.Beckmann, M.R.Marty, M.Xu, A.R.Alameldeen, K.E.Moore, M.D.Hill, and D.A.Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," *Computer Architecture News(CAN)*, 2005.
- [12] S.C.Woo, M.Ohara, E.Torrie, J.P.Singh, and A.Gupta, "The splash-2 programs: Characterization and methodological considerations," in *Proceedings of ISCA*, 1995.
- [13] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, October 2008.