VQ Compression Algorithms on A Multiprocessor System

Akiyoshi Wakatani
Faculty of Science and Engineering, Konan University
8-9-1, Okamoto, Higashinada, Kobe, 658-8501, Japan
email: wakatani@konan-u.ac.jp

ABSTRACT

A variety of parallel processing technologies have been implemented in a processor, and thus a cutting edge algorithm for multimedia applications should be aware of parallel processing features. We implemented parallel algorithms for VQ compression on two parallel environments and evaluated the effectiveness of the parallel algorithms.

On a multiprocessor system with distributed memories, we evaluate two parallel algorithms for the codebook generation of the VQ compression: parallel LBG and aggressive PNN. We measured the speedups and elapsed times of both algorithms on a PC cluster system and find that both algorithms can achieve scalable parallelisms for the case with a large number of training vectors.

On the other hand, for a codeword search on a system with a shared memory, the p-dist approach and the c-dist approach with the aggregation of synchronizations are suitable for a small codebook, and the c-dist approach and the p-dist approach with the ADM or the strip-mining method are suitable for a large codebook. However, since the aggregation of synchronizations and the strip-mining method increases the space complexity of the algorithm, the p-dist approach and the c-dist approach are more suitable for a small codebook and for a large codebook, respectively.

Keywords: vector quantization, parallel processing, compression, LBG, codebook generation

1. INTRODUCTION

A variety of parallel processing technologies have been implemented in a processor, such as an execution pipeline and a super scalar architecture[1]. A recent "hyper-threading" processor emulates two virtual processors in order to enhance the utilization of pipeline phases and plural execution units by executing two different threads simultaneously[2] and some processors have plural processing cores in a chip ("dual core") to execute plural threads in a parallel way[3]. Thus, a cutting edge algorithm for multimedia applications must be aware of parallel processing features and should be parallelized easily and efficiently.

Multimedia applications include communicating, saving and retrieving still and motion pictures and audio data, mostly, through a high-speed network such as Internet. For such an environment, advanced compression technologies are key issues. Among several compression algorithms, *VQ (vector quantization)* is one of the most prominent methods for compressing multimedia data at a high compression rate. A key to the high compression rate is to build an efficient codebook that represents the source data with the least quantity of bit stream. The VQ compression consists of two parts: 1) codebook generation, and 2) codeword search[4]. Two major methods of generating a codebook for the VQ compression are *PNN(Pairwise Nearest Neighbor)*[5] and *LBG*[4] algorithms. Both methods require vast computing resources to determine an efficient codebook. In the codeword search, each vector of the image should be assigned to the most appropriate codeword to minimize the quantity of the bit stream.

We consider two types of multiprocessor systems: a system with distributed memories and a system with a shared memory. In a system with distributed memories, each processor has its own main memory and address space, and then has to exchange a message explicitly between processors to get the data of other processor. Since the communication cost is relatively higher than the computational cost, it is required that the number of the messages and the quantity of the communication must be minimized. On the other hand, in a system with a shared memory, every processor can reach any memory location without message exchanges, so it is easy to write a parallel program on that because users concentrate only on a task distribution over processors. However, since a synchronization between processors creates an overhead time and the increase of the cache miss rate results in the congestion of the access to the shared main memory, it seems that the effective parallelism falls easily without careful

examination by users.

In this paper, we focus on a system with distributed memories and a system with a shared memory, and consider the availability and efficiency of the VQ compression algorithm on the environments. The remainder of this paper is organized as follows: Section 2 provides the description of algorithms for codebook generation on a distributed memory system and its evaluation. In Section 3, algorithms for optimal codeword search on a shared memory system and its implementation are described and some improvements are also presented. Finally, Section 4 concludes the paper with a summary.

2. CODEBOOK GENERATION ON A DISTRIBUTED MEMORY SYSTEM

2.1. Parallel LBG

The LBG algorithm generates a codebook by clustering initial training vectors with the K-mean method. Dhillon et al. realized a parallel version of the K-mean method with the MPI library on distributed memory multiprocessors[6]. The MPI is de facto standard library for communicating messages on distributed memory multicomputers and has been implemented on many platforms. The parallel version of the LBG algorithm (*Parallel LBG*) is based on Dhillon's K-mean algorithm.

On this parallel algorithm, the training vectors are distributed over processors and the centroid of the clusters are shared and updated locally and then globally. The LBG mainly has two steps: the distance calculation and minimum determination step and the centroid update step. The calculations of the first step can be carried out in parallel because the training vectors are distributed. However the second step consists of three stages. On the sequential LBG, the new centroid of the k-th cluster is calculated as follows:

$$centroid_k = \sum_{i=0}^{n_k} T_{k,i}/n_k$$

where $T_{k,i}$ is the i-th training vector which belongs to the k-th cluster and n_k is the number of training vectors which belong to the k-th cluster. Instead, on the parallel LBG, each processor calculates local centroids first and broadcasts the local centroids to others after that. Then the new centroids should be calculated independently as follows:

$$\begin{aligned} local\_centroid_{p,k} &= \sum_{i=0}^{n_{p,k}} T_{p,k,i}/n_{p,k} \\ \texttt{broadcast} &\quad local\_centroid_{p,k} \\ centroid_k &= \frac{\sum_{i=0}^{p} local\_centroid_{p,k} \times n_{p,k}}{\sum_{i=0}^{p} n_{p,k}} \end{aligned}$$

where $T_{p,k,i}$ and $n_{p,k}$ are the i-th training vector which belongs to the k-th cluster on processor p and the number of training vectors which belong to the k-th cluster on processor p, respectively. Note that the computation of the third stage is duplicated over processors.

The cost of the parallel LBG mainly consists of a) the computation part (the first step and the first stage of the second step), b) the communication part (the second stage of the second step) and c) the update part (the third stage of the second step). The complexity of the first step of the parallel LBG is $O(ITR*T*K/P)$ and the first stage of the second step is $O(ITR*T/P)$ where ITR is the number of iterations until the codebook is converged and T and K are the number of training vectors and codebook, respectively. The second stage of the second step is a broadcast communication of K local centroids which requires the complexity of $O(ITR*K*P*\log P)$ by using Van de Geijn's broadcast algorithm[7]. Finally the complexity of the third stage of the second step is $O(ITR*K*P)$. Note that the second and third stages are overheads which the sequential LBG does not contain and are proportional to the number of processors, thus these stages may degrade the effective parallelism.

2.2. Aggressive PNN

In order to improve the performance of the PNN algorithm further, we should parallelize the Lazy PNN algorithm without any loss of scalability to the problem size and suitability for a high compression rate. The Lazy PNN consists of five steps: 1) determine the nearest neighbor of vector (NN) of each training vector and the distance between the training vector and its NN, 2) sort the distances, 3) merge the pair of training vectors with the minimum distance into one vector, 4) stop if the total number of the vectors equals the size of codebook (K), and 5) recalculate the NNs and distances if necessary and goto 2.

We proposed the safe PNN algorithm and the aggressive PNN algorithm[8]. The safe PNN algorithm generates the same codebook as the sequential PNN does, but cannot provide enough effective parallelism. On the other hand, the codebook generated by the aggressive PNN is slightly different from that by the sequential PNN, but the aggressive PNN outperforms the safe PNN in terms of scalability and parallelism. Thus we focus on the aggressive PNN in this paper. The aggressive PNN consists of eight steps: 1) share all training vectors over processors, but divide them into sub vector groups which are assigned to different processors, 2) determine the nearest neighbor of vector (NN) of each training vector in the sub vector group and the distance between the training vector and its NN, 3) sort the distances locally (sort list A), 4) broadcast the first B pairs in the sort list A to others, 5) sort the $B*P$ broadcasted pairs (sort list B), 6) merge the first B pairs

in the sort list B, 7) stop if the total number of the vectors equals the size of codebook (K), and 8) recalculate the NNs and distances if necessary, sort them again and goto 4. Note that if B is 1, the aggressive PNN generates the same codebook as the sequential PNN does. The main concept of the aggressive PNN is that each processor has its own sort list, broadcasts the block of pairs to others and merges them aggregately. Since the most costly part of the aggressive PNN is the communication overhead of the broadcast in the step 4, the aggregate merge is implemented to reduce the number of the broadcast communications.

The cost of the aggressive PNN mainly consists of a) the computation part (steps 2 and 3), b) the communication part (step 4) and c) the merge part (steps 5, 6, 7 and 8). The complexities of the steps 2 and 3 of the aggressive PNN are $O(T*T/P)$ and $O((T/P)*(T/P))$, respectively, but the steps 2 and 3 are carried out only once. The complexity of the step 4 is $O(((T-K)/B)*P*\log P)$ because B is not large, so the communication cost is determined by the number of the communications and the cost of a broadcast with P processors is $O(\log P)$. Note that the cost of the step 4 can be reduced by increasing B, but the difference of codebooks generated by the aggressive PNN and the sequential PNN algorithms increases and it may degrade the quality of the codebook generated by the aggressive PNN. The complexity of the steps 5 and 6 is $O(((T-K)/B)*((B*P)*B))$ because the dominant part of these two steps is selecting the first B pairs among $B*P$ pairs. Finally, the cost of the step 8 is slight because the number of recalculations is small compared with T and K according to the result of our preliminary experiments, so it can be ignored.

2.3. Experiment and Discussion

2.3.1. Degradation of Image Quality for Aggressive PNN

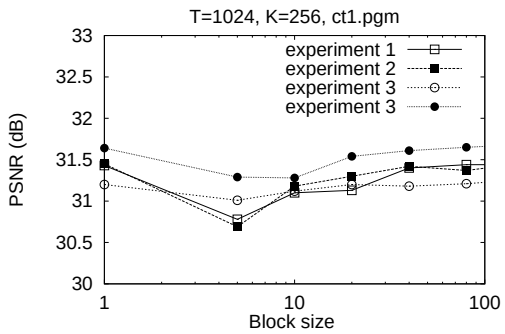


Figure 1. Degradation of image quality

In order to evaluate the quality of an image compressed by using the aggressive PNN algorithm, we considered several images including a CT scan image with the size of 512×512 and the gray level of 8 bit. For the image, 1024 training vectors are randomly chosen to determine a codebook of 256 vectors. Four experiments are carried out with varying initial training vectors. Figure 1 shows the relation of the block size and PSNR (Peak Signal Noise Ratio) for each experiment. The block size (B) means the number of vector pairs that are aggressively merged in concurrent. As mentioned earlier, the aggressive PNN with the block size of 1 generates the same codebook as the sequential PNN does.

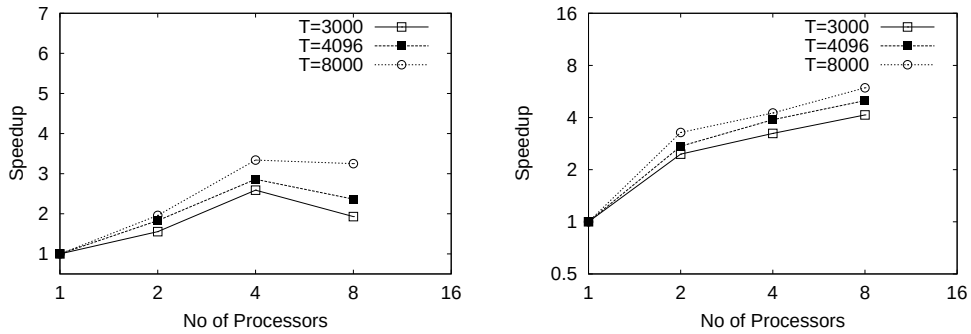
As shown in Figure 1, by increasing the block size, PSNR is slightly degraded. For example, for the experiment 1, the sequential PNN algorithm generates a codebook at PSNR of 31.5dB, while the aggressive PNN algorithm with the block size of 10 generates a codebook at PSNR of 31.0dB. It is concluded that the aggressive PNN algorithm can be applicable to the codebook generation for the VQ compression because the degradation of image quality is slight.

2.3.2. Comparison of Parallel LBG and Aggressive PNN

We measured an elapsed time of each part of both algorithms on a PC cluster which consists of 8 CPUs (Celeron 1GHz) and LAN (100Mbps) under MPICH1.2.4 and Linux 2.4. Our experiments generate a codebook of 2048 beginning with training vectors of 3000, 4096 and 8000. The parallel LBG iterates the procedure until the total difference of the calculated and previous centroids is under $0.01/K$. Note that the number of iterations is $((T-K)/B)$ for the aggressive PNN and we choose 64 as B for our experiments. For the parallel LBG, several experiments were carried out with varying the initial conditions and the best results are plotted in the graphs.

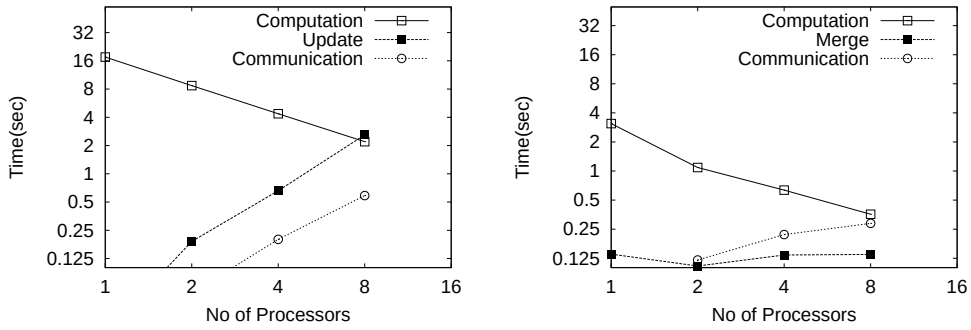
As mentioned in the previous section, the costs of the three parts of the parallel LBG are 1) the computation part of $O(ITR*T*K/P)$, 2) the communication part of $O(ITR*K*P*\log P)$ and 3) the update part of $O(ITR*K*P)$ and those of the three parts of the aggressive PNN are 1) the computation part of $O(T*T/P)$, 2) the communication part of $O(((T-K)/B)*P*\log P)$ and 3) the merge part of $O(((T-K)/B)*((B*P)*B))$. Thus it is expected that the part 1 of both algorithms can be carried out in parallel, but the elapsed times of the rest parts increase as the number of processors increases.

The results of the experiments are shown in Figure. 2 and 3. The speedups of both algorithms are almost same except for an 8 CPU case: as the size of training vectors increases, the speedups increase, say, the speedups of the aggressive PNN with 3000 and 8000 training vectors on



(a) Parallel LBG (b) Aggressive PNN

Figure 2. Speedup (lenna)



(a) Parallel LBG (T=8000, 32.46dB) (b) Aggressive PNN (T=3000, 32.10dB)

Figure 3. Case where the image quality is same

an 8 CPU cluster are 4.16 and 5.94, respectively. However, the speedups of the parallel LBG are degraded on an 8 CPU cluster, because the costs of the communication and update parts exceed the parallelization merit of the computation part, namely, the former are proportional to $P * \log P$ and P and the latter is inversely proportional to P. The costs of the update part are nearly equal to or more than that of the computation part for all the cases on an 8 CPU cluster and that of the communication is in a similar situation. Therefore, the scalability of the parallel LBG is not so good.

When the qualities of images compressed by both algorithms are same, the number of training vectors required by the aggressive PNN is much less than that by the parallel LBG, and the aggressive PNN is superior in terms of the elapsed time. For example, the quality of images compressed by the parallel LBG with 8000 training vectors and the aggressive PNN with 3000 training vectors are almost same (32.46dB and 32.10dB), but the total elapsed times on an 8 CPU cluster are 5.4 sec and 0.78 sec for the parallel LBG and the aggressive PNN, respectively. The reason is partially the difference of the hit rate of cache memories because the aggressive PNN requires a smaller memory area for the smaller number of training vectors.

As shown above, the parallel algorithm for the VQ codebook generation using the PNN method requires that the first several data elements of a locally-sorted list on each processor should be broadcasted and then sorted later, but MPI does not provide any collectives for this procedure, called “Allsort”. The Allsort procedure consist of several steps as follows: 1) sort B data locally and generates a locally-sorted list, 2) collect all the locally-sorted lists of the size of $P \cdot B$ and 3) sort them and select the first C data of the globally-sorted list. Note that C is between B and $P \cdot B$ and C is B for the aggressive PNN. We will propose the implementation of this collective in the future[9].

3. CODEWORD SEARCH ON A SHARED MEMORY SYSTEM

3.1. Two Approaches

The key to the enhancement of the effective parallelism is to decompose a task evenly over processors and reduce the number of synchronizations. We have two alternatives to search an optimal codeword for each vector of the image: 1) *p-dist:* the image is divided into sub images and each processor is in charge of finding the optimal codeword for the vectors of one of the sub images, and 2) *c-dist:* the codebook is divided into sub codebooks and each processor is in charge of finding the locally-optimal codeword for the vector with one of the sub codebooks and then finds the globally-optimal codeword among all the local-optimal codewords. Figure 4 shows the outline of the codeword search where I, D and K are the number of pixels, the size of a vector and the number of codewords in the codebook. Note that $\frac{I}{D}$ is the number of vectors. Namely the p-dist approach distributes the outer loop ("A") of the code in the Figure over processors and the c-dist approach distributes the inner loop ("B") instead.

```
for(i=0;i < I/D;i++){ /* (A) */
    xmin=C; jmin=-1;
    for(j=0;j < K;j++){ /* (B) */
        x=distance(image[i], codebook[j]);
        if(xmin > x){xmin=x; jmin=j;}
    }
    cluster[i]=jmin;
}
```

Figure 4. Optimal codeword search

In the p-dist approach, each processor takes a look at the whole codebook for each vector. Thus, if the size of the codebook is larger than the size of the cache memory of the processor, the elapsed time of the codebook access increases due to the increase of the main memory access. However, when the codebook is less than the size of the cache memory, this approach can be completely parallelized because of no synchronizations required. Moreover, in order to reduce the cache miss rate for a large codebook, we have two alternatives. One is "strip-mining" method and another is "alternating direction memory-access" method (ADM). The strip-mining method divides the inner loop into sub loops and places a new loop out of the outer loop. Then it enhances the cache hit rate of the codebook accesses in the new most-inner loop by keeping the intermediate search results for all the vectors on a buffer. In the ADM method, the cache hit rate of the early part of the codebook accesses can be improved by changing the direction of the inner loop as in Figure 5.

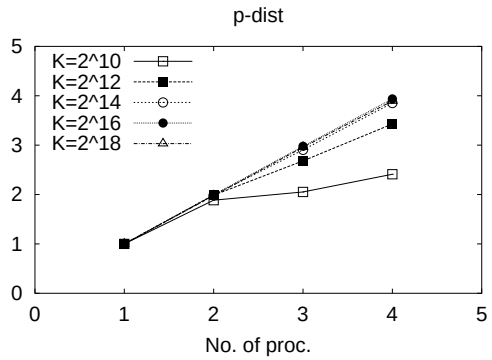
```
startI=(I/D)*myID;  endI=(I/D)*(myID+1);
for(i=startI;i < endI ;i++){
    xmin=C; jmin=-1;
    if(i%2==0)
        for(j=0;j < K;j++){
            x=distance(image[i], codebook[j]);
            if(xmin > x){xmin=x; jmin=j;}
        }
    else
        for(j=K − 1;j >= 0;j–){
            x=distance(image[i], codebook[j]);
            if(xmin > x){xmin=x; jmin=j;}
        }
    cluster[i]=jmin;
}
```

Figure 5. P-dist with ADM method

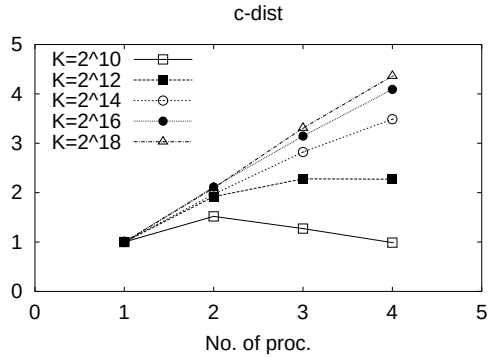
On the other hand, it seems natural that the c-dist approach require a synchronization after all the processors decides the locally-optimal codeword to decide the globally-optimal codeword. However, the number of synchronizations can be easily reduced by keeping several locally-optimal codewords for several vectors on a buffer, *LOC buffer*(Locally Optimal Codeword buffer), and finding the globally-optimal codewords aggregately. The size of the LOC buffer depends on the size of globally-optimal codewords to be determined aggregately, which is called "synchronization period". Note that as the size of the buffer grows, the number of synchronization decreases but the space complexity increases. Moreover, since the codebook is divided, the c-dist approach works well for a large codebook compared with the p-dist approach in terms of cache hit rate. The complexity of the synchronization (T_{sync}) is $O(\frac{\log P}{S})$ and the complexity of the c-dist with the LOC buffer (T_{loc}) is $O(\frac{1}{P})$ where P is the number of processors and S is the synchronization period. Thus, since $T_{sync} : T_{loc} = O(\frac{P \log P}{S}) : O(1)$, the iso-efficiency parallelism can be achieved if B is directly proportion to $P \log P$.

3.2. Experiments and Discussion

We implemented two algorithms for the codeword search on a SMP system with four AMD Opteron 846 processors (2.0GHz, L2 cache: 1M Byte, L1-D cache: 64K Byte) and 2G Byte main memory under SUSE-Linux Ver9.0 (kernel 2.4.21) and created programs with C language and POSIX thread library, which were compiled by gcc 3.3.21. We measured the elapsed time and speedup for



(a) Distributed pixel



(b) Distributed codebook

Figure 6. Speedup of codeword search

encoding of a graymap image with the size of 1024×1024 using the size of a vector of 16 pixels.

Figure 6-(a) shows the speedup of the p-dist approach with the codebook size of 2^{10} to 2^{18}. A linear speedup cannot be achieved for the cases where the codebook size is 2^{10} and 2^{12} because the overhead of thread creation is relatively large. Although a linear speedup can be achieved for a large codebook, the elapsed time increases due to the cache miss penalty of the codebook access. We will describe the detail of this phenomena later.

As mentioned earlier, the c-dist approach requires a synchronization between processors to determine the globally-optimal codeword, so the overhead of the synchronization degrades the effective parallelism. As shown in Figure 6-(b), a linear speedup cannot be achieved for the case of the codebook of 2^{10} to 2^{14}, but the speedup is getting close to a linear speedup as the size of the codebook increases because the overhead cost of the synchronization is amortized.

3.3. Reduction of Synchronizations

By aggregating several synchronization points, the effective parallelism can be improved for the c-dist approach. Thus, each processor has to keep several local-optimal codewords on the LOC buffer and synchronize with other processors to decide globally-optimal codewords aggregately.

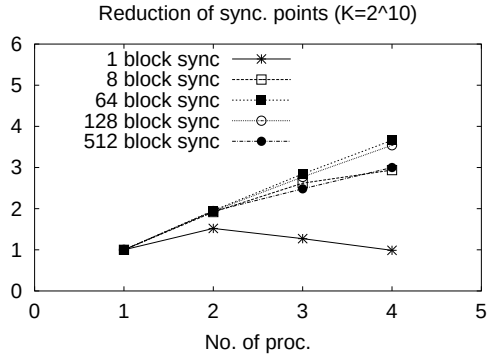


Figure 7. Reduction of synchronizations

Figure 7 shows the results for the size of the codebook of 2^{10}. For 4 processor case, the speedup is just 0.98 when synchronized every vector, but the speedup is 2.93 when synchronized every 8 vectors and the speedup goes up to 3.66 when synchronized every 64 vectors. However, the speedup goes down to 3.54 when synchronized every 128 vectors and the speedup is reduced to 3.00 when synchronized every 512 vectors. The reason is that the overhead of the synchronization is still large for a small synchronization period and the cache hit rate is worse due to the increase of the LOC buffer area for a large synchronization period. Therefore the key to the efficient c-dist approach is to determine the synchronization period appropriately and optimally.

3.4. Cache Effect for A Large Codebook

Since all the codewords of the codebook are accessed sequentially for the p-dist approach, the access latency easily increases when the size of codebook exceeds the size of the cache memory of the processor.

Figure 8 shows the relative performance of several methods when the elapsed time of the c-dist is 1. When $K = 2^{18}$, the size of the codebook is 2^{22} Byte, that is, larger than the size of the L2 cache memory (1M Byte), so the elapsed time of the p-dist is larger than that of the c-dist by about 10% for the 4 processor case. However, by applying the ADM to the p-dist, the cache hit rate is improved and its relative performance almost reaches the performance of the c-dist (1.01). Then the strip-mining is applied to

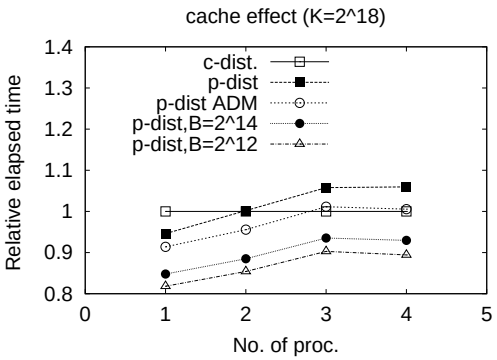


Figure 8. Effect of cache miss for a large codebook

the p-dist and we carry out two cases where the size of the most inner loop is 2^{14} and 2^{12}. For both cases, the relative performance is about 0.9, so it is confirmed that the p-dist with the strip-mining method outperforms the c-dist even for a large codebook, however the strip-mining method requires a buffer area to keep the intermediate results of the whole vectors of $O(\frac{I}{P})$, thus the space complexity of the main memory increases dramatically.

4. CONCLUSION

On a multiprocessor system with distributed memories, we evaluate two parallel algorithms for the codebook generation of the VQ compression. We measured the speedups and elapsed times of both algorithms on a PC cluster system and find that both algorithms can achieve scalable parallelisms for the case with a large number of training vectors. However, as the the number of processors increases, the parallelism of the parallel LBG is degraded. In addition, it is found that when the qualities of images compressed by both algorithms are same, the aggressive PNN is superior to the paralle LBG in terms of the total elapsed time.

For a codeword search on a system with a shared memory, the p-dist approach and the c-dist approach with the aggregation of synchronizations are suitable for a small codebook, and the c-dist approach and the p-dist approach with the ADM or the strip-mining method are suitable for a large codebook. However, since the aggregation of synchronizations and the strip-mining method increases the space complexity of the algorithm, the p-dist approach and the c-dist approach are more suitable for a small codebook and for a large codebook, respectively.

In the near future, we will evaluates the codebook generation algorithms on shared memory systems and the codeword search algorithms on distributed memory systems. Also we will confirm the effectiveness of our algorithm for a system with a shared memory on a large SMP system, a dual-core processor and a hyper-threading system and so on.

ACKNOWLEDGMENTS

This work was supported by MEXT ORC (2004-2008), Japan.

REFERENCES

[1] Culler D., Singh J. and Gupta A.: “Parallel Computer Architecture: A Hardware/Software Approach”m Morgan Kaufmann Pub., (1998)

[2] Marr D. et. al.,: “Hyper-Threading Technology Architecture and MIcroarchitecture”, Intel Technology Journal, Vol.6, http://www.intel.com/technology/itj/2002/volume06issue01/vol6iss1_hyper_threading_technology.pdf (2002)

[3] “Smash the Hourglass with the AMD Athlon? 64 X2 Dual-Core Processor”, http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_9485_13041,00.html

[4] Gersho A. and Gray R.: ”Vector Quantization and Signal Compression”, Kluwer Academic Publishers, Boston (1992)

[5] Equitz W.: ”A new vector quantization clustering algorithm”, IEEE trans. on Acoustics, Speech and Signal Processing, Vol.37, No.10, pp.1568-1575 (1980)

[6] Dhillon I. and Modha D.: “A data-clustering algorithm on distributed-memory multiprocessors”, Large-Scale Parallel Data Mining, pp.245-260 (1999)

[7] Thakur R. and Gropp W.: “Improving the performance of collective operations in MPICH”, Proc. of Euro PVM/MPI 2003, pp.259-267 (2003)

[8] Wakatani A.: “Parallelization of VQ Codebook Generation using Lazy PNN Algorithm”, Parallel Computing: Software Technology, Algorithms, Architectures & Applications, Editors; G. Joubert et al., Elsevier Science, pp.415-422, (2004)

[9] Wakatani A.: “A VQ compression algorithm on a multiprocessor system with a global sort collective function”, IEEE ISSPIT 2005 (submitted) (2005)