PHDCM : Efficient Compression of Hangul Text in Parallel

PHDCM : 병렬 컴퓨터에서 한글 텍스트의 효율적인 축약

Yong Sik Min*
민 용 식*

ABSTRACT

This paper describes an efficient coding method for Korean characters using a three-state transition graph. To our knowledge, this is the first achievement of its kind. This new method, called the Parallel Hangul Dynamic Coding Method(PHDCM), compresses about 3.5 bits per a Korean character, which is more than 1 bit shorter than the conventional codes introduced thus far to achieve extensive code compression. When we ran the method on a MasPar machine, which is on SIMD-SM (EREW PRAM)., it achieved a 49.314-fold speedup with 64 processors having 10 million Korean characters.

요 약

본 논문은 3가지 상태의 전이 그래프를 이용해서, 병렬 컴퓨터인 MasPar에 적합한 한글에 대한 효율적인 부호화를 제시하고자 한다. 본 논문에서 제시한 PHDCM(Parallel Hangul Dynamic Coding Method)의 방법을 이용한 경우에 한글 한 음절당 약 3.5비트이상의 축약이 가능함을 보였다. 그리고 기존의 방법과 비교해 볼때 1비트이상의 축약이 가능함도 보였다. 또한 약 천만자의 한글을 이용해서, 병렬 컴퓨터인 MasPar에 프로세서 64개를 이용하여 실제 실행을시켰을때의 가속도(Speedup)은 49.314임을 보였다.

Ⅰ. Introduction

Because of rapid progress in data communications, we are able to acquire the information we need with ease. Although the parallel computer has made it possible to receive/transmit large volumes of data easily, it is necessary, nonetheless, to transmit only a minimal amount of data bits. In particular, in the case of Hangul, (the Korean characters), we constructed a device which processes more efficient information if used for its own special function. Even though most people studied the mechanism and the construction of Hangul's architecture, we need to minimize the length of bits in Hangul.

This paper suggests a parallel data compression method that applies to Hangul. Using a parallel machine such as the MasPar computer, which is on EREW-PRAM, we developed the Parallel Hangul Dynamic Coding Method(PHDCM), which minimizes the bits, compared with any other conventional method. We used the dynamic Huffman code and our analysis of Hangul characteristics to develop a tri-state transition graph that greatly improves the efficiency of data compression for the Hangul text. When the PHDCM was implemented on a MasPar machine, it achieved a 49.314-fold speedup using 64-processors having 10 million Korean characters.

Ⅱ. The Characteristics of Hangul

To develop our method, we studied data-compression methods that minimize the bits in sending characters and then worked to improve the transmission speed of the data [2, 5, 6, 7].

One of the methods is the bit-mapping technique, which has a great compression effect when there are many spaces in the source symbol stream. The bit-mapping technique works as follows. In the source

*호서대학교 전자계산학과
접수일자 : 1995년 5월 4일

symbol stream $S=\{s_1, s_2, \ldots, s_q\}$, after determining the total number of all symbols, we created a one-byte bit-map zone part. If the source symbol is a blank, the bit-map zone corresponds to 0 ; otherwise, it corresponds to 1. Next, we created the EBCDIC with the right side of the bit-map zone. We proceeded with this method until the last symbol. In the source symbol stream $S=\{s_1, s_2, \ldots, s_q\}$, if s_q is not the last 8th symbol, (i.e., $q \neq 8*i$, where $i \geq 1$ and i is an integer), the remaining part of the bit-map zone is regarded as blanks. We represented the average length of code, such as source symbol $S=\{s_1, s_2, \ldots, s_q\}$, as follows :

$$L=\sum p(s_i) I(s_i),$$

where $p(s_i)$ is the probability of symbol si and where $I(s_i)$ obtains an amount of information equal to

$$I(s_i)=\log_2 \frac{1}{p(s_i)} \text{ bits.}$$

The average amount of information per source symbol is called the entropy H(s). It is as follows :

$$H(s)=\sum p(s_i) \log_2 \frac{1}{p(s_i)} \text{ bits :}$$

that is, $H(s) \leq \log_2 q$ (q : the number of source symbol). However, since $I(s_i)$ has to be an integer, the average length of code for an arbitary symbol S_i is as follows :

$$H(S) \leq l(s_i) < H(S)+1.$$

The bit-mapping method deals with fixed-length codes. Next, we consider the compact binary code method, which deals with the variable code lengths of different symbols. In this method, the source symbol that occurs infrequently in the text represents a long code length whereas the symbol that occurs frequently represents a short code length. The binary compact code method compresses the data to 58% of its original length. This method, however, has a major disadvantage. If the source symbols have many blanks, or there are many symbols in the text that occur frequently, this method requires longer code lengths in total. Despite this shortcoming, the binary compact code technique is the best [2].

In order to design the shortest code of entropy, we adopted the variable length code. We found frequency with which each Hangul symbol apperars in the text, and then developed a compact code, which has a different average length of codes.

In the case of Hangul, we had to consider at least 24 states since we were using Hangul's 24 basic characters. There is no tri state conditional probability that treats one symbol with Chosong(initial symbol) pair-consonant, double-vowel and Jongsong(final symbol) double-consonant. Also, we can use the graph such as in Fig. 1 [6], thus we have to achieve the efficient code.

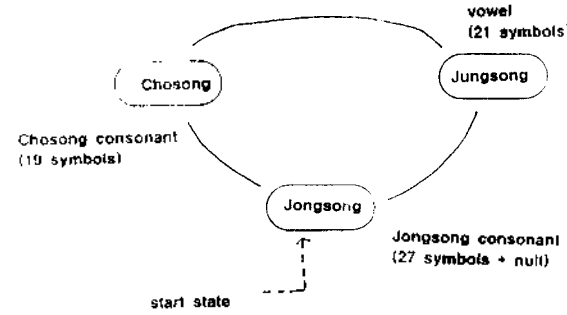

Fig. 1 Tri State transition graph

III. Parallel Hangul Coding Method

To begin with, let us think about the sequential dynamic compact method as follows. Given non-negative weights($w_1, w_2, \ldots, w_n$), we can use the well-known algorithm of Huffman's code to construct a binary tree with n external nodes and n-1 internal nodes, where the external nodes are labeled with weights($w_1, w_2, \ldots, w_n$) in increasing/decreasing order. Huffman's tree has the minimum value of $w_1 l_1 + \ldots + w_n l_n$ over all such binary trees, where l_i is the level at which wj occurs in the tree. Binary trees with n external nodes are in one-to-one correspondence with sets of n strings or {0, 1}. For example, the binary tree in Fig. 2 corresponds to the minimal code {0, 10, 110, 111}. In Fig. 2, Huffman's method combines the two smallest weights w_i and w_j(the characters that have the lowest probabilities to appear), then replaces them with their sum w_i+w_j, and repeats this process until only one weight is left. In this situation(see Fig. 3), there is no way to distinguish weight 6 associated with symbol A from weight 6 associated with symbol C and D. This procedure, therefore, may form two different trees(Fig. 2 and Fig. 3), depending upon where the weight 6 that is associated with '2+

4=6' is placed. Both trees are optimum for the given weights, since

$$6\times1+5\times2+4\times3+2\times3=2\times4+2\times2+5\times2+6\times2.$$

We call this method a dynamic compact code [1]. This procedure reforms Huffman's tree dynamically in order to reduce the height of the tree. If weight 6 associated with A increases to 7, Fig. 2 is better; but, if weight 2 associated with D increases to 3, Fig. 3 is better. In the average case, however, Fig. 3 is better even though it has some advantages and disadvantages [1].

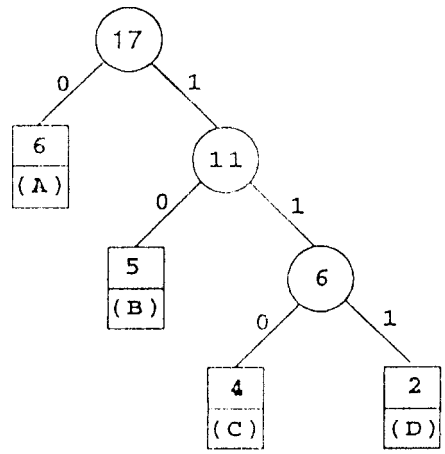

Fig. 2 Huffman tree

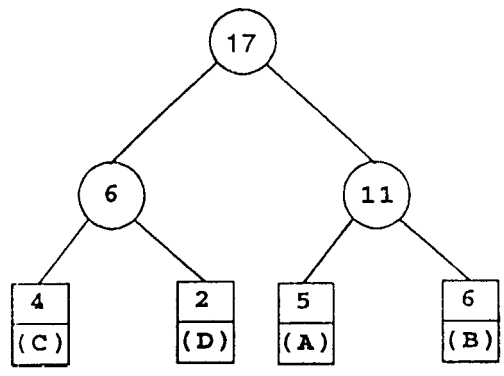

Fig. 3 Dynamic tree

To construct the Huffman tree, we must complete several steps. First, we investigate the probabilities of each symbol in the context in order for it to correspond to the character. This exercise proves that the statistical data of 1,150 Korean characters from arbitary text is suitable for this purpose.

Given the data structure above, it is not difficult to design pseudo-algorithms of the binary code tree as follows.

```
procedure binarycodetree(float p)
/* the source S with symbols {S1, S2, ..., Sq} and
   symbol probabilities {P1, P2, ..., Pq} */
begin
  (1) Let the symbols(except blank symbol) be ordered so that
      P1 ≥ P2 ≥ ... ≥ Pq
  (2) We assigned the words 0 and 1 to the last sequence
  (3) Combine the last two symbols of S into one symbol
  (4) Search back from the last sequence to the original sequence through the reduced sources
  (5) Repeat (2)-(4) until there left only two symbols codes
end
```

The total time for the procedure binarycodetree requires O(n log n) to construct the binary code tree. Step 1 requires O(n log n), which is the time complexity of the best sorting algorithm such as Quicksort or Mergesort [3]. Step 2 requires O(log n), which constructs the tree. Step 3 takes a constant time; and Step 4 takes O(l), where l is the level of the tree.

We construct a dynamic Huffman tree from the binary code tree as follows [1].

```
procedure dynamictree
begin
  (1) Represent a binary code tree with weights in each symbol
  (2) Maintain a linear list of symbols, in nodecreasing order by weight
  (3) Find the last symbol in this linear list that has the same weight as a given symbol
  (4) Interchange two subtrees of the same weights
  (5) Increase the weight of the last node in some block by unity
  (6) Represent the correspondence between letters and external symbols
end
```

This procedure requires O(n); that is, a binary code tree is constructed in steps 1 and 2 in the same

manner as for the above procedure binarycodetree. Step 3 takes O(log n), which traverses the tree. Steps 4 and 5 require O(1), which updates an element at level l of the tree and step 6 requires O(n). Together, the steps require an overall O(n) time.

Next, we describe the parallel method referred to earlier as the Parallel Hangul Dynamic Coding Method(PHDCM). PHDCM has three phases which compress the source symbols. In the first phase, we construct the binary code tree from raw source symbols, each of which has a probability. Before constructing the binary code tree, we consider the number of processors that are going to be used on the machine. In this situation, there are three cases. P, the number of processors, is less than, equal to, or greater than the number of symbols at level l, which contains either all the symbols or part of the symbols.

*If P is greater than or equal to the number of symbols at level l, then each processor at level i is con*nected to a single parent processor at level i-1 and to each of its two child processors at level i+1, except for the root processor at level 0(which has no parent) and the leaf processor at level d-1(which has no children). If P is less than the number of symbols at level l, then each processor at level i can be connected to either the same or different parent processor. Afterward, using the processor, we construct the binary code tree described in the preceding section. Let us consider step 1 in the procedure binarycodetree. In that case, we will use the parallel algorithms to sort the sequence $S=\{x_1, x_2, \ldots, x_n\}$ of distinct probabilities in increasing order [3]. This method requires n^{1-e} processors, where $0<e<1$ runs in $O(n^e \log n)$ time. In the first step of PHDCM, we produce the code using the same method as the parallel tree construction. It requires O(log n), which supports the code.

The second phase is analogous to the first. The second phase requires only exchanging the two subtrees of the same weights the different processors have. It is quite simple to implement. This phase, (that is, step 2 of PHDCM) requires O(1) to update an element at level l of the tree.

In the third phase, we encode or decode the text data from the dynamic octal-compact mapping code. In this phase, each processor reads the text data and we assign the proper code. The third step of the PHDCM method is implemented by O(n/p) time in each processor. The PHDCM pseudo-algorithm is as follows.

```
procedure PHDCM
/* n : the size of text data,
   p : the number of processor */
begin
/* first step */
 (1. a) Parallel quicksort using each probability
 (1. b) for (traverse from the root to leaves) do in parallel
            We assigned each processor's word 0 or 1
            Search previous two symbols of S which were
                combined as one symbol
        allfor
/* second step */
 (2) If we choose two processors which have same
     weights and different level, then they exchange.
     If we choose two processors, one which exists a
     higher level and the other which has a same
     weights with one's leaf, then they exchange one
     processor's leaf with the other's subtree.
/* third step */
 (3) for(i=p*⌈n/p⌉ to ((p+1)*⌈n/p⌉)-1) do in parallel
         p-read(one character in text data)
         we assigned the proper code in dynamic
         compact code
     allfor
end
```

Ⅳ. Experimental Results

To implement the PHDCM on MasPar, we tested randomly generated text sentences with various distributions. To properly use the tri-state transition of Fig. 1, we have to know the occurrence probability of each state's symbol. For purposes of this paper, we randomly extracted about 1,150 characters from a Hangul text. In case of applying other data with probabilities, it represents the similar effect since the mth-order Marcove source has an advantage [2, 6]. The probability of each symbol was computed to create the statistical data used in the preceding section. The result of the dynamic compact tree is shown in Figures 4, 5, and 6, which represent Chosong(initial symbol), Jungsong(middle symbol), and Jongsong(final symbol), respectively in Korean characters. Although these figures show only one example, other examples may exist which have the same mathematical characteristic [2, 6]. Tables 1, 2, and 3 summarize these results.

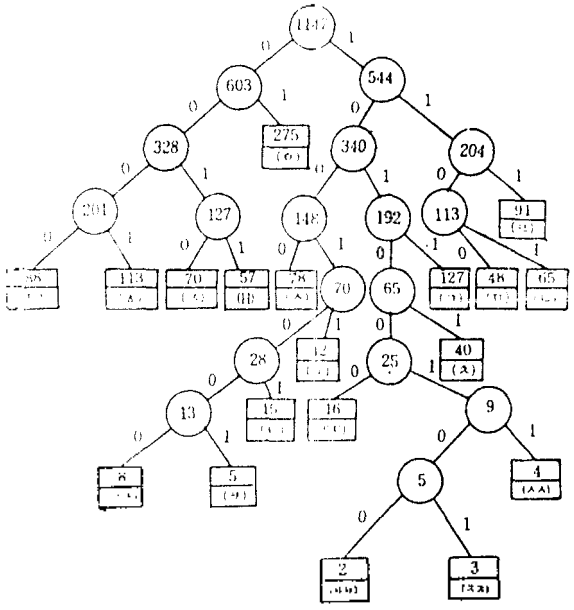

Fig. 4 Dynamic code tree for Chosong

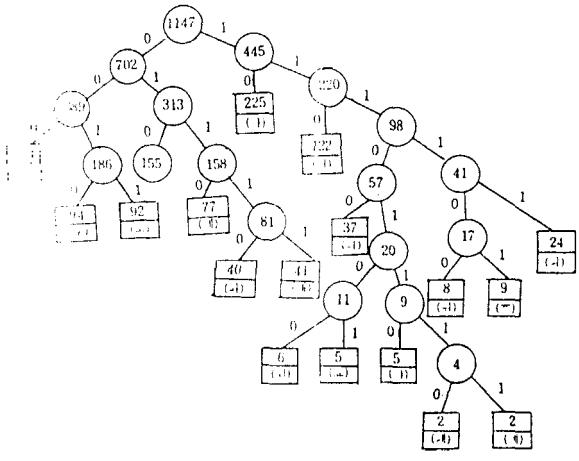

Fig. 5 Dynamic code tree forJungsong

In tables 1, 2, and 3, the total sum of $\sum$ (probability × length) means the average code length L which represents one symbol in each state. In order to represent a Korean character, the average code length is 9.2879 bits such as $\sum$ (probability × length) ; that is, 3.48238 (Chosong) + 3.398339 (Jungsong) + 2.40714 (Jongsong).

The size of the sentences to be compressed ranged from 0.01 million to 10 million symbols. Experiments were conducted using each of 1, 2, 4, 8, 16, 32, and 64 processors on the MasPar machine. Each data point presented in this section was obtained from the average of one program's execution. Each processed 10 million characters.

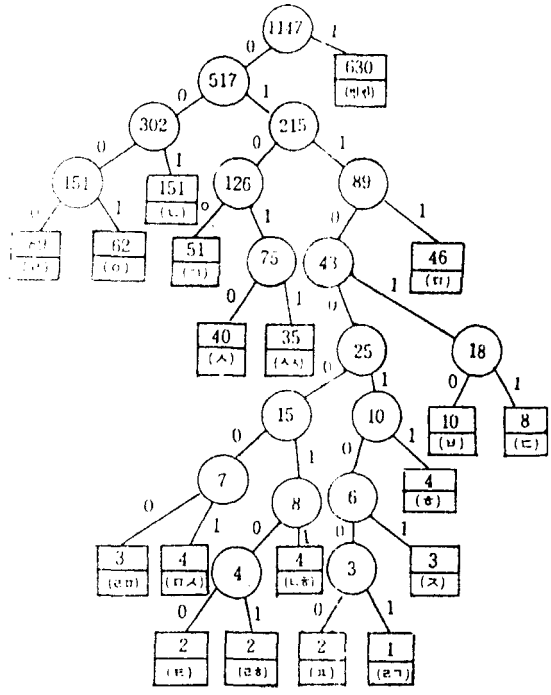

Fig. 6 Dynamic code tree for Chongsong

We have developed a program that provides the optimal sequential DCM (Dynamic Compact Mapping). The time was used on 1 processor. It needs the speedup [3] which evaluates a new data-compression method for a problem. It is reasonable to assume that the time of data compression using sequential DCM is one PE :

$$t_{pe}(n) = c\ n \log n,$$

where c is a constant independent of size n. Sequential times for lists of more than 0.2 million elements were calculated using the formula.

$$t_{pe}(n) = \frac{n \log n}{100{,}000 \log 100{,}000} * t_{pe}(100{,}000),$$

where 0.02 million $\leq n \leq$ 10 million and $t_{pe}(100{,}000) =$ 0.73 seconds. Note that if one uses this formula to compute $t_{pe}(200{,}000)$, the result is almost a perfect match with the corresponding experimental time.

Table 4 shows the time required to compress using PHDCM, and Fig. 7 plots the speedups achieved. As the problem size increases, the task granularity increases. Offsetting the overheads of the algorithms results in better speedup. Compression of 10 million text data with 64 processors yielded a 49.314-fold speedup, compared with one processor. This method was implemented in each processor's local memory.

Global memory was used to communicate the code.

Table 1. Chosong

symbol	frequency	probability	code	ength	prob. X length
ㄱ	127	0.11072	111	3	0.33216
ㄲ	8	0.00679	1001000	7	0.04879
ㄴ	65	0.05667	1101	4	0.22668
ㄷ	88	0.07672	0000	4	0.30688
ㄸ	16	0.01359	101000	6	0.0837
ㄹ	91	0.07933	111	3	0.23799
ㅁ	48	0.04185	1100	4	0.1674
ㅂ	57	0.04969	0011	4	0.19876
ㅃ	2	0.00174	10100100	8	0.01392
ㅅ	78	0.06801	1000	4	0.27204
ㅆ	4	0.00349	1010011	7	0.0238
ㅇ	275	0.23976	01	2	0.47952
ㅈ	70	0.06103	0010	4	0.24412
ㅉ	3	0.00262	10100101	8	0.02096
ㅊ	40	0.03487	1101	4	0.13948
ㅋ	5	0.00436	1001001	7	0.03052
ㅌ	15	0.01308	100101	6	0.07848
ㅍ	42	0.03662	10011	5	0.1831
ㅎ	113	0.09852	0001	4	0.39408
total	1147	1.0			3.48238

Table 2. Jungsong

symbol	frequency	probability	code	ength	prob. X length
ㅏ	225	0.19616	01	2	0.39232
ㅐ	41	0.03575	01111	5	0.17875
ㅑ	5	0.00436	1110110	7	0.03052
ㅓ	122	0.10637	110	3	0.31911
ㅔ	77	0.06713	0110	4	0.26852
ㅖ	2	0.00174	11101111	8	0.01392
ㅗ	92	0.08021	0011	4	0.32084
ㅚ	24	0.02092	11111	5	0.1046
ㅘ	40	0.03487	01110	5	0.17435
ㅙ	2	0.00174	11101110	8	0.01392
ㅛ	5	0.00436	1110101	7	0.03052
ㅜ	94	0.08195	0010	4	0.3278
ㅝ	6	0.00523	1110100	7	0.03661
ㅞ	9	0.00785	111101	6	0.0471
ㅟ	37	0.03226	11100	5	0.1613
ㅣ	155	0.13514	010	3	0.40542
ㅡ	203	0.17699	000	3	0.53097
ㅢ	8	0.00697	111100	6	0.04182
total	1147	1.0			3.39839

V. Conclusion

This paper suggests an efficient coding method in parallel to be applied for Korean characters using a three-state transition graph. The suggested method represents 9.28781 bits per Korean character. In comparison, the binary compact code which is known as the best so far, represents 10.37505 bits per Korean character [6]. In other words, we compressed about 3.5 bits per Korean character. When we applied this method to the parallel machine such as the MasPar, it achieved a 49.314-fold speedup. For this run, we used the English alphabet in one-to-one correspondence with Korean characters since the MasPar computer has no Korean characters.

In conclusion, PHDCM reduces redundancy so that

Table 3. Jongsong

symbol	frequency	probability	code	ength	prob. X length
ㄱ	51	0.44446	0100	4	0.17784
ㄴ	151	0.13156	001	3	0.39495
ㄷ	8	0.00679	011011	6	0.04182
ㄹ	89	0.07759	0000	4	0.31036
ㄺ	1	0.00087	011001001	9	0.00783
ㄻ	3	0.00262	01100000	8	0.02096
ㄼ	2	0.00174	011000101	9	0.01556
ㅁ	46	0.0401	0111	4	0.1604
ㅂ	10	0.00872	011010	6	0.05232
ㅄ	4	0.00349	01100001	8	0.02792
ㅅ	40	0.03487	01010	5	0.17438
ㅆ	35	0.03052	01010	5	0.1526
ㅇ	62	0.05406	01011	4	0.21624
ㅈ	3	0.00262	01100101	8	0.02096
ㅌ	2	0.00174	011000100	9	0.01566
ㅍ	2	0.00174	011001000	9	0.01566
ㅎ	4	0.00349	0110011	7	0.02443
	630	0.54926	1	1	0.54926
ㅀ	4	0.00349	01100011	8	0.02792
total	1147	1.0			2.40714

Table 4. Time to compress using PHDCM (unit : second)

n PE	1	2	4	8	16	32	64
100,000	0.73	0.464	0.194	0.083	0.038	0.0216	0.0258
200,000	1.26	0.905	0.436	0.205	0.083	0.0345	0.0402
400,000	2.67	1.997	0.942	0.457	0.222	0.1045	0.0805
800,000	5.19	4.017	2.273	1.045	0.457	0.228	0.1545
1,000,000	6.49	–	2.360	1.071	0.574	0.263	0.1882
2,000,000	12.87	–	–	2.339	1.288	0.5627	0.3715
4,000,000	25.51	–	–	–	2.495	1.075	0.6537
8,000,000	60.51	–	–	–	–	2.468	1.368
10,000,000	64.01	–	–	–	–	–	1.298

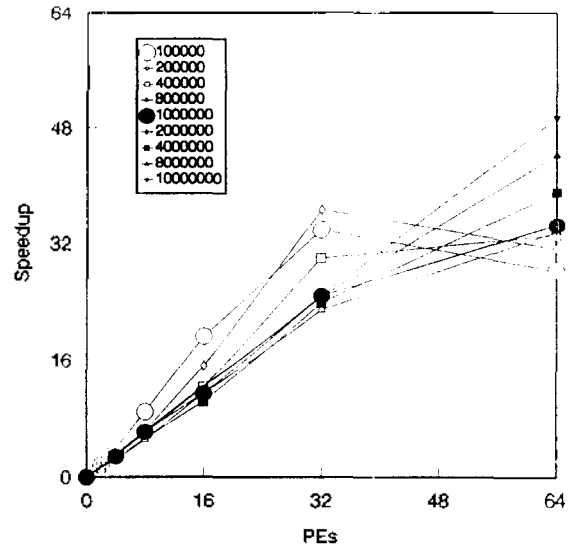

Fig. 7 Speedup of PHDCM

we can send and receive more data with a minimal number of bits. Error-detection problems on the transmission line were not considered in this research.

Acknowledgements

Dr. Kundu supply me with constructive suggestions for improving this paper. And LSU provided me with using MasPar Machine.

본 논문은 1994년도 호서대학교 학술연구조성비에 의한것임.

References

1. Kunth, Donald E., "Dynamic Huffman Coding", Journal of Algorithm, vol. 6, no. 2, pp. 163-180, June, 1985.
2. Abramson, Norman, Information Theory and Coding, McGraw-Hill, 1963.
3. Akl, Selim G., Parallel Sorting Algorihtms, Academic press, 1985
4. Bookstein, A. and Klein, S. T., "Is Huffman Coding Dead", Proceedings of Data Compression, IEEE, p. 464, 1993.
5. Kim, K. T. and Min, Y. S., "A Study on the Composition of Compact Code using OCM", Journal of KCI, vol. 9, no. 3, pp. 103-107, 1984.
6. Kim, K. T. and Min, Y. S., "A Study on an Efficient Coding of Hanguel", Journal of KCI, vol. 14, no. 6, pp. 533-641, 1987.
7. S. Roman, Coding and Information Theory, Spring-Verag, 1992.
8. Yong Sik Min, "PDOCM : Fast Text Compression on MasPar Machine", Journal of the Acoustical Society of Korea, Vol. 14, No. 1, pp. 40-47, 1995.

▲민 용 식(Min, Yong Sik)
현재 : 호서대학교 전자계산학과 부교수
한국음향학회지 제14권1호