۲×۰/۴=۰/۸ امتیاز

کنفرانس

Fifth Annual
International

Phoenix CONFERENCE on COMPUTERS and COMMUNICATIONS '86

March 26-28, 1986
SunBurst Resort Hotel & Conference Center
Scottsdale, Arizona

1986 Conference Proceedings

Sponsored by:
Institute of Electrical and Electronics Engineers

In Conjunction with:
IEEE Computer Society
IEEE Communications Society
Phoenix IEEE Section and
Arizona State University

IEEE Computer Society Order Number 691
ISBN 0-8186-0691-6

IEEE COMPUTER SOCIETY

IEEE THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC

COMPUTER SOCIETY PRESS

NOTES ON PARALLEL COMPUTATION FOR STRING TRANSFORMATION PROBLEMS

KENNETH WILLIAMS, Western Michigan University

M.R. MEYBODI, Ohio University

ABSTRACT

Sorting, matrix transposition in computer memory, string rotation and many other common tasks fall into the class of string transformation problems. This paper provides observations on the amount of information necessary, as well as the processing time required, for the parallel processing of string transformations.

INTRODUCTION

This paper deals with parallel processing of those problems for which the input can be expressed as a set of character strings and the output is a reordering (with certain modifications possible) of the same symbols. Simple examples include string reversal, matrix transposition and conversion of infix notation to postfix notation. Considerably more complex problems may be formulated. The basic approach to a string transformation problem is finding a function that maps the position of any symbol in an input string into its corresponding position in the output string. Concurrency can be achieved by simultaneous computation of positions of different symbols in the output string using different processors.

A classification of problems with related examples based on the amount of information necessary to find the function is presented. It is shown that the problem classification system can be generalized.

Once the output function is determined, it is noted that with enough processors the actual function assignment is an O(1) operation. It is shown that determination of the output function, although usually simple, may not even be computable.

In addition, it is established that outside of the general classification system some types of string transformation problems depend only on knowledge of local information within the input string. In certain cases an information reduction process can be defined where, although a great deal of information is necessary to process portions of the input, the amount of information is reduced for the processing of other input symbols.

BASIC CONCEPTS

It is well known that parallelism may be used to increase the speed of certain computations. Among many references on the subject, one may especially consider [2], [4], [5], [8], and [9]. String transformation problems as a subject for parallel computation were first considered in [6]. The basic concept for the problems is finding a function δ that maps the position of any character in an input string into its corresponding position in the output string. Since the output string represents a permutation of the input string for a given problem incident, function δ is a bijection from N to N where N is the set of integers from 1 to n, and n is the length of the string. This function reflects the algorithm that describes the transformation to be implemented. Parallelism can be obtained by determining the output positions of different symbols at the same time.

As an example, consider the problem of reversing a string of characters. The obvious sequential algorithm has complexity O(n). Using function δ, given below, a parallel computer with P processors can do the reversal in $\lceil n/p \rceil$ time. This will be an O(1) operation if n processors are available. For α_i the ith character in the input string:

$$\delta(\alpha_i) = \begin{cases} n-i+1 & \text{if } 1 \leq i \leq n \\ \text{undefined} & \text{otherwise} \end{cases}$$

This paper first defines the notation and basic premises and then provides a set of examples to illustrate the approach. Through the examples it becomes evident that there is a natural classification of problem types based on the amount of information required by δ. A generalization of the classification system is developed.

Let string $I = \alpha_1\alpha_2\ldots\alpha_p$ be the input string and $O = \beta_1\beta_2\ldots\beta_p$ be the output string for a given algorithm. Each symbol may consist of one or more characters.

For each $\alpha \in \Pi$ define:

$\text{inpos}(\alpha) = i$ where $\alpha = \alpha_i$, and

$\text{outpos}(\alpha) = j$ where $\alpha = \beta_j$.

Let δ be the function that defines outpos(α). Then the range of δ is given by Range (δ) = $\{1,\ldots,p\}$. However, the required domain of δ may be different for different altorithms.

Having defined function δ, we can determine positions of all the symbols in the output string in $L + \lceil n/P \rceil T$ time, for T the time needed to compute function δ, P the size of the parallel processor (i.e. the number of processing elements that the parallel processor contains), n the length of the input string and L the amount of time needed to gather the information required by function f (the "information gathering time").

Problems of this type may be classified into several cases based on the amount of information required by function f.

CLASSIFICATION AND EXAMPLES

Case I

Outpos (α) is a function of the output position of α only. That is

$$\text{outpos}(\alpha) = f(\text{inpos}(\alpha))$$

Example 1: Identity relationship.

$$\alpha_1\alpha_2\ldots\alpha_n \Rightarrow \alpha_1\alpha_2\ldots\alpha_n.$$

$$f(\text{inpos}(\alpha)) = \text{inpos}(\alpha)$$

Example 2: $\alpha_1\alpha_2\ldots\alpha_n \Rightarrow \alpha_2\alpha_1\alpha_4\alpha_3\ldots\alpha_n\alpha_{n-1}.$

$$f(\text{inpos}(\alpha)) = \begin{cases}\text{inpos}(\alpha)+1 \text{ if inpos}(\alpha) \text{ is odd}\\ \text{inpos}(\alpha)-1 \text{ if inpos}(\alpha) \text{ is even}\end{cases}$$

Note that if we have n processors, each operating on exactly one of the input symbols, $\forall\alpha$, outpos(α) can be computed in O(1) time. If we have P processors, this will require O(n/P) time.

Case 2

Outpos(α) is a function of inpos(α) and n only. That is

$$\text{outpos}(\alpha) = f(\text{inpos}(\alpha), n)$$

Example: String reversal.

$$\alpha_1\ \alpha_2\ldots\alpha_n \Rightarrow \alpha_n\ldots\alpha_1.$$

$$f(\text{inpos}(\alpha),n) = n - \text{inpos}(\alpha) + 1$$

As in Case 1, once the information gathering time is complete, this type of function can be calculated in O(n/P) time when we have P processor and in O(1) time given n processors.

Case 3

Outpos(α) is a function of inpos(α) and certain other fixed information. That is

$$\text{outpos}(\alpha) = f(\text{inpos}(\alpha), t_1,t_2,\ldots,t_m).$$

Example: Matrix transposition. We assume that the elements of the matrix are stored in row major order. That is, the elements are stored in lexicographic order by index with the row index as the major key and the column index as the minor key. Using row major order, the two dimensional array $A(1{:}l_1,\ 1{:}l_2)$ can be interpreted as l_1 rows: row_1, $\text{row}_2,\ldots,\text{row}_{l_1}$ with each row consisting of l_2 elements. One may observe that inpos(α), for α= A[i,j], is displaced from the base of the array by the amount $l_2(i-1) + j$ and, therefore, location (A[i,j]) is given by $l_2(i-1) + j$ + base. From the above, we note that $f(\text{inpos}(\alpha),l_1,l_2)$ for α= A[i,j] is given by:

$$i = \lfloor(\text{inpos}(\alpha)-1)/l_2\rfloor + 1 \text{ in A.}$$

and $j = ((\text{inpos}(\alpha)-1)\text{MOD } l_2) + 1$ in A.

To compute the transpose, A[i,j] will become $A^T[j,i]$ which will occupy the $l_1(j-1) + i$ position. (The number of rows in $A^T = l_2$, the number of columns $= l_1$.)

Therefore,

$$f(\text{inpos}(\alpha),\ l_1,l_2) = l_1((\text{inpos}(\alpha)-1)\text{MOD } l_2) + \lfloor(\text{inpos}(\alpha)-1)/l_2\rfloor + 1$$

So, once information gathering is complete, the transpose of a matrix can be computed in O(1) time if $l_1 * l_2$ processors are available and in O(n/P) time when P processors are available.

Case 4

Outpos(α) is a function of α only. That is

$$\text{outpos}(\alpha) = f(\alpha)$$

Example: Input sequence $\alpha_1\alpha_2\ldots\alpha_n$ is a random ordering of the integers 1, ..., n. For the output we want them in sequential order. The outpos(α) = $f(\alpha) = \alpha$.

This can be calculated in O(n/P) time when P processors are available.

Case 5

Outpos(α) is a function of the input position of α and of the entire input string. That is

$$\text{outpos}(\alpha) = f(\text{inpos}(\alpha),I)$$

Example 1: Sorting. The problem of sorting is one of determining a permutation that arranges a set of symbols in a particular order. We can use the following function to compute the output position of a symbol.

$$\text{For } \alpha = \alpha_i,\ f = \begin{cases} n_i + 1 & \text{for } 1 \le i \le n\\ \text{undefined} & \text{otherwise}\end{cases}$$

where n_i is the number of symbols less than the ith symbol in the input string plus the number of symbols equal to the ith symbol that preceed the ith symbol in the input string.

Muller and Preparata [7] have shown that each value of n_i can be determined in O(log n) time (information gathering time) using n log n processors. Once each value of n_i is known, the final positions can be computed in O(1) time using n processors.

Example 2: Consider a string of symbols of length n that consists of symbols belonging to three different groups: G1, G2, and G3. We want an algorithm that transforms the input string into an output string in which all symbols from G1 are placed at the beginning, all from G3 at the end and all from G2 in the middle of the string. The order of symbols within each group for output should be the same as their order in the input. For $\alpha = \alpha_i$, $f(\text{inpos}(\alpha), I)$ may be given by:

For $\alpha \in G1$:

> f = (number of symbols that belong to group G1 and precede the ith symbol in the input string) + 1

For $\alpha \in G2$:

> f = (number of symbols that belong to group G1) + (number of symbols that belong to group G2 and precede the ith symbol in the input string) + 1

For $\alpha \in G3$:

> f = (number of symbols that belong to group G1) + (number of symbols that belong to group G2) + (number of symbols that belong to G3 and precede the ith symbol in the input string) + 1

Information needed to compute f can be obtained in $O(n)$ time with a single sequential pass through the input or by each processor adding 1 to an accumulator from a set of common accumulators with one accumulator to represent each of the groups of symbols. Once this information is obtained, the position of symbols in the output string can be determined in $O(1)$ time if n processors are available.

Example 3: Transformation of infix notation into postfix notation. A solution to this problem is given by Dekel and Sahni [1]. In [1], the Shared Memory Model (SMM) is used as a model of computation. Using this model of computation it is shown that the proper positions of symbols in the output string can be computed in $O(\log^2 n)$ using n processors. To ensure that every input symbol has a corresponding position we may assume that the input expression is free of parentheses or we may extend the class of problems being considered to allow for some of the input symbols to not be repeated in the output. (Their outpos may be regarded as 0.)

Case 6

> Outpos(α) is a function of the input position of α, of certain fixed information and of n. That is
>
> $$\text{outpos}(\alpha) = f(\text{inpos}(\alpha), t_1, t_2, \ldots t_m, n)$$

Example: Rotate the input string r symbols to the right.

$$\text{outpos}(\alpha) = (\text{inpos}(\alpha) + r - 1) \text{ MOD}_n + 1$$

Case 7

> Outpos(α) is a function of α, of the input position of α, and of some additional local information concerning α's neighbors and their input positions. That is
>
> $$\text{outpos}(\alpha) = (\alpha_{i-k_1} \ldots \alpha_{i+k_2},$$
> $$\text{inpos}(\alpha_{i-k_1}), \ldots, \text{inpos}(\alpha_{i+k_2}) \text{ for}$$
> $$\alpha = \alpha_i \text{ and constants } k_1 \text{ and } k_2).$$

Example: The output sequence is to be the symbols of the input sorted in a "local" manner. For instance, perhaps the first five input symbols are to be the first five sorted output symbols, then the next five etc. Considerations for local sorting complexity are similar to those for general sorting as described in Case 5. In fact, one might regard this as repeated instances of the general sorting case.

The amount of information required for this case is encompassed by knowing I rather that just the local information, if I is available.

Case 8

> Outpos(α) is a function of n and of outpos(β) for all β such that inpos(α) $\neq$ inpos(β). That is
>
> $$\text{Outpos}(\alpha) = f(n, \text{outpos}(\beta) \ \forall \beta \text{ with } \text{inpos}(\alpha) \neq \text{inpos}(\beta))$$

Example: The output is a random reordering of the input. We cannot assign output index j to α unless we are sure that no other input symbol has been assigned the same output index.

GENERAL CLASSIFICATION

Note that there is a set of information types, F, consisting of 6 different types of information required for the parallel computers which are used to calculate outpos(α). Let $F = \{F_1, F_2, \ldots F_6\}$ where:

$F_1 = \alpha$

F_2 = inpos(α)

$F_3 = n$

F_4 = fixed information $T_1, T_2, \ldots, T_m$

$F_5 = I = I_1, I_2, \ldots, I_p$

F_6 = outpos(β) $\forall \beta$ such that inpos(β) $\neq$ inpos(α)

(i.e. F_6 means that to calculate f, for a given α, a processor must know outpos of all other input symbols - worst case.)

This classification system does not take the possible necessity for knowledge of local information into account. Knowledge of F_5, I, provides knowledge of any necessary local information. Also, knowledge of F_5 implies knowledge of F_3, n, since if a processor knows the complete input string it can count the symbols to find n. Sometimes the fixed information, F_4, may imply n (as in the case of the matrix transposition example) but it may not. In fact one could think of F_4 as consisting of two subcases. In one case the "fixed" information is only fixed as it applies to a given string. In the second case it is fixed for all possible strings of any length. (In this case, the fixed information could simply be incorporated into the definition of f.)

A general classification scheme can then be defined where each given problem will require an amount of information which is an element of 2^F, the power set of F. Some of the subsets of F are achievable (i.e. there will be problems which require precisely that much information - worst case - for each processor) and some will not be achievable. The following observations indicate certain achievable and not achievable subsets.

Observation 1

There are achievable problems that require precisely each of the following types of information:

a) F_1

b) F_2

c) F_6

Note that examples of (a) and (b) have been previously presented. Consider the following example for (c) where outpos(α) = $f(F_6)$. For a given input symbol, α, when given F_6, n can be calculated.

Define set A = {outpos(β) such that inpos(β) $\neq$ inpos(α)}.

Then n = $|A| + 1$

Define set B = $\{1,2,\ldots,n\}$

Define set C = B-A.

C will contain one element, the "missing output position" from set A.

Let $x(\alpha)$ = the integer in C.

So we have essentially a "don't care" or random permutation of input symbols produced, where to produce any one it is necessary only to know the others.

Observation 2

Every achievable problem must use information from F_1, F_2 or F_6.

Assume there was a problem where computing f required only information which was a subset of $\{F_3,F_4,F_5\}$. Then, for a given input string, f would compute exactly the same outpos for each input symbol since there are no variables involved. (i.e. For a given string F_3, F_4 and F_5 are fixed and do not change for any of the input symbols.)

HARD PROBLEMS

Note that for all the problems and examples described thus far, f can be computed in constant time or in linear time based on n. This is not always the case. The following string transformation is not even computable. Let $w_1,w_2,\ldots$ be an enumeration of all strings from {0,1} and $T_1,T_2\ldots$ be an enumeration (encoded into strings of binary digits) of all Turing Machines over input alphabet {0,1}. For string x composed of the digits {0,1} consider transformation:

$$f(x01) = \begin{cases} x01 \text{ if } x = w_i \text{ and } w_i \text{ is accepted by } T_i \\ x10 \text{ if } x = w_i \text{ and } w_i \text{ is not accepted by } T_i \end{cases}$$

In order for f to be computable both $S_1 = \{x01 \mid x=W_i \text{ and } w_i \in T_i\}$ and $S_2 = \{x01 \mid x=W_i \text{ and } w_i \notin T_i\}$ must be recursively enumerable. (See[3].) S_2, however, is not recursively enumerable, therefore f cannot be computed in any amount of time by any finite number of processors.

LOCALIZATION

We say a transformation can be localized if there are certain identifiable symbols in the input string which enable us to partition the input string in such a way that each partition can be transformed independently from other partitions. For instance, in an infix to postfix transformation, if we locate the symbol which will be mapped into the last position of the postfix string (this symbol will be the last output in the case of the sequential algorithm) we can decompose the input string into two substrings (subexpressions) such that the output position of every symbol in one subexpression is independent of the other subexpression.

Let the partitions on the elements of the input string be $\Pi_1,\Pi_2,\ldots,\Pi_r$. For each input symbol α in a partition associate an (i,j) pair where i = partition number containing α and j = inpos(α). We say the input string is statically partitioned if for all partitions Π_i the associated pairs are $(i,i_1),(i,i_2),\ldots,(i,i_k)$ and $i_1,i_2,\ldots,i_k$ is a sequence of consecutive integers. Otherwise we say the input string is dynamically partitioned. In the infix to postfix transformation, partitions which are induced by knowing the symbol that is mapped into the last position of the output string are static partitions, whereas partitions induced by finding the output position of the pivot element in a quicksort algorithm are dynamic partitions.

INFORMATION REDUCTION PROCESS

The concept of locality is closely related to what may be termed the information reduction process. Inspection of some of the previous examples reveals the fact that knowledge of the output positions of some of the input symbols allows computation of the output position of the remaining symbols without requiring the same amount of information that was needed by symbols whose output positions was previously computed. If knowledge of the output position of a symbol or set of symbols reduces the amount of information needed to compute the output position of a symbol or set of symbols we say information reduction has taken place. The amount of information reduction occuring as a result of knowing the output position of a symbol (or a set of symbols) can be used as a measurement to compare different algorithms used for a particular transformation.

CONCLUDING REMARKS

Approaches to develop faster algorithms to solve string transformation problems through the use of parallelism lead to a classification of problem types based on the amount of knowledge required for each parallel device. A number of initial results for problems of this type are presented. These methods may lead to faster algorithms for solving a variety of different problems. It is shown, however, that not all string transformation problems are even computable.

REFERENCES

[1] E. Dekel and S. Sahni, "Parallel Generation of Postfix and Tree Forms", ACM Transactions on Programming Languages and Systems, Vol 5, No 3, July, 1983, pp. 300-317.

[2] H.J. Flynn, "Some Computer Organizations and their Effectiveness", IEEE Transactions on Computers, C-21(a): 448-460, Sept., 1972.

[3] J. Hopcroft and J. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, 1979.

[4] D.J. Kuck, The Structure of Computers and Computation, Vol 1, John Wiley and Sons, 1978.

[5] H.T. Kung, "Synchronized and Asynchronous Parallel Algorithms for Multiprocessors". Algorithms and Complexity: New Directions and Recent Results, J.F. Traub, ed., Academic Press, 1976, pp. 153-200.

[6] M.R. Meybodi, "On Writing Algorithms for Parallel Computers", WMU Technical Report, 1984.

[7] D.E. Muller and F.P. Preparata, "Bounds to Complexities of Networks for Sorting and for Switching," Journal of ACM, 1975, pp. 195-201.

[8] H.S. Stone, "Introduction to Computer Architecture", H.S. Stone, ed., Palo Alto, Science Research Associates, 1980.

[9] H.S. Stone, "Parallel Processing with Perfect Shuffle", IEEE Transactions on Computers(2): 153-161, 1971.