۲۴۰/۸ = ۱/۷

MODELING AND SIMULATION

Volume 17

Part 3: Computers, Vision Systems, Hydrology and Fluids

MODELING AND SIMULATION CONFERENCE

MSC

Proceedings of the Seventeenth Annual Pittsburgh Conference
Held April 24-25, 1986

University of Pittsburgh School of Engineering

Published and Distributed by:

Instrument Society of America

ISA

MODELING AND SIMULATION
VOLUME 17
Part 3

Proceedings of the Seventeenth Annual Pittsburgh Conference

Held
April 24-25, 1986
University of Pittsburgh
Sponsored by
The Department of Electrical Engineering
School of Engineering
University of Pittsburgh
In Cooperation with
The Pittsburgh Sections
of the
Institute of Electrical and Electronic Engineers
and
The Instrument Society of America
and
The Systems, Man and Cybernetics Society
The Society for Computer Simulation
The International Association for Mathematics
and Computers in Simulation (formerly AICA)

Edited by

William G. Vogt
Marlin H. Mickle

Associate Editors

Library of Congress Catalog Card Number 73-85004

ISBN Part 3 0-87664-970-3
ISSN 0198-0092

Distributed by

INSTRUMENT SOCIETY OF AMERICA
67 Alexander Drive
P.O. Box 12277
Research Triangle Park, N.C. 27709
Phone: (919) 549-8411
Telex: 802 540 ISA DURM

Printed in U.S.A.
By
Technical Communication Services

ACKNOWLEDGMENTS

Conference Co-Chairmen

William G. Vogt **Marlin H. Mickle**

Conference Coordinator
J. Marquis

Committee

S. Ak
P. Barsotti
W. Boles
T. C. Chen
C. Gubala
B. Heggy
M.-L. Hsiao
H. Q. Lu
Li Luo
H. S. Mallikarjuna
L. Mears
H. Miranda
M. Nuzzo
J. Oliva
K. Tamenne
S. A. Weisberg

An international conference such as the Modeling and Simulation Conference owes its success to many people. To list each of those who helped and the manner in which he helped would likely require a volume the size of the PROCEEDINGS. But we can mention some who have been of particular help.

First, our special thanks to the Instrument Society of America for help in many ways during the past year, and particularly for editorial help.

The editors wish to express sincere thanks to Professor H. B. Hamilton, Acting Chairman of the Department of Electrical Engineering, University of Pittsburgh, for his support of this Conference in many ways.

The School of Engineering of the University of Pittsburgh has supported the Modeling and Simulation Conference in a variety of ways. Accordingly, the editors wish to thank Acting Dean Y. T. Shah.

A special thanks also to Kathy Tamenne, Sandy Weisberg and Madeline Nuzzo, each of whom helped with enthusiasm and grace. An extra special thanks to Betty Victor and Barb Dippold for help in many ways.

A final thanks goes to all the others who organized sessions, chaired sessions, operated audio-visual equipment, made signs, helped with registration and handled efficiently those many situations and problems, some of which could have meant disaster to the Conference, but each of which were satisfactorily handled in a truly professional spirit.

William G. Vogt
Marlin H. Mickle
Editors

ON DEVELOPING ALGORITHMS TO SOLVE CERTAIN CLASSES OF PROBLEMS USING PARALLEL COMPUTERS

M. R. Meybodi
Ohio University
Computer Science Department
Athens, Ohio 45701

Kenneth Williams
Western Michigan University
Computer Science Department
Kalamazoo, Michigan 49008

ABSTRACT

This paper proposes a simple approach to writing algorithms for parallel computers. The approach is suitable for those problems for which the input and output can each be expressed as sets of character strings composed of the same symbols. Examples include string reversal, matrix transposition and conversion of infix notation to postfix notation. The basic concept of this approach 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. An introductory classification of problems based on the amount of knowledge necessary to find the function is presented. A set of examples is provided for illustration. Once the output function is determined, it is noted that with enough processors the actual function assignment is an O(1) operation.

I. Introduction

It is well known that parallelism may be used to increase the speed of computation. This report suggests a simple approach to writing algorithms for parallel computers. The approach is suitable for those problems whose input and output can be expressed as a set of strings of characters over the same symbols. Such problems include string reversal and the transformation of infix notation into postfix notation. The basic concept of the approach is to find a function f that maps the position of any character in an input string into its corresponding position in the output string. Assuming the output string represents a permutation of the input string, function f 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 f , given below, a parallel computer with P processors can do the reversal in $|n/P|$ time. This will be an O(1) operation if n processors are available.

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

α_i is the ith character in the input string.

In this report we first define the notation and basic premises and then give a set of examples to illustrate the approach. The examples are designed to provide insight into the overall concept rather than provide optimal algorithms.

II. Notation

Let set $I = \{I_1, I_2, \ldots, I_p\}$ where each I_i is a character string of length n_i over set Σ, Set $O = \{O_1, O_2, \ldots, O_q\}$ where each O_i is a string of length m_i over Σ. I is the input and O is the output for a given algorithm. Σ is any finite set of symbols. Each symbol may consist of one or more characters. Let $I_i = \alpha_{i1}\alpha_{i2} \ldots \alpha_{in_i}$ and $O_i = \beta_{i1}\beta_{i2} \ldots \beta_{im_i}$. Also

$$\bigcup_{i=1}^{p} \bigcup_{j=1}^{n_i} \alpha_{i_j} = \bigcup_{i=1}^{q} \bigcup_{j=1}^{m_i} \beta_{i_j} \quad \text{and } (C_2) \quad \sum_{i=1}^{q} m_i = \sum_{i=1}^{p} n_i$$

further define our basic assumptions and terminology:

et $N = \sum_{i=1}^{q} m_i$. Let $\Pi = \bigcup_{i=1}^{p} \bigcup_{j=1}^{n_i} \alpha_{i_j}$. Then $|\Pi| = N$.

ne that each symbol in Π is unique. Of course, in practice, each may only be o the extent that it occurs at a unique position in I and in O.)

each $\alpha \in \Pi$ define:

os(α) = i,j) where $\alpha = \alpha_{i_j}$, outpos(α) = (i,j) where $\alpha = \beta_{i_j}$.

f be the function that defines outpos(α). Then, in general, the range of f is :

ge(f) = $\{1, \ldots, q\} \times \{1, \ldots, \mathrm{Max}(m_i)\}$ for $1 \le i \le q$.

required domain of f may be different for different algorithms. In fact, a :ation of the types of problems which may be solved for f may be developed (See) based on the amount of information that is required in the domain of f.

litions C_1 and C_2 which are imposed on the input and output strings are not always , but in order to simplify the determination of function we restrict our dis- only to those algorithms whose input and output strings satisfy these conditions.

ing defined function , we can determine positions of all the symbols in the output 1 L + $\lceil n/P \rceil$T time. In this formula T is the time needed to compute function , P ize of the parallel processor (i.e. the number of processing elements that the processor contains), n is the length of the input string and L is the amount of ied to gather the information required by function (the information gathering The algorithm used to gather this information is called the "information gathering s". Obviously improvement is achieved only if the time needed by the sequential n, S, (i.e. an algorithm for a single processor machine) is greater than T, that is S > L + $\lceil n/P \rceil$T. If P = n, then improvement can be achieved if r.

algorithm for information gathering must be carefully designed or any potential n time from parallel processing may be offset by the cost of L. In fact, the less on that needs to be gathered, the better.

sification and Examples

consider the following classification of cases for the information required by

os(α) is a function of the input position of α only. That is

outpos(α) = f(inpos(α))

: Identity relationship. $\alpha_1\alpha_2 \ldots \alpha_n \Rightarrow \alpha_1\alpha_2 \ldots \alpha_n$.

f(inpos(α)) = inpos(α)

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

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

that if we have n processors, each operating on exactly one of the input symbols, s(α), $\forall\alpha$, 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

outpos(α) = f(inpos(α), n)

Example: String reversal. $\alpha_1\alpha_2 \ldots \alpha_n \Rightarrow \alpha_n \ldots \alpha_1$.

f(inpos(α),n) = n = inpos(α) + 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 processors and in O(1) time when we have n processors.

Case 3

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

outpos(α) = f(inpos(α),t_1,t_2, ... , 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 all$_1$ rows: row$_1$, row$_2$, ... , 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 an amount of l_1(i-1) + j and, therefore, location(A[i,j]) is given by l_1(i-1) + j + base. From the above, we note that f(inpos(α), l_1, l_2) for α = A[i,j] is given by:

i = $\lfloor$(inpos(α)-1)/$l_1\rfloor$ + 1

j = ((inpos(α)-1)MOD i) + 1

To compute the transpose, A[i,j] should be stored in A^T[j,i] which will be the l_2(j-1) + i position.

Therefore,

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

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

outpos(α) = f(α).

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. Then outpos(α) = f(α) = α.

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

outpos(α) = f(inpos(α),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.

$$\ldots_i, \int = \begin{cases} n_i + 1 & \text{for } 1 \le i \le n \\ \text{undefined} & \text{otherwise} \end{cases}$$

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

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

2: Consider a string of symbols of length n that consists of symbols belonging different groups: G1, G2 and G3. We want to write an algorithm that transforms string into an output string in which all symbols from G1 are placed at the ;, all from G3 at the end and all from G2 in the middle of the string. The order ls within each group for output should be the same as their order in the input. $_i$, (inpos(α),I) may be given by:

31:
(number of symbols that belong to group G1 and precede the ith symbol in the input string) + 1

32:
(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

33:
(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

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

i: Transformation of infix notation into postfix notation. A solution to this s given by Dekel and Sahni [2]. In [2], the Shared Memory Model (SMM) is used as f computation. Using this model of computation it is shown that the proper posi- symbols in the output string can be computed in O(n $\log^2$ n) using n processors. y conditions C1 and C2 we may assume that the input expression is free of paren- we may extend the class of problems being considered to allow for some of the bols to not be repeated in the output. (Their outpos may be regarded as 0.)

os(α) is a function of α, of the input position of α, and of some additional local on concerning α's neighbors and their input positions. That is

$$\text{os}(\alpha) = \int(\alpha_{i-k_1} \ldots \alpha_{i+k_2}, \text{inpos}(\alpha_{i-k_1}), \ldots, \text{inpos}(\alpha_{i+k_2}) \text{ for } \alpha = \alpha_i$$

and constants k_1 and k_2).

The output sequence is to be the symbols of the input sorted in a "local" For instance, perhaps the first five input symbols are to be the first five sorted mbols, then the next five etc. Considerations for local sorting complexity are o those for general sorting as described in Case 5. In fact, one might regard he general sorting case where the first five input symbols are I_1, the next five c.

os(α) is a function of inpos(α) and of outpos(β) for all β such that inpos(α) ≠ That is

$$\text{os}(\alpha) = \int(\text{inpos}(\alpha), \text{outpos}(\beta)) \quad \beta \text{ with inpos}(\alpha) \ne \text{inpos}(\beta).$$

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.

For cases of this nature, it appears that it will not be possible to achieve a savings in time through the use of parallelism.

Clearly, these do not provide all the cases that one might consider, however they do provide some insight in this approach.

IV. Observations

We can make two general observations.

Theorem 1

Given a total of n input symbols, each of which receives a unique output index 1, ..., n, there are a total of n! different functions which can constitute outpos(α).

Since the output is just a permutation of the input, the number of possible functions is the number of permutations of n symbols taken n at a time which is n!.

Theorem 2

Once function outpos(α) is known, it may be applied in O(n/P) time where P processors are available.

This is trivial since outpos(α) = which just gives an integer 1,...,n. Note that for n processors this is an O(C) operation whereas for a single processor it is an O(n) operation. This alone assures us of faster algorithms through parallelism in cases where the information gathering time is reasonable.

V. Conclusion

A simple approach to the development of algorithms for parallel computation is considered. This approach may lead to new and faster algorithms for solving old problems. A general type of problem where the output is a permutation of the input has been examined and several different types of such problems, classified by the amount of information required, have been identified.

REFERENCES

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

[2] 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.

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