MODELING AND SIMULATION

Volume 17

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
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

M.A. Aburdene
S. Ak
K. Aminian
P. Barsotti
W. Boles
C.L. Brooks
F. Calzonetti
E. Cassetti
L. Chatterjee
O.T. Chen
T.C. Chen
F.D. Chichester
L. De Cola
E.S. de Llinas
D.R. Drew
L.D. Durbin
R. Duval
M. El Nokali
H. El-Ghobary
G. Elmes
M.M. Fahim
D.P. Garg
D.S. Gill
A. Giorgini

R.D. Green
C. Gubala
R. Hanham
A.M.A. Haridy
T.T. Hartley
S. Heckman
B. Heggy
R. Hoover
M.-L. Hsiao
M. Husain
R. Jackson
K.G. Janardan
R. Jensen
J.P. Jones, III
C.P. Kartha
P. Kochanowski
J.W. Kohn
L. Kohut
D.O. Koval
J. Kryspin
R.K. Krzywiec
H.Q. Lee
C.E. Lenz
P. Liang

H.Q. Lu
V.P. Lukic
L. Luo
R.C. Magel
R. Maggio
H.S. Mallikarjuna
J. Marquis
M.K. Masten
F.F. Mattar
L. Mears
G.A. Milham
H. Miranda
L.C. Monticone
C. Moore
G.W. Moser
M.A. Nasser
M. Nuzzo
J. Oliva
G. Padmanabhan
M. Paul
M.R. Perryman
K. Ping-Hung Ying
M.J. Radzicki
M. Rafiquzzaman

A. Rose
H.A. Ryaciotski-Boussalis
M.M. Sallam
V. Selman
D. Selwood
A.M. Sharaf
S. Shelmani
F.O. Simons
V. Simunek
J. Storbeck
K. Tamenne
S.W. Tarasewicz
W.E. Thompson
S.A. Thoreson
K. Tojo
T.K. Tran
L. Ubertini
R. Walker
R.H. Wang
S.A. Weisberg
M.G. Wilder
C. Wu
R.R. Yager
M. Yener
ACKNOWLEDGMENTS

Conference Co-Chairmen
William G. Vogt  Marlin H. Mickle

Conference Coordinator
J. Marquis

Committee
S. Ak  Li Luo
P. Barsotti  H. S. Mallikarjuna
W. Boles  L. Mears
T. C. Chen  H. Miranda
C. Gubala  M. Nuzzo
B. Heggy  J. Oliva
M. -L. Hsiao  K. Tamene
H. Q. Lu  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 Tamene, 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. Meybofh
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 a set 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 the 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.

1. 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. Each problem includes 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 \( 1 \) 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 \( O(n) \). Given below, a parallel computer with \( P \) processors can do the reversal in \( |n/P| \) time. This will be an \( O(1) \) operation if \( P \) processors are available.

\[
f(x_i) = \begin{cases} n+1 & \text{if } 1 \leq i \leq n \\ \text{undefined} & \text{otherwise} \end{cases}
\]

\( x_i \) is the \( i \)th 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 \( T = \{c_1, c_2, \ldots, c_m\} \) where each \( c_i \) is a character string of length \( n \) over set \( I \). Set \( O = \{o_1, o_2, \ldots, o_P\} \) where each \( o_i \) is a string of length \( m \) over \( I \). \( I \) is the input and \( O \) is the output for a given algorithm. \( I \) is any finite set of symbols. Each symbol may consist of one or more characters. Let \( I_1 = c_{11} c_{12} \ldots c_{1n} \) and \( O_1 = o_{11} o_{12} \ldots o_{1m} \). Also
Further define our basic assumptions and terminology:

Let \( B = \{1, \ldots, n\} \) and \( S = \{\alpha_1, \ldots, \alpha_j\} \) such that \( \alpha_1 < \alpha_2 < \cdots < \alpha_j \). Then \( |B| = n \).

We also assume that each symbol in \( B \) is unique. Of course, in practice, each may only be a sequence of symbols that occurs at a unique position in \( B \) and in \( S \).

**Case 1**

Given \( \alpha \in B \), let \( f(\alpha) \) be the function that defines \( \text{output}(\alpha) \). Then, in general, the range of \( f \) is

\[ f(\alpha) = \{1, \ldots, q\} \times \{1, \ldots, \max(\xi_2, \xi_3)\} \] for \( 1 \leq i \leq q \).

**Case 2**

Output(\( \alpha \)) is a function of \( \text{impos}(\alpha) \) and \( n \) only. That is

\[ \text{output}(\alpha) = f(\text{impos}(\alpha), n) \]

Example: String reversal. \( \alpha_1 \alpha_2 \alpha_3 \alpha_4 \alpha_5 \alpha_6 \alpha_7 \)

\[ \text{impos}(\alpha, n) = n - \text{impos}(\alpha) + 1 \]

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 \( P \) processors and in \( O(1) \) time when we have \( n \) processors.

**Case 3**

Output(\( \alpha \)) is a function of \( \text{impos}(\alpha) \) and certain other fixed information. That is

\[ \text{output}(\alpha) = f(\text{impos}(\alpha), \xi_1, \xi_2, \xi_3, \ldots, \xi_s) \]

Example: Matrix transposition. We assume that the elements of the matrix are stored in row major order. That is, the elements are stored in listographic 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 \( A(1:1; 1:1) \) can be interpreted as \( a_{11} \), \( a_{21} \), \( a_{31} \), \( \ldots \), \( a_{12} \), \( a_{22} \), \( a_{32} \), \( \ldots \), \( a_{1n} \), \( a_{2n} \), \( a_{3n} \), \( \ldots \), \( a_{1m} \), \( a_{2m} \), \( a_{3m} \), \( \ldots \), where each row consists of \( \xi_2 \) elements. One may observe that \( \text{impos}(\alpha) \), for \( \alpha = A(i, j) \), is displaced from the base of the array by an amount of \( j(i-1) \) and \( j(i-1) \) and \( j \) and therefore, \( \text{location}(A(i, j)) \) is given by \( j(i-1) + 1 + \) base. From the above, we note that

\[ j(i-1) + 1 \] for \( \alpha = A(i, j) \) is given by:

\[ j = ((\text{impos}(\alpha) - 1)/j) + 1 \]

To compute the transpose, \( A(i, j) \) should be stored in \( A(j, i) \) which will be the \( j(i-1) \) + \( 1 \) position.

Therefore,

\[ f(\text{impos}(\alpha), 1, 1) = j_2(\text{impos}(\alpha) - 1) + 1 \]

So, once information gathering is complete, the transpose of a matrix can be computed in \( O(1) \) time if \( j_2 \) processes are available and in \( O(n/P) \) time when \( P \) processes are available.

**Case 4**

**Case 5**

Output(\( \alpha \)) is a function of \( \text{impos}(\alpha) \) and certain other fixed information. That is

\[ \text{output}(\alpha) = f(\text{impos}(\alpha), l) \]

Example: Input sequence \( \xi_1 \xi_2 \xi_3 \ldots \xi_n \) is a random ordering of the integers \( 1, 2, \ldots, n \).

For the output we want them in sequential order. Then \( \text{output}(\alpha) = f(\alpha) = 0 \)

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

**Case 6**

Output(\( \alpha \)) is a function of the input position of \( \alpha \) and the entire input string.

That is

\[ \text{output}(\alpha) = f(\text{impos}(\alpha), l) \]

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

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.

Consider a string of symbols of length \( n \) that consists of symbols belonging
different groups: \( G_1, G_2 \) and \( G_3 \). We want to write an algorithm that transforms
string into an output string in which all symbols from \( G_1 \) are placed at the
t of \( G_3 \) at the end and all from \( G_2 \) in the middle of the string. The order
is within each group for output should be the same as their order in the input.

\( \text{input}(a) \) may be given by:

1. (number of symbols that belong to group \( G_1 \) and precede the ith symbol
in the input string) + 1
2. (number of symbols that belong to group \( G_1 \)) + (number of symbols that
belong to group \( G_2 \) and precede the ith symbol in the input string) + 1
3. (number of symbols that belong to group \( G_1 \)) + (number of symbols that
belong to group \( G_2 \)) + (number of symbols that belong to \( G_3 \) 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
be input or by each processor adding 1 to an accumulator from a set of accumulators
with one accumulator to represent each of the groups of symbols. Once this
is obtained, the position of symbols in the output string can be determined in
\( n \) processors are available.

Transformation of infix notation into postfix notation. A solution to this
problem given by Dekel and Sahal [2]. In [2], the Shared Memory Model [3] is used as
of computer. Using this model of computation it is shown that the proper
symbols in the output string can be computed in \( O(n \log^2 n) \) using \( n \) processors.

x conditions \( G_1 \) and \( G_2 \) 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
the functions to be repeated in the output. (Their output may be regarded as 0.)

\( \text{output}(a) \) is a function of \( n_i \) of the input position of \( a \), and of some additional local
on concerning \( a \)'s neighbors and their input positions. That is

\( \text{output}(a) = \sum_{k=1}^{\infty} \sum_{k=1}^{\infty} \text{input}(a_{i+k}) \) for \( a = a_k 
\text{and constants } b_1 \) and \( b_2 \).

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
places, then the next five etc. Considerations for local sorting complexity are
those for general sorting as described in Case 5. In fact, one might regard
as a general sorting case where the first five input symbols are \( i_k \), the next five
\( c_k \).

\( \text{output}(a) \) is a function of \( \text{input}(a) \) and of \( \text{output}(b) \) for all \( a \) such that \( \text{input}(a) \neq \text{output}(b) \)
That is

\( \text{output}(a) = \sum_{k=1}^{\infty} \text{input}(a_k), \text{output}(b_k) \) with \( \text{input}(a) \neq \text{input}(b) \).

The output is a random reordering of the input. We cannot assign output index

REFERENCES

[1] D.E. Muller and F.P. Preparata, "Bounds to Complexities of Networks for Sorting and


1984