RESEARCH MONOGRAPHS IN PARALLEL AND DISTRIBUTED COMPUTING

Péter Kacsuk
Multilogic Computing

Execution Models of Prolog for Parallel Computers

RESEARCH MONOGRAPHS IN PARALLEL AND DISTRIBUTED COMPUTING

Péter Kacsuk
Multilogic Computing

Execution Models of Prolog for Parallel Computers

Pitman, London

The MIT Press, Cambridge, Massachusetts

PITMAN PUBLISHING
128 Long Acre, London WC2E 9AN

First published 1990

Available in the Western Hemisphere and Israel from
The MIT Press
Cambridge, Massachusetts (and London, England)

ISSN 0953-7767

British Library Cataloguing in Publication Data
Kacsuk, Péter
Execution models of Prolog for parallel computers.
(Research monographs in parallel and distributed
computing, ISSN 0953-7767)
1. Computer systems. Parallel programming
I. Title II. Series
004'.35

ISBN 0-273-08806-8

Library of Congress Cataloging-in-Publication Data
Kacsuk, Péter.
Execution models of Prolog for parallel computers / Péter Kacsuk.
p. cm.—(Research monographs in parallel and distributed
computing)
Includes bibliographical references.
ISBN 0-262-11149-7
1. Prolog (Computer program language) 2. Parallel processing
(Electronic computers) I. Title. II. Series.
QA76.73.P76K33 1990
005.13'3—dc20

Reproduced and printed by photolithography
in Great Britain by Biddles Ltd, Guildford

FOREWORD

I am pleased to be able to write a preface to this monograph, as it represents two milestones in this series. On the one hand it is the first manuscript we have published from an Eastern European source, and with the opening up that has occurred through the recent initiatives in *perestroika*, I hope that it will be the first of many. As this monograph demonstrates and as I have discovered myself through recent travels to Eastern Europe, research in parallel processing there is very strong since it is quite rightly considered a growth area in computing technology.

The volume itself considers the area of parallel implementation of logic programming languages through Prolog, another first in this series, with some previous and forthcoming volumes showing a western preference for a functional style of declarative parallel programming.

The text itself is easily read and introduces parallel implementations of Prolog including some novel schemes developed by Kacsuk himself. These include an SIMD style implementation based on sets of solutions, and, more importantly, at least so far as I am concerned, the ECDAM model of Prolog interpretation, which is truly distributed. The latter was proposed as a means of distributing a Prolog search space over a set of homogeneous processors and was implemented in occam 1.

More recently we have extended this model at Southampton University in occam 2, so that it now handles *recursion* and *cut* in a fully distributed manner. We have also made proposals for optimising communication through the use of structure stores. However, our primary interest in this model is in implementing Prolog over the VSA virtual machine architecture, which provides code generation for SIMD and MIMD architectures using data parallelism. The ECDAM model is well suited to this implementation paradigm as the search tree and structure store can both be considered as distributed or parallel data-structures, and load balancing—a critical requirement for this model—comes for free with a good implementation of the VSA.

For those interested, I will be happy to disseminate our further developments to the ECDAM model, and finally, something to watch out for . . . details of the VSA standard definition will soon be published as a monograph in this series.

Chris Jesshope
Southampton University

ACKNOWLEDGEMENT

The research presented in this book was done in the Theoretical Laboratory of the Computer Research Institute and Innovation Center (SZKI) led by Bálint Dömölki and in the DAP Support Unit (DAPSU) of the Queen Mary College of the University of London led by Dennis Parkinson.

I owe a special debt to Bálint Dömölki who has supported my research and made it possible for me to make contact with researchers outside of Hungary.

I am gratetful to Dennis Parkinson who invited me to Queen Mary College and ensured all the conditions needed for fruitful research work.

Both institutes were very generous with their resources, human and otherwise. The members of my department, Péter Garami, Tamás Dénes, András Domán, István Marosi and many members of the Prolog staff of SZKI, Péter Szeredi, Péter Köves, Zsuzsa Farkas, Edit Sántáné-Tóth, Iván Futó, all gave me a great deal of help. Péter Garami provided constructive critisism, which often forced me to search for better solutions.

The whole staff of DAPSU helped me to adapt myself to a perfectly new environment. Special thanks are due to Andrew Bale who assisted me in the implementation of DAP Prolog on the DAP.

I am grateful to Maarten van Emden and Mantis Cheng who invited me to the University of Waterloo so that I could implement my ideas on their iPSC computer. I would also like to thank AL Sary and Lindsay Errington who cooperated with me in the implementation of the Extended Cellular Dataflow model on the iPSC, and Niel Ostlund who made it possible for us to use the iPSC.

I also benefited by being in continuous correspondence with Andrzej Ciepielewski of the Royal Institute of Technology, Stockholm and Michael Ratcliffe of European Computer-Industry Research Centre, Munich.

My deepest thanks are due to my wife Lívia and children Zsófia, Zoltán and Dániel for having put up with the occasional disadvantage of my research.

GLOSSARY

AI	Artificial Intelligence
DAP	Distributed Array Processor
DST	Dataflow Search Tree
ECDAM	Extended Cellular Dataflow Model
ETF	Extended Transition Function
FIN	Fast Interconnection Network
GDM	Generalized Dataflow Model
GGF	Guarded Goal Form
HPS	Homogeneous Processor Spaces
ILN	Intelligent Logic Network
LRDF	Left-Right selection strategy, Depth First search strategy
MCU	Master Control Unit
MIMD	Multiple Instruction Multiple Data computer
PE	Processing Element
PPAM	Parallel Prolog Abstract Machine
RDBS	Relational Data Base Systems
SIMD	Single Instruction Multiple Data computer
SPP	Sequential Processor Pool

CONTENTS

INTRODUCTION

In the field of artificial intelligence and particularly in programming expert systems, Prolog and other logic programming languages have become widely accepted and popular tools. On the other hand some weaknesses of Prolog became evident when it was used for solving large practical problems:

- o Due to the recursive programming style of Prolog the size of memory required to solve a problem rapidly increases with the search space.
- o The resolution mechanism of Prolog requires a long search time in case of large data bases which results in an unacceptable response time.
- o The efficiency of solving numerical subproblems within the framework of logic programs is extremely low.

Recent advances in micro-electronics, particularly in the area of VLSI fabrication, has solved the first problem by offering large size memories at a reasonable price and also made experimentation with massively parallel computers a reality. Parallelism is seen in Artificial Intelligence as an absolute necessity, in order to solve the second problem, and consequently a lot of researchers are now focusing on the development of these new architectures and the development of radically new computational paradigms to utilize the parallelism inherent in the problems being solved and available in the architecture level.

The main motivation of the research described in this book is derived from the "semantic gap" between the logic programming languages and the architecture of the parallel computers. On the one hand there is a widely accepted, popular programming language for solving Artificial Intelligence problems and on the other hand there are available parallel computers. The main question raised by the semantic gap is how to implement logic programming languages on parallel computers in an effective way capable of exploiting the inherent parallelism of logic programs and utilising the parallel architecture offered by the parallel computers.

One of the main advantages of logic programming languages can be described by an equation introduced by Kowalski [Kowa79]:

algorithm = logic + control

expressing that in case of the logic programming languages the programmer should not bother about the control of the program: it is sufficient to describe the logic of the problem to be solved. Standard, sequential implementation techniques of Prolog rely on the so-called LRDF (Left-to-Right Depth-First) control. The objective of research for parallel implementation of logic programs is to discover control strategies different from LRDF which allow the parallel execution of logic programs.

The majority of proposals have been aimed at implementing logic programs on shared memory multiprocessors where the number of processors is limited by the access mechanism of the shared memory. Another large group of researchers has been dealing with the question of how to implement logic programs on computer networks. A relatively small number of proposals have considered massively parallel computers as the target architecture for logic programs. There have been no attempts at all to exploit SIMD architectures for this purpose, though SIMD machines can help to solve the third problem of logic programs, namely the numerical inefficiency.

In this book two research projects are described. The first was intended to explore the possibilities of implementing logic programs on MIMD, non-shared memory type, massively parallel computers containing 100-1000 processing elements, which are identical and connected in a regular, neighbourhood oriented communication network. For brevity this aggregate of processors will be called Homogeneous Processor Space (HPS) in the book. The second project investigates the possibility of implementing Prolog on a typical SIMD machine, called Distributed Array Processor (DAP).

Considering the research for parallel implementation of logic programs three levels of investigation can be distinguished [SyWe85]:

- o The execution level which involves the underlying parallel computer architecture.
- o The model level which describes how the parallel processes are created and how their communication and synchronization is organized.
- o The language level which decides whether explicit or implicit parallelism is applied.

The two research projects described in the book explore these levels in the following way:

a) Parallel Prolog Abstract Machine (PPAM)

execution level:	Transputer-like arrays
model level:	Extended Cellular-Dataflow Model (ECDAM)
language level:	Pure logic programs

b) DAP Prolog, a parallel variant of Prolog on the DAP

execution level:	AMT´s Distributed Array Processor (DAP)
model level:	Set- and array-oriented execution models
language level:	Parallel extension of Prolog called DAP Prolog

The objectives of the research described in this book are as follows:

- o To define a parallel computational paradigm (Extended Cellular-Dataflow Model) to overcome the semantic gap between the logic programming languages and the architecture of massively parallel computers.
- o Based on the parallel computational paradigm to create a Parallel Prolog Abstract Machine (PPAM) as a general starting point for the implementation of logic programming languages on parallel computers.
- o To exploit the different types of parallelism (Search, OR and AND parallelism) of logic programs on parallel computers by means of the Extended Cellular-Dataflow Model.
- o To define a parallel logic machine, which is efficient for executing in parallel logic programs based on the Extended Cellular-Dataflow Model.
- o To explore the possibilities of implementing logic programming languages on array processors, like the DAP. To invent parallel implementation techniques for the effective execution of Prolog on the DAP.
- o To define a parallel extension of Prolog which is able to utilize the processor aggregate of the DAP for effective solution of numerical subproblems within logic programs.

The book is organized in three main parts. The first part is an overview of the results of other research projects. It encompasses Chapters 1, 2 and 3. Its purpose is to give an overall presentation of the state of the art in parallel processing and in its application for implementing logic programming languages on parallel computers. Chapter 1 gives an overview of parallel computers, languages and computational paradigms. Chapter 2 introduces the basic notions of logic programming and shows the basic implementation techniques of Prolog for sequential computers. Chapter 3 is a description of the parallel implementation techniques proposed so far for implementing logic programs on parallel computers.

The second part consists of Chapter 4, 5 and 6 and describes the results of the first research project. Chapter 4 describes the Extended Cellular-Dataflow Model for parallel interpretation of logic programs and the Parallel Prolog Abstract Machine for implementing logic programs on parallel computers. Restricted OR- and AND-parallel interpretations of Prolog are shown in Chapter 4 based on ECDAM. Chapter 5 introduces some improvements into ECDAM for achieving higher level of parallelism. Chapter 6 describes some mapping techniques of PPAM for transputer-like arrays and proposes an HPS architecture, called LOGFLOW, for parallel implementation of Prolog programs. Four different implementations of ECDAM are summarized in Chapter 6 and three of them are described in detail in the Appendices. Appendix 1 describe the T-Prolog implementation of ECDAM for Prolog programmers who are interested in making experiments with the model. Appendix 2 contains the Occam source code of a restricted version of ECDAM for those interested in developing Prolog on multi-transputers without shared memory. Finally Appendix 3 demonstrates how the same model can be implemented on an SIMD machine giving the DAP FORTRAN code of the PPAM interpreter.

The third part consists of Chapter 7 and 8, and presents the results of the second research project. Chapter 7 defines DAP Prolog as a parallel extension of Prolog for the Distributed Array Processor and describes the set- and array-oriented execution models of DAP Prolog. Chapter 8 presents the implementation techniques used for DAP Prolog on the DAP. Appendix 4 contains a simple DAP Prolog program demonstrating the array-oriented execution mode of DAP Prolog.

This book is intended for advanced workers in parallel logic programming and for those investigating parallel programming paradigms and symbolic programming on novel computer architectures. It can also be recommended to those who are getting started in parallel logic programming who have either a logic programming or parallel computer background.

How to read the book? The reader is assumed to have a basic general knowledge of Prolog though no particular Prolog programming experience is required. Advanced workers in parallel logic programming can skip the first three chapters which are written for beginners in the field. The second and third part of the book are independent of each other so readers interested in only MIMD or SIMD machines can read only the relevant chapters. The book can be read without the appendices. However the appendices will assist those who found a particular part of the book interesting and wish to gain a deeper knowledge about it.

1 PARALLEL PROCESSING

There are many application areas of computers such as artificial intelligence, neural network simulation, meteorology, and image processing, that require a large amount of processing power in order to obtain a usable result for the problem being solved. Though the speed of the conventional computers keeps increasing their architecture limits them by a basically serial approach to computation based on the von Neumann organization. These von Neumann principles include:

- o A single computing element incorporating a processor, communications and memory.
- o Memory organized as a linear chain of fixed-size memory cells
- o Data and instructions not distinguished in memory.
- o Application of sequential, centralized control of computation

Advances in the design and fabrication of VLSI circuits has enabled one to build computers consisting of hundreds or thousands of processing elements [Fahl83], [StMi84], [Hill85]. The main novelty of these massively parallel computers is that the processing elements (PEs) are able to work cooperatively on the solution to a single problem. However this feature highlights a most difficult problem - how to organize the computation so that the large number of processing elements can effectively be utilized during computation. To help solve this problem many different computational paradigms and novel computer architectures have been proposed or built which are radically different from the von Neumann organization.

In this chapter a short overview of parallel computers is given, based on the three main aspects of parallel processing :

- o Parallel computational paradigms
- o Parallel computer architectures
- o Parallel programming languages

1.1 Classification of Parallel Computers

The most generally accepted classification of parallel computers was given by Flynn [Flyn72], who introduced the distinction between parallel computers based on the