Research Notes in Artificial Intelligence

ANOOP GUPTA

Parallelism in Production systems

Pitman, London

Morgan Kaufmann Publishers, Inc., Los Altos, California

Anoop Gupta
Department of Computer Science
Carnegie-Mellon University

Parallelism in Production Systems

Pitman, London

Morgan Kaufmann Publishers, Inc., Los Altos, California

PITMAN PUBLISHING
128 Long Acre, London WC2E 9AN

First published 1987

Available in the Western Hemisphere from
MORGAN KAUFMANN PUBLISHERS, INC.,
95 First Street, Los Altos, California 94022

ISSN 0268-7526

British Library Cataloguing in Publication Data
Gupta, Anoop
Parallelism in production systems.—
(Research notes in artificial intelligence,
ISSN 0268-7526).
1. Artificial intelligence
I. Title II. Series
006.3 Q335

ISBN 0-273-08782-7

Library of Congress Cataloging in Publication Data
Gupta, Anoop.
Parallelism in production systems.

(Research notes in artificial intelligence;
ISSN 0268-7526)
Originally presented as the author's thesis
(doctoral-Carnegie-Mellon University, 1986)
Bibliography: p.
Includes index.
1. Parallel processing (Electronic computers)
2. Artificial intelligence. 3. Expert systems
(Computer science) I. Title. II. Series: Research
notes in artificial intelligence (London, England)
QA76.5.G857 1987 004'.35 87-20428
ISBN 0-934613-55-9 (U.S.)

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

Acknowledgements

I would like to thank my advisors Charles Forgy, Allen Newell, and HT Kung for their guidance, support, and encouragement. Charles Forgy helped with his deep understanding of production systems and their implementation. Many of the ideas presented in this thesis originated with him or have benefited from his comments. Allen Newell, in addition to being an invaluable source of ideas, has shown me what doing research is about, and through his own example, what it means to be a good researcher. He has been a constant source of inspiration and it has been a great pleasure to work with him. HT Kung has been an excellent sounding board for ideas. He greatly helped in keeping the thesis on solid ground by always questioning my assumptions. I would also like to thank Al Davis for serving on my thesis committee. The final quality of the thesis has benefited significantly from his comments.

The work reported in this thesis has been done as a part of the Production System Machine (PSM) project at Carnegie-Mellon University. I would like to thank its current and past members — Charles Forgy, Ken Hughes, Dirk Kalp, Ted Lehr, Allen Newell, Kemal Oflazer, Jim Quinlan, Milind Tambe, Leon Weaver, and Robert Wedig — for their contributions to the research. I would also like to thank Greg Hood, John Laird, Bob Sproull (my advisor for the first two years at CMU), and Hank Walker for many interesting discussions about my research.

I would like to thank all my friends in Pittsburgh who have made these past years so enjoyable. I would like to thank Greg Hood, Ravi Kannan, Gudrun and Georg Klinker, Roberto Minio, Bud Mishra, Pradeep Sindhu, Pedro Szekely, Hank Walker, Angelika Zobel, and especially Paola Giannini and Yumi Iwasaki for making life so much fun. Finally, I would like to thank my family, my parents and my two sisters, for their immeasurable love, encouragement, and support of my educational endeavors.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4864, monitored by the Space and Naval Warfare Systems Command under Contract N00039-85-0134. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

Abstract

Production systems (or rule-based systems) are widely used in Artificial Intelligence for modeling intelligent behavior and building expert systems. Most production system programs, however, are extremely computation intensive and run quite slowly. The slow speed of execution has prohibited the use of production systems in domains requiring high performance and real-time response. This thesis explores the role of parallelism in the high-speed execution of production systems.

On the surface, production system programs appear to be capable of using large amounts of parallelism — it is possible to perform match for each production in a program in parallel. The thesis shows that in practice, however, the speed-up obtainable from parallelism is quite limited, around 10-fold as compared to initial expectations of 100-fold to 1000-fold. The main reasons for the limited speed-up are: (1) there are only a small number of productions that are *affected* (require significant processing) per change to working memory; (2) there is a large variation in the processing requirement of these productions; and (3) the number of changes made to working memory per recognize-act cycle is very small. Since the number of productions affected and the number of working-memory changes per recognize-act cycle are not controlled by the implementor of the production system interpreter (they are governed mainly by the author of the program and the nature of the task), the solution to the problem of limited speed-up is to somehow decrease the variation in the processing cost of affected productions. The thesis proposes a parallel version of the Rete algorithm which exploits parallelism at a very fine grain to reduce the variation. It further suggests that to exploit the fine-grained parallelism, a shared-memory multiprocessor with 32-64 high performance processors is desirable. For scheduling the fine-grained tasks consisting of about 50-100 instructions, a hardware task scheduler is proposed.

The thesis presents simulation results for a large set of production systems exploiting different sources of parallelism. The thesis points out the features of existing programs that limit the speed-up obtainable from parallelism and suggests solutions for some of the bottlenecks. The simulation results show that using the suggested multiprocessor architecture (with individual processors performing at 2 MIPS), it is possible to obtain execution speeds of about 12000 working-memory element changes per second. This corresponds to a speed-up of 10-fold over the best known sequential implementation using a 2 MIPS processor. This performance is significantly higher than that obtained by other proposed parallel implementations of production systems.

Contents

1 Introduction 1
- 1.1 Preview of Results 3
- 1.2 Organization of the Thesis 4

2 Background 7
- 2.1 OPS5 7
  - 2.1.1 Working-Memory Elements 8
  - 2.1.2 The Left-Hand Side of a Production 8
  - 2.1.3 The Right-Hand Side of a Production 9
- 2.2 Soar 10
- 2.3 The Rete Match Algorithm 10
- 2.4 Why Parallelize Rete? 15
  - 2.4.1 State-Saving vs. Non-State-Saving Match Algorithms 15
  - 2.4.2 Rete as a Specific Instance of State-Saving Algorithms 17
  - 2.4.3 Node Sharing in the Rete Algorithm 19
  - 2.4.4 Rete as a Parallel Algorithm 19

3 Measurements on Production Systems 21
- 3.1 Production-System Programs Studied in the Thesis 21
- 3.2 Surface Characteristics of Production Systems 22
  - 3.2.1 Condition Elements per Production 23
  - 3.2.2 Actions per Production 24
  - 3.2.3 Negative Condition Elements per Production 24
  - 3.2.4 Attributes per Condition Element 25
  - 3.2.5 Tests per Two-Input Node 25
  - 3.2.6 Variables Bound and Referenced 27
  - 3.2.7 Variables Bound but not Referenced 27
  - 3.2.8 Variable Occurrences in Left-Hand Side 28
  - 3.2.9 Variables per Condition Element 28
  - 3.2.10 Condition Element Classes 30
  - 3.2.11 Action Types 32
  - 3.2.12 Summary of Surface Measurements 32
- 3.3 Measurements on the Rete Network 32
  - 3.3.1 Number of Nodes in the Rete Network 33
  - 3.3.2 Network Sharing 34
- 3.4 Run-Time Characteristics of Production Systems 34
  - 3.4.1 Constant-Test Nodes 35
  - 3.4.2 Alpha-Memory Nodes 36

3.4.3 Beta-Memory Nodes 37
3.4.4 And Nodes 37
3.4.5 Not Nodes 39
3.4.6 Terminal Nodes 39
3.4.7 Summary of Run-Time Characteristics 39
4 Parallelism in Production Systems 43
4.1 The Structure of a Parallel Production-System Interpreter 43
4.2 Parallelism in Match 44
4.2.1 Production Parallelism 45
4.2.2 Node Parallelism 48
4.2.3 Intra-Node Parallelism 50
4.2.4 Action Parallelism 51
4.3 Parallelism in Conflict-Resolution 53
4.4 Parallelism in RHS Evaluation 53
4.5 Application Parallelism 54
4.6 Summary 54
4.7 Discussion 55
5 Parallel Implementation of Production Systems 59
5.1 Architecture of the Production-System Machine 59
5.2 The State-Update Phase Processing 62
5.2.1 Hash-Table Based vs. List Based Memory Nodes 62
5.2.2 Memory Nodes Need to be Lumped with Two-Input Nodes 64
5.2.3 Problems with Processing Conjugate Pairs of Tokens 66
5.2.4 Concurrently Processable Activations of Two-Input Nodes 67
5.2.5 Locks for Memory Nodes 69
5.2.6 Linear vs. Binary Rete Networks 69
5.3 The Selection Phase Processing 75
5.3.1 Sharing of Constant-Test Nodes 75
5.3.2 Constant-Test Node Successors 75
5.3.3 Alpha-Memory Node Successors 77
5.3.4 Processing Multiple Changes to Working Memory in Parallel 77
5.4 Summary 77
6 The Problem of Scheduling Node Activations 81
6.1 The Hardware Task Scheduler 82
6.1.1 How Fast Need the Scheduler be? 82
6.1.2 The Interface to the Hardware Task Scheduler 83
6.1.3 Structure of the Hardware Task Scheduler 85
6.1.4 Multiple Hardware Task Schedulers 86
6.2 Software Task Schedulers 88
7 The Simulator 93
7.1 Structure of the Simulator 94
7.1.1 Inputs to the Simulator 94
7.1.1.1 The Input Trace 94
7.1.1.2 The Computational Model 94
7.1.1.3 The Cost Model 95

7.1.1.4 The Memory Contention Model 97
7.1.2 Outputs of the Simulator 100
7.2 Limitations of the Simulation Model 101
7.3 Validity of the Simulator 102
8 Simulation Results and Analysis 105
8.1 Traces Used in the Simulations 105
8.2 Simulation Results for Uniprocessors 107
8.3 Production Parallelism 110
8.3.1 Effects of Action Parallelism on Production Parallelism 115
8.4 Node Parallelism 117
8.4.1 Effects of Action Parallelism on Node Parallelism 117
8.5 Intra-Node Parallelism 121
8.5.1 Effects of Action Parallelism on Intra-Node Parallelism 121
8.6 Linear vs. Binary Rete Networks 126
8.6.1 Uniprocessor Implementations with Binary Networks 126
8.6.2 Results of Parallelism with Binary Networks 128
8.7 Hardware Task Scheduler vs. Software Task Queues 133
8.8 Effects of Memory Contention 140
8.9 Summary 145
9 Related Work 147
9.1 Implementing Production Systems on C.mmp 147
9.2 Implementing Production Systems on Illiac-IV 148
9.3 The DADO Machine and the TREAT Match Algorithm 148
9.4 The NON-VON Machine 152
9.5 Oflazer's Work on Partitioning and Parallel Processing of Production Systems 154
9.5.1 The Partitioning Problem 154
9.5.2 The Parallel Algorithm 155
9.5.3 The Parallel Architecture 156
9.5.4 Discussion 157
9.6 Honeywell's Data-Flow Model 159
9.7 Other Work on Speeding-up Production Systems 160
10 Summary and Conclusions 161
10.1 Primary Results of Thesis 161
10.1.1 Suitability of the Rete-Class of Algorithms 161
10.1.2 Parallelism in Production Systems 162
10.1.3 Software Implementation Issues 165
10.1.4 Hardware Architecture 165
10.2 Some General Conclusions 166
10.3 Directions for Future Research 167
References 171
Appendix A ISP of Processor Used in Parallel Implementation 179
Appendix B Code and Data Structures for Parallel Implementation 185
B.1 Code for Interpreter with Hardware Task Scheduler 185
B.2 Code for Interpreter Using Multiple Software Task Schedulers 204
Appendix C Derivation of Cost Models for the Simulator 211
C.1 Cost Model for the Parallel Implementation Using HTS 211
C.2 Cost Model for the Parallel Implementation Using STQs 223

List of Figures

Figure 2-1: A sample production. 7
Figure 2-2: The Rete network. 12
Figure 3-1: Condition elements per production. 23
Figure 3-2: Actions per production. 24
Figure 3-3: Negative condition elements per production. 25
Figure 3-4: Attributes per condition element. 26
Figure 3-5: Tests per two-input node. 26
Figure 3-6: Variables bound and referenced. 27
Figure 3-7: Variables bound but not referenced. 28
Figure 3-8: Occurrences of each variable. 29
Figure 3-9: Variables per condition element. 29
Figure 4-1: OPS5 interpreter cycle. 43
Figure 4-2: Soar interpreter cycle. 44
Figure 4-3: Selection and state-update phases in match. 45
Figure 4-4: Production parallelism. 46
Figure 4-5: Node parallelism. 49
Figure 4-6: The cross-product effect. 51
Figure 5-1: Architecture of the production-system machine. 60
Figure 5-2: A production and the associated Rete network. 63
Figure 5-3: Problems with memory-node sharing. 65
Figure 5-4: Concurrent activations of two-input nodes. 67
Figure 5-5: The long-chain effect. 72
Figure 5-6: A binary Rete network. 73
Figure 5-7: Scheduling activations of constant-test nodes. 76
Figure 5-8: Possible solution when too many alpha-memory successors. 78
Figure 6-1: Problem of dynamically changing set of processable node activations. 82
Figure 6-2: Effect of scheduler performance on maximum speed-up. 83
Figure 6-3: Structure of the hardware task scheduler. 85
Figure 6-4: Effect of multiple schedulers on speed-up. 89
Figure 6-5: Multiple software task queues. 90
Figure 7-1: A sample trace fragment. 95
Figure 7-2: Static node information. 96
Figure 7-3: Code for left activation of an and-node. 97
Figure 7-4: Degradation in performance due to memory contention. 100
Figure 8-1: Production parallelism (nominal speed-up). 112
Figure 8-2: Production parallelism (true speed-up). 112
Figure 8-3: Production parallelism (execution speed). 112

Figure 8-4: Production and action parallelism (nominal speed-up). 116
Figure 8-5: Production and action parallelism (true speed-up). 116
Figure 8-6: Production and action parallelism (execution speed). 116
Figure 8-7: Node parallelism (nominal speed-up). 118
Figure 8-8: Node parallelism (true speed-up). 118
Figure 8-9: Node parallelism (execution speed). 118
Figure 8-10: Node and action parallelism (nominal speed-up). 120
Figure 8-11: Node and action parallelism (true speed-up). 120
Figure 8-12: Node and action parallelism (execution speed). 120
Figure 8-13: Intra-node parallelism (nominal speed-up). 122
Figure 8-14: Intra-node parallelism (true speed-up). 122
Figure 8-15: Intra-node parallelism (execution speed). 122
Figure 8-16: Intra-node and action parallelism (nominal speed-up). 124
Figure 8-17: Intra-node and action parallelism (true speed-up). 124
Figure 8-18: Intra-node and action parallelism (execution speed). 124
Figure 8-19: Average nominal speed-up. 125
Figure 8-20: Average true speed-up. 125
Figure 8-21: Average execution speed. 125
Figure 8-22: Production parallelism (nominal speed-up). 129
Figure 8-23: Production parallelism (execution speed). 129
Figure 8-24: Production and action parallelism (nominal speed-up). 129
Figure 8-25: Production and action parallelism (execution speed). 129
Figure 8-26: Node parallelism (nominal speed-up). 130
Figure 8-27: Node parallelism (execution speed). 130
Figure 8-28: Node and action parallelism (nominal speed-up). 130
Figure 8-29: Node and action parallelism (execution speed). 130
Figure 8-30: Intra-node parallelism (nominal speed-up). 131
Figure 8-31: Intra-node parallelism (execution speed). 131
Figure 8-32: Intra-node and action parallelism (nominal speed-up). 131
Figure 8-33: Intra-node and action parallelism (execution speed). 131
Figure 8-34: Average nominal speed-up. 132
Figure 8-35: Average execution speed. 132
Figure 8-36: Effect of number of software task queues. 135
Figure 8-37: Production parallelism (nominal speed-up). 136
Figure 8-38: Production parallelism (execution speed). 136
Figure 8-39: Production and action parallelism (nominal speed-up). 136
Figure 8-40: Production and action parallelism (execution speed). 136
Figure 8-41: Node parallelism (nominal speed-up). 137
Figure 8-42: Node parallelism (execution speed). 137
Figure 8-43: Node and action parallelism (nominal speed-up). 137
Figure 8-44: Node and action parallelism (execution speed). 137
Figure 8-45: Intra-node parallelism (nominal speed-up). 138
Figure 8-46: Intra-node parallelism (execution speed). 138
Figure 8-47: Intra-node and action parallelism (nominal speed-up). 138
Figure 8-48: Intra-node and action parallelism (execution speed). 138
Figure 8-49: Average nominal speed-up. 139

Figure 8-50: Average execution speed. 139
Figure 8-51: Processor efficiency as a function of number of active processors. 141
Figure 8-52: Intra-node and action parallelism (nominal speed-up). 143
Figure 8-53: Intra-node and action parallelism (execution speed). 144
Figure 9-1: The prototype DADO architecture. 149
Figure 9-2: The NON-VON architecture. 153
Figure 9-3: Structure of the parallel processing system. 157

List of Tables

Table 3-1: VT: Condition Element Classes 30
Table 3-2: ILOG: Condition Element Classes 30
Table 3-3: MUD: Condition Element Classes 31
Table 3-4: DAA: Condition Element Classes 31
Table 3-5: R1-SOAR: Condition Element Classes 31
Table 3-6: EP-SOAR: Condition Element Classes 31
Table 3-7: Action Type Distribution 32
Table 3-8: Summary of Surface Measurements 32
Table 3-9: Number of Nodes 33
Table 3-10: Nodes per Production 33
Table 3-11: Nodes per Condition Element (with sharing) 33
Table 3-12: Nodes per Condition Element (without sharing) 34
Table 3-13: Network Sharing (Nodes without sharing/Nodes with sharing) 34
Table 3-14: Constant-Test Nodes 35
Table 3-15: Alpha-Memory Nodes 36
Table 3-16: Beta-Memory Nodes 37
Table 3-17: And Nodes 38
Table 3-18: Not Nodes 39
Table 3-19: Terminal Nodes 39
Table 3-20: Summary of Node Activations per Change 40
Table 3-21: Number of Affected Productions 40
Table 3-22: General Run-Time Data 41
Table 7-1: Relative Costs of Various Instruction Types 98
Table 8-1: Uniprocessor Execution With No Overheads: Part-A 107
Table 8-2: Uniprocessor Execution With No Overheads: Part-B 108
Table 8-3: Uniprocessor Execution With Overheads: Part-A Node Parallelism and Intra-Node Parallelism 110
Table 8-4: Uniprocessor Execution With Overheads: Part-B Node Parallelism and Intra-Node Parallelism 110
Table 8-5: Uniprocessor Execution With Overheads: Part-A Production Parallelism 111
Table 8-6: Uniprocessor Execution With Overheads: Part-B Production Parallelism 111
Table 8-7: Uniprocessor Execution With No Overheads: Part-A 126
Table 8-8: Uniprocessor Execution With No Overheads: Part-B 127
Table 8-9: Uniprocessor Execution With Overheads: Part-A Node Parallelism and Intra-Node Parallelism 127
Table 8-10: Uniprocessor Execution With Overheads: Part-B Node Parallelism and Intra-Node Parallelism 127

Table 8-11: Uniprocessor Execution With Overheads: Part-A Production Parallelism 127
Table 8-12: Uniprocessor Execution With Overheads: Part-B Production Parallelism 128
Table 8-13: Comparison of Linear and Binary Network Rete 128

1 Introduction

Production systems (or rule-based systems) occupy a prominent position within the field of Artificial Intelligence. They have been used extensively to understand the nature of intelligence — in cognitive modeling, in the study of problem-solving systems, and in the study of learning systems [2, 45, 46, 65, 76, 77, 95]. They have also been used extensively to develop large expert systems spanning a variety of applications in areas including computer-aided design, medicine, configuration tasks, oil exploration [11, 14, 40, 42, 43, 56, 57, 84]. Production-system programs, however, are computation intensive and run quite slowly. For example, OPS5 [10, 19] production-system programs using the Lisp-based or the Bliss-based interpreter execute at a speed of only 8-40 working-memory element changes per second (wme-changes/sec) on a VAX-11/780.[1] Although sufficient for many interesting applications (as demonstrated by the current popularity of expert systems), the slow speed of execution precludes the use of production systems in many domains requiring high performance and real-time response. For example, one study that considered implementing the Harpy algorithm as a production system [66] for real-time speech recognition required that the program be able to execute at a rate of about 200,000 wme-changes/sec. The slow speed of execution of current systems also impacts the research that is done with them, since researchers often avoid programming styles and systems that run too slowly. This thesis examines the issue of significantly speeding up the execution of production systems (several orders of magnitude over the 8-40 wme-changes/sec). A significant increase in the execution speed of production systems is expected to open up new application areas for production systems, and to be valuable to both the practitioners and the researchers in Artificial Intelligence.

There also exist deeper reasons for wanting to speed up the execution of production systems. The cognitive activity of an intelligent agent involves two types of search: (1) *knowledge search*, that is, search by the agent of its knowledge base to find information that is relevant to solving a given problem; and (2) *problem-space search*, that is, search within the problem space [65] for a goal state. Problem-space search manifests itself as a combinatorial AND/OR search [68]. Since problem-space search when not pruned by knowledge is combinatorially explosive,

[1] This corresponds to an execution speed of 3-16 production firings per second. On average, 2.5 changes are made to the working memory per production firing.

a highly intelligent agent, regardless of what it is doing, must engage in a certain amount of knowledge search after each step that it takes. This results in knowledge search being a part of the *inner loop* of the computation performed by an intelligent agent. Furthermore, as the intelligence of the agent increases (the size of the knowledge base increases), the resources needed to perform knowledge search also increase, and it becomes important to speed up knowledge search as much as possible.

As an example, consider the problem of determining the next move to make in a game of chess. Problem-space search corresponds to the different moves that the player tries out before making the actual move. However, the fact that he tries out only a small fraction of all possible moves requires that he use problem and situation-specific knowledge to constrain the search. Knowledge search corresponds to the computation involved in identifying this problem and situation-specific knowledge from the rest of the knowledge that the player may have.

Knowledge search forms an essential component of the execution of production systems. Each execution cycle of a production system involves a knowledge-search step (the *match* phase), where the knowledge represented in rules is matched against the global data memory. Since the ability to do efficient knowledge search is fundamental to the construction of intelligent agents, it follows that the ability to execute production systems with large rule sets at high speeds will greatly help in constructing intelligent programs. In short, the match-phase computation (knowledge search) done in production systems is not something specific to production systems, but such computation has to be done, in one form or another, in any intelligent system. Thus, speeding up such computation is an essential part of the construction of highly intelligent systems. Furthermore, since production systems offer a highly transparent model of knowledge search, the results obtained about speed-up from parallelism for production systems will also have implications for other models of intelligent computation involving knowledge search.

There are several different methods for speeding up the execution of production systems: (1) the use of faster technology; (2) the use of better algorithms; (3) the use of better architectures; and (4) the use of parallelism. This thesis focuses on the use of parallelism. It identifies the various sources of parallelism in production systems and discusses the feasibility of exploiting them. Several implementation issues and some architectural considerations are also discussed. The main reasons for considering parallelism are: (1) Given any technology base, it is always possible to use multiple processors to achieve higher execution speeds. Stated another way, as technology advances, the new technology can also be used in the construction of multiple processor systems. Furthermore, as the rate of improvement in technology slows (as it must) parallelism becomes even more important. (2) Although significant improvements in speed have been obtained in the past through better compilation techniques and better algorithms [17, 20, 21, 22], we appear to be at a point where too much more cannot be expected. Furthermore, any improvements in compilation technology and algorithms will probably also carry over to the parallel implementations. (3) On the surface, production systems appear to be capable of using large amounts of parallelism — it is possible to perform the match for each

production in parallel. This apparent mismatch between the inherently parallel production systems and the uniprocessor implementations, makes parallelism the obvious way to obtain significant speed up in the execution rates.

The thesis concentrates on the parallelism available in OPS5 [10] and Soar [47] production systems. OPS5 was chosen because it has become widely available and because several large, diverse, and real production-system programs have been written in it. These programs form an excellent base for measurements and analysis. Soar was chosen because it represents an interesting new approach in the use of production systems for problem solving and learning. Since only OPS5 and Soar programs are considered, the analysis of parallelism presented in this thesis is possibly biased by the characteristics of these languages. For this reason the results may not be safely generalized to production-system programs written in languages with substantially different characteristics, such as EMYCIN, EXPERT, and KAS [60, 96, 14].

Finally, the research reported in this thesis has been carried out in the context of the *Production System Machine* (PSM) project at Carnegie-Mellon University, which has been exploring all facets of the problem of improving the efficiency of production systems [22, 23, 30, 31, 32, 33]. This thesis extends, refines, and substantiates the preliminary work that appears in the earlier publications.

1.1. Preview of Results

The first thing that is observed on analyzing production systems is that the speed-up from parallelism is quite limited, about 10-fold as compared to initial expectations of 100-fold or 1000-fold. The main reasons for the limited parallelism are: (1) The number of productions that require significant processing (the number of *affected* productions) as a result of a change to working memory is quite small, less than 30. Thus, processing each of these productions in parallel cannot result in a speed-up of more than 30. (2) The variation in the processing requirements of the affected productions is large. This results in a situation where fewer and fewer processors are busy as the execution progresses, which reduces the average number of processors that are busy over the complete execution cycle. (3) The number of changes made to working memory per recognize-act cycle is very small (around 2-3 for most OPS5 systems). As a result, the speed-up obtained from processing multiple changes to working memory in parallel is quite small.

To obtain a large fraction of the limited speed-up that is available, the thesis proposes the exploitation of parallelism at a very fine grain. It also proposes that all working-memory changes made by a production firing be processed in parallel to increase the speed-up. The thesis argues that the Rete algorithm used for performing the match step in existing uniprocessor implementations is also suitable for parallel implementations. However, there are several changes that are necessary to the serial Rete algorithm to make it suitable for parallel implementation. The thesis discusses these changes and gives the reasons behind the design decisions.

The thesis argues that a highly suitable architecture to exploit the fine-grained parallelism in production systems is a shared-memory multiprocessor, with about 32-64 high performance processors. For scheduling the fine grained tasks (consisting of about 50-100 instructions), two solutions are proposed. The first solution consists of a hardware task scheduler. The hardware task scheduler is to be capable of scheduling a task in one bus cycle of the multiprocessor. The second solution consists of multiple software task queues. Preliminary simulation studies indicate that the hardware task scheduler is significantly superior to the software task queues.

The thesis presents a large set of simulation results for production systems exploiting different sources of parallelism. The thesis points out the features of existing programs that limit the speed-up obtainable from parallelism and suggests solutions for some of the bottlenecks. The simulation results show that using the suggested multiprocessor architecture (with individual processors performing at 2 MIPS), it is possible to obtain execution speeds of 5000-27000 wme-changes/sec. This corresponds to a speed-up of 4-fold to 23-fold over the best known sequential implementation using a 2 MIPS processor. This performance is significantly higher than that obtained by other proposed parallel implementations of production systems.

1.2. Organization of the Thesis

Chapter 2 contains the background information necessary for the thesis. Sections 2.1 and 2.2 introduce the OPS5 and Soar production-system formalisms and describe their computation cycles. Section 2.3 presents a detailed description of the Rete algorithm which is used to perform the match step for production systems. The Rete algorithm forms the starting point for much of the work described later in the thesis. Section 2.4 presents the reasons why it is interesting to parallelize the Rete algorithm.

Chapter 3 lists the set of production-system programs analyzed in this thesis and presents the results of static and run-time measurements made on these production-system programs. The static measurements include data on the surface characteristics of production systems (for example, the number of condition elements per production, the number of attribute-value pairs per condition element) and data on the structure of the Rete networks constructed for the programs. The run-time measurements include data on the number of node activations per change to working memory, the number of working-memory changes per production firing, etc. The run-time data can be used to get rough upper-bounds on the speed-up obtainable from parallelism.

Chapter 4 focuses on the sources of parallelism in production-system implementations. For each of the sources (production parallelism, node parallelism, intra-node parallelism, action parallelism, and application parallelism) it describes some of the implementation constraints, the amount of speed-up expected, and the overheads associated with exploiting that source. Most of the chapter is devoted to the parallelism in the match phase; the parallelism in the conflict-resolution phase and the rhs-evaluation phase is discussed only briefly.