Informal Proceedings of

PPAI-91

International Workshop on Parallel Processing for AI

in conjunction with IJCAI-91

edited by

Laveen N. Kanal and Christian B. Suttner

24. - 25. August 1991
Sydney, Australia

P P A I – 91 Program Schedule

Saturday, August 24, 1991

8:30 - 9:00 **Registration & Coffee**

9:00 - 9:30 **Introduction and Overview** (L. Kanal)

9:30 - 11:30 **Session I – Search** (Chair: V. Kumar)
Analyzing Scalability of Parallel Algorithms and Architectures
V. Kumar/A. Gupta
A Parallel Genetic Algorithm for Solving the School Timetabling Problem
D. Abramson/J. Abela
IDA* on the Connection Machine
C. Powley/R.E. Korf
Superlinear Speedup Using Bidirectionalism and Islands
P.C. Nelson/A.A. Toptsis

11:30 - 11:45 **Break & Refreshments**

11:45 - 12:45 **Session II – Constraint Satisfaction** (Chair: A.K. Mackworth)
Parallel and Distributed Constraint Satisfaction
Y. Zhang/A.K. Mackworth
Parallel Path Consistency
S.Y. Susswein/T.C. Henderson/J.L. Zachary/C. Hansen/P. Hinker/G.C. Mar

12:45 - 14:00 **Lunch Break**

14:00 - 16:15 **Session III – Automated Theorem Proving** (Chair: W. Bibel)
Massive Parallelism in Inference Systems
F. Kurfess
Constructing Proofs using Connectionist Networks
G. Pinkas
Massively Parallel Processing and Automated Theorem Proving
J. Riche/R. Whaley/J. Barlow
A Parallel Model-Generation Theorem Prover with Ramified Term-Indexing
R. Hasegawa/H. Fujita/M. Fujita

16:15 - 16:30 **Break**

16:30 - 17:30 **Session IV – Computer Vision** (Chair: J. Aggarwal)
An Experimental Parallel Implementation of a Rule-Based Image Interpretation System
C.-C. Chu/J.C. Aggarwal
Parallel Processing of Sparse Data Structures for Computer Vision
S. Ranka

Sunday, August 25, 1991

8:30 - 9:00 Registration & Coffee

9:00 - 11:00 Session V – Logical Inference (Chair: C.B. Suttner)

Competition versus Cooperation
C.B. Suttner
ROO – a Parallel Theorem Prover
E.L. Lusk/W.W. McCune/J.K. Slaney
Random Competition: A Simple, but Efficient Method for Parallelizing Inference Systems
W. Ertel
Introduction to PMS-Prolog: A Distributed Coarse Grain Prolog with Processes, Modules and Streams
M.J. Wise

11:00 - 11:15 Break

11:15 - 13:00 Session VI – Natural Language Processing (Chair: H. Kitano)

Massively Parallel Artificial Intelligence and its Application to Natural Language Processing
H. Kitano
Graph-based Constraint Propagation for Massively-Parallel Natural Language Processing
H. Tomabechi
The MP-1 Benchmark Set for Parallel AI Architectures
R.F. DeMara/H. Kitano
A Marker Passing Parallel Processor for AI
D. Moldovan/W. Lee/C. Lin

13:00 - 14:15 Lunch Break

14:15 - 15:45 Session VII – Reasoning (Chair: J. Hendler)

Massively-Parallel Marker-Passing in Semantic Networks
J. Hendler
Constraint Satisfaction in Time-Constrained Memory
J.-P. Corriveau
Upward-Inductive Inheritance and Constant Time Downward Inheritance in Massively Parallel Knowledge Representation
J. Geller

15:45 - 16:00 Break

16:00 - 17:00 Session VIII – Misc. Topics and Discussion (Chair: L. Kanal)

Using Linda to Build Parallel AI Applications
M. Factor/S. Fertig/D.H. Gelernter
Key Issues in PPAI – Open Discussion

Preface

Most algorithms for solving AI problems are computationally very demanding. The current trend in computer architecture towards parallel machines offers the chance to exploit much more computational power than ever before and to design algorithms which would not have been feasible on sequential machines.

The intent of the organizers has been to bring together hardware architects, AI researchers and application engineers who are engaged in or interested in parallel processing for artificial intelligence.

Through sessions dedicated to particular topics we have sought to focus on the current interests of the community. Each session is designed to consist of an introductory statement or key presentation by the session chairman and a mix of selected submissions and invited talks.

There were fourteen invited presentations and 40 contributed submissions to the workshop. The limited time available mandated a stringent selection from among the submitted papers. One fourth of the submitted papers were selected for presentation at the workshop and the rule was adopted that if a selected paper's author could not be present in person at the workshop, that slot would be given to a paper accepted as an alternate. Approximately one third of the contributed papers were not accepted for inclusion in the proceedings although some of these submissions expressed interesting ideas and surely will find an audience at some other event. This proceedings includes the extended abstracts for most of the other papers. Missing are the ones that did not get to us by the final deadline. In a few such cases we have extracted a page with the title and abstract from the long paper submitted and included that page to represent the paper.

We are indebted to Wolfgang Bibel, Uwe Egly, Masayuki Fujita, James Hendler, Steffen Hölldobler, Mayuki Koshimura, Vipin Kumar, Ewing Lusk, Dan Moldovan, Gerd Neugebauer, Torsten Schaub, Georg Strobl, Christian Suttner, and Kazuo Taki for serving as reviewers under a short time constraint. We thank the invited authors for agreeing to participate in the workshop and thank all the authors who submitted contributions in response to the call for participation.

Organizers of PPAI-91,
L. Kanal, D. Moldovan, and C. Suttner

Upward-Inductive and Constant Time Downward Inheritance in Massively Parallel Knowledge Representation

James Geller*
Institute for Integrated Systems, CIS Department
and Center for Manufacturing Systems
New Jersey Institute of Technology, Newark, NJ 07102
geller@mars.njit.edu

ABSTRACT: We review an approach to Massively Parallel Knowledge Representation based on a limited inference strategy. A class tree is coded using a preorder numbering scheme and represented as a list. Algorithms for standard inheritance and upward-inductive inheritance are given for the preorder encoded class tree. Experimental data from a Connection Machine implementation are reported.

1. MASSIVELY PARALLEL KNOWLEDGE REPRESENTATION

The development of the Connection Machine and of Massively Parallel Computing in general was heavily influenced by problems from AI. Hillis (1985) used examples from Computer Vision and Knowledge Representation to motivate his design, e.g., Fahlman's (1979) famous NETL system. NETL was the first attempt in AI to create a Knowledge Representation model that could be translated naturally into hardware.

Knowledge Representation is the heart of many areas of Artificial Intelligence. Unfortunately, Knowledge Representation implementations are also notoriously slow. Shastri (1988, 1989) has stated that "To be deemed intelligent, a system must be capable of action within a specified time frame..." (1988, p.3). We concur that "A possible solution of the computational effectiveness problem lies in a synthesis of the limited inference approach and massive parallelism." (1988, p. 4).

Waltz (1990) has pointed out that AI has made surprisingly little use of massively parallel processing. However, there is a small number of researchers working on what we refer to as Massively Parallel Knowledge Representation (MaPKR, pronounced "mapcar"). Evett, Hendler, and Spector (1990) have been working on PARKA, an implementation of a Knowledge Representation system on the Connection Machine.

Geller (1990) and Geller and Du (1991) have presented an alternative approach that concentrates on parallelizing the "IS-A hierarchy." We first review the basics of this approach and then discuss two types of inheritance operations, standard (downward) inheritance and upward-inductive inheritance. It will be shown that in this implementation downward inheritance in a tree is a constant time operation *for a given machine size* even if the height of the tree is varied.

2. PREVIOUS WORK: SCHUBERT'S TREES

Many Knowledge Representation systems use a so called IS-A hierarchy as their backbone. In the simplest possible case this hierarchy is a tree. In Fig. 1 the node with the label Bird stands for the class of all Birds. Below the Bird node and connected to it are the Eagle node and the Crane node. Therefore, every eagle is a bird, and every crane is also a bird.

*This work was conducted using the computational resources of the Northeast Parallel Architectures Center (NPAC) at Syracuse University, which is funded by and operates under contract to DARPA and the Air Force Systems Command, Rome Air Development Center (RADC), Griffiss Air Force Base, NY, under contract# F306002-88-C-0031.

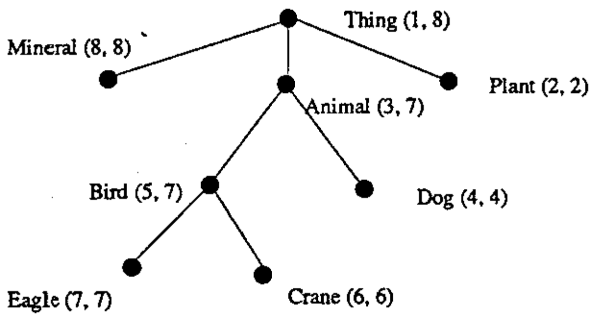

Figure 1: An IS-A Tree

One method of reasoning which is commonly associated with IS-A hierarchies is inheritance. It is assumed that every node in the hierarchy has certain associated attributes. If an attribute is associated with a node A and the user wants to know whether a node B has this attribute, then it is possible to answer this question in the affirmative, if A is above B and connected to it by a path of IS-A arcs.

Schubert *et al.* (1983) use a class reasoner as a "server" to a general purpose resolution theorem prover. This special purpose reasoner has to decide quickly whether A IS-A B and report the result back. To achieve this, Schubert has introduced a coding for the nodes in an IS-A tree of mutually exclusive classes. In Fig. 1 every node is followed by a pair of numbers. The first number is the result of a preorder right-to-left tree traversal. This traversal visits the nodes in Fig. 1 in the order (Thing, Plant, Animal, Dog, Bird, Crane, Eagle, Mineral). The second number of a node A is the largest first number that occurs under A. For example, under Animal (4, 5, 6, 7) occur as "first numbers". The largest of these numbers is 7, therefore the second number of Animal is 7. By definition, a leaf node's second number is identical to its first number.

The decision whether B is under A can be made easily by comparing the number pairs of A and B: the pair of B is a subinterval of the pair of A *iff* B is a subclass of A. Bird is a subclass of Animal, because (5, 7) is a subinterval of (3, 7).

3. THE LINEAR TREE REPRESENTATION

Schubert's representation is ideal for fast retrieval of subclass relations. However, any attempt to update the tree requires, that the number pairs of many of the nodes be recomputed. This difficulty can be overcome if one makes use of two observations. **(1)** The number pairs make the tree redundant. Instead of a tree, one can use a list of nodes with number pairs associated. **(2)** It is possible to update all the number pairs in parallel. This is the point where the Connection Machine comes into play.

We will now explain the intuition behind **(1)**. Proofs are contained in Geller and Du (1991). The basic idea for using a list representation instead of a tree representation is to transform a given tree into a list by a left-to-right preorder traversal. For Fig. 1 this results in (Thing (1, 8) Mineral (8, 8) Animal (3, 7) Bird (5, 7) Eagle (7, 7) Crane (6, 6) Dog (4, 4) Plant (2, 2)).

The verification of the subclass relation between any two nodes does not change. What is needed is an update operation for adding a new node to this list. This update operation should have the same effect as if we would have added that new node to the tree and would have recreated the tree by a left-to-right traversal. It is possible to find such an update operation if one always adds a new node at the leftmost possible position. Then we can add this new node to the list immediately after its parent, and we get the

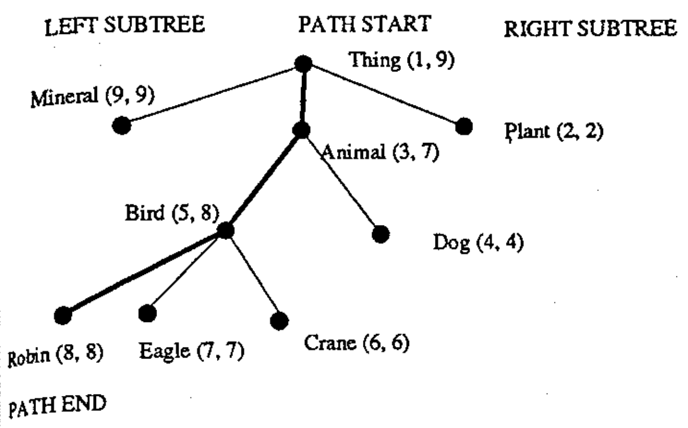

Figure 2: Three Parts of an IS-A Tree

same effect. For example, if Robin is added under Bird (to the left of Eagle) then a new left-to-right traversal results in (Thing (1, 8) Mineral (8, 8) Animal (3, 7) Bird (5, 7) Robin () Eagle (7, 7) Crane (6, 6) Dog (4, 4) Plant (2, 2)). This is identical to adding Robin into the list immediately after Bird.

The next step is to correctly update the number pairs of all the nodes. Fig. 2 shows that when one adds a node to a tree as a new leaf, this operation divides the tree into three parts. The first part is a path of IS-A arcs from the root node to the newly added node. This path cuts the tree into a left subtree and a right subtree. There are three simple rules for updating the number pairs of these three parts. **(1)** All the nodes in the right subtree have unchanged number pairs. **(2)** All the nodes in the path will have their first number unchanged but will have their second number incremented (by 1). **(3)** All the nodes in the left subtree will have both numbers incremented. (For a proof, see Geller and Du, 1991).

The update rules can be translated into rules that are valid for the list representation of a class hierarchy. The nodes in the right subtree will be exactly the nodes that occur in the list AFTER the newly added node. However, the left subtree and the path occur intermixed in the list BEFORE the newly added node. There is a way to decide whether a node belongs to the path or the left subtree. All nodes in the path will have a first number which is smaller than the first number of the parent node of the newly added node. This is not the case for all nodes in the left subtree.

In our example the nodes Plant, Dog, Crane, and Eagle do not require any changes to their number pairs. The nodes Thing, Animal and Bird belong to the path and therefore have their second numbers incremented. The node Mineral forms the left subtree and has both its numbers incremented. The resulting list representation is (Thing (1, 9) Mineral (9, 9) Animal (3, 8) Bird (5, 8) Robin () Eagle (7, 7) Crane (6, 6) Dog (4, 4) Plant (2, 2)). The new node Robin is numbered with (8, 8) (Geller and Du, 1991).

The algorithm described has been implemented on the Connection Machine (Thinking Machines Corporation, 1988). In semantic network implementations on massively parallel hardware every node in a concept network is assigned to one processor (Hillis, 1985). Attributes are usually represented as parallel variables and attribute values as values of these parallel variables. This is a localized representation of attributes which we share with Evett *et al.* (1990). Details about the implementation and about successful test runs are described in (Geller and Du, 1991).

4. INHERITANCE: TERMINOLOGY AND MOTIVATION

Downward inheritance is a well established technique. The contribution of this paper to the research in downward inheritance consists of experimental verification that downward inheritance can be done in a massively parallel environment in constant time for a given machine size[1] and a strict hierarchy. On the other hand, upward-inductive inheritance is a new technique which needs some justification.

Preliminary Notation: The letter X, possibly with subscripts, represents an individual or class node. Greek letters represent variables over nodes. The letters S and T represent sets of nodes. The letter A represents an attribute, and the letter a represents its corresponding value. When we say that an attribute A is *not defined at a node* X then there is no value associated with A.

Situation: Assume a set S of n nodes $S = \{X_1, X_2, \ldots, X_n\}$ which are all subclasses of X, or individuals belonging to X. Assume a set T of m nodes, such that $T \subset S$, (i.e., $m < n$) and an attribute A, that is defined for every $\Phi \in T$. Note that a_i of X_i and a_j of X_j are not necessarily the same, unless $i = j$. Finally, assume that A is undefined for X and cannot be derived by standard inheritance, and A is undefined for all $\Psi \in (S - T)$.

Problem: The problem is to find an algorithm that assigns a correct value a to the attribute A of X. This is an induction problem, and induction is generally considered an unsolved problem. We do not claim any research on solving the induction problem in general. However, it is claimed that a reasoner may find itself in a situation where no answer would be more disastrous than an incorrect one. If there is any probability for a given answer to be correct, then such an answer is certainly preferred to no answer at all.[2] Reasoning that relies on such an inductive step occurs quite often in everyday life. We consider this as a variant of what Shastri (1988) has described as evidential reasoning.

5. DOWNWARD INHERITANCE AND UPWARD-INDUCTIVE INHERITANCE

The downward inheritance algorithm relies on the fact that in our linear tree representation all the ancestors of a node are to its left. In addition, the number pairs of all ancestors are superintervals of the number pair of a given node. Therefore, using the Connection Machine, the algorithm looks as follows:

```
ACTIVATE-PROCESSORS-WITH
        PRENUM!! <=!! (!! PRENUM(N))  AND!! MAXNUM!! >=!! (!! MAXNUM(N))
        AND!! ATTRIBUTE!! <>!! (!! -1)
DO (RETRIEVE-PVAR ATTRIBUTE!! (*MAX SELF-ADDR!!))
```

We first activate all the processors that have a number pair which includes the given number pair. These are the possible ancestors. Of these we want only the nodes that have a value for a given attribute. We are marking the omission of a value by the number -1. Of these we select the right most node, which is lowest in the hierarchy. This is achieved by finding the node with the highest address. N is the node for which we want to know a value for ATTRIBUTE!!. *MAX finds, in parallel, the largest value of a given parallel variable.

The algorithm for upward-inductive inheritance is more complicated. All the nodes under N are activated. Of these we need to select all the nodes that have a value for the given attribute. Then the algorithm selects the left-most active node. It retrieves its property value and then deactivates temporarily all the nodes that do NOT have the same value. It counts the number of nodes that have the given value and then permanently

[1] The dependency on the machine size was pointed out to me by Lokendra Shastri and his students.

[2] This situation occurs for instance at written midterm examinations.

deactivates all those nodes. It repeats the same step, if necessary, several times. In the end it reports all found attribute values together with the number of occurrences. It is left to the user or a general purpose reasoner to decide whether he wants to use the most commonly occurring value.

Clearly, this algorithm has a serial component, and the worst case retrieval time depends on the number of different existing attribute values. This might result in very long response times for some attributes; however, for many practically useful attributes, such as color, there is a limited number of possible attribute values. Our implementation is recursive. The following is an iterative version of the algorithm.

```
ACTIVATE-PROCESSORS-WITH
      PRENUM!! >=!! (!! PRENUM(N))  AND!! MAXNUM!! <=!! (!! MAXNUM(N))
      AND!! ATTRIBUTE!! <>!! (!! -1)
DO WHILE some processors are still active DO
        ATT := (RETRIEVE-PVAR ATTRIBUTE!! (*MIN SELF-ADDR!!))
        CNT := *COUNT processors with ATTRIBUTE!! =!! (!! ATT)
        DEACTIVATE processors with ATTRIBUTE!! =!! (!! ATT)
        (PRINT ATT CNT)
  END-WHILE
```

6. EXPERIMENTAL RESULTS

We now describe six experiments that analyze the temporal behavior of downward inheritance and upward-inductive inheritance. In the first experiment, 5 class trees (11 levels high), are created such that the number of nodes increases from tree to tree. In every tree the root node is assigned a property and the value of this property is queried at one of the leaf nodes. The purpose of this experiment is to show that the inheritance algorithm is independent of the number of nodes in the tree.

The first column shows the number of nodes in the tree. The second column shows the run times (in seconds) for downward inheritance. Experiments were repeated three times. Times for experiments interrupted by dynamic garbage collections are omitted. Clearly the speed of inheritance is independent of the number of nodes in the tree. The second experiment (third column) uses the same trees with upward-inductive inheritance. One of the leaf nodes receives a property, and that property is queried at the root node. The times required for upward inheritance grow very moderately with the size of the tree.

Number of Sons	Times for Downward Inheritance	Times for Upward Inheritance
1534	0.46 0.47	0.46 0.47 0.46
2558	0.46 0.47 0.48	0.54 0.57
3582	0.47 0.47 0.46	0.68 0.64
4350	0.47 0.45 0.45	0.74 0.72 0.72
5630	0.46 0.46 0.48	0.83 0.81 0.79

In the third set of experiments, 5 trees of nearly equal size are created; however, every tree has a different height. The first column contains the number of levels in the tree, the second the number of nodes, and the third the runtimes. The height of the tree has no influence on the runtime of standard inheritance. The fourth experiment displays the use of the same class trees with upward-inductive inheritance. Runtimes are in the fourth column. Again, tree height has no influence on the time for this operation.

Levels	Number of Sons	Downward Inheritance	Upward Inheritance
5	1016	0.44 0.45	0.41 0.42 0.41
9	1006	0.46 0.45 0.46	0.42 0.43
13	1052	0.47 0.46 0.45	0.44 0.43 0.44
17	1102	0.49 0.47 0.48	0.44 0.45 0.45
21	1131	0.46 0.47 0.46	0.45 0.44 0.44

The fifth set of experiments creates 5 trees with 4 levels each and a different branching factor throughout. The first column shows branching factors, the second the number of nodes in the tree, and the third run times. [3] The times are clearly independent of the branching factor. In the sixth experiment, the same set of trees is used for upward inheritance. Run times are in the fourth column. Times are growing slightly as we would expect from our previous results with growing numbers of nodes.

Branching Factor	Number of Sons	Downward Inheritance	Upward Inheritance
2	14	0.62 0.60 0.63	0.48 0.52 0.48
4	84	0.64 0.63 0.58	0.48 0.43 0.45
6	258	0.60 0.53 0.52	0.45 0.49 0.44
8	584	0.54 0.45 0.46	0.56 0.49
10	1110	0.61 0.59 0.55	0.52 0.59 0.57

REFERENCES:

Evett, M., Hendler, J., and Spector, L. (1990). PARKA: Parallel Knowledge Representation on the Connection Machine, UMIACS-TR-90-22, Department of Computer Science, University of Maryland.
Fahlman, S. E., (1979). NETL: A System for Representing and Using Real-World Knowledge, MIT Press, Cambridge, MA.
Geller, J., (1990). A Theoretical Foundation for Massively Parallel Knowledge Representation. Parallel Computing News, **3**(11), pp. 4-8.
Geller, J. and Du, Charles Y, (1991). Parallel Implementation of a Class Reasoner, Journal paper, in press.
Hillis, W. Daniel (1985). The Connection Machine, MIT Press Cambridge, MA.
Schubert, Lenhart K., Papalaskaris, Mary A. and Taugher, Jay, (1983). Determining Type, Part, Color, and Time Relationships, Computer, **16**(10).
Shastri, L. (1988). Semantic Networks: An Evidential formalization and its Connectionist Realization, Morgan Kaufmann Publishers, San Mateo, CA.
Shastri, L. (1989). Default Reasoning in Semantic Networks: A Formalization of Recognition and Inheritance, Artificial Intelligence, **39**(3), pp 283-356.
Thinking Machines Corporation (1988). *LISP Reference Manual. Thinking Machines Corporation, Cambridge, Massachussetts, version 5.0 edition.
Waltz, David L., (1990). Massively Parallel AI, Eighth National Conference on Artificial Intelligence, Morgan Kaufmann, San Mateo, CA, pp 1117-1122.

ACKNOWLEDGEMENT: Mike Halper has improved the English of this paper.

[3] These experiments were run at the Pittsburgh Super Computing Center. Therefore the times are not identical to the times of the previous experiments which were run at the NPAC center in Syracuse.

Age

The f
the synth
behavior
Afterwar
to constr
argue th
specificat
a sequen
plan of e

Key words an
calculus.

1 Intr

In this paper
the occurren
action execut
all possible r
interval and
author [GW
of two action
A predec
a large class
could not be