A MASSIVELY PARALLEL KNOWLEDGE-BASE SERVER USING A HYPERCUBE MULTIPROCESSOR

Frank Dehne, Afonso Ferreira
and Andrew Rau-Chaplin

SCS-TR-170, APRIL 1990

School of Computer Science, Carleton University
Ottawa, Canada, KIS 5B6

A MASSIVELY PARALLEL KNOWLEDGE-BASE SERVER USING A HYPERCUBE MULTIPROCESSOR*

FRANK DEHNE[1], AFONSO G. FERREIRA[2], AND ANDREW RAU-CHAPLIN[1]

[1] *Center for Parallel and Distributed Computing*
School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6

[2] *Laboratoire de l'Informatique du Parallelisme - IMAG*
Ecole Normale Superieure de Lyon, 69364 Lyon cedex 07, France

Abstract. In this paper we study the parallel implementation of a traditional frame based knowledge representation system for a general purpose massively parallel hypercube architecture (such as the Connection Machine). We show that, using a widely available parallel system (instead of a special purpose architecture), it is possible to provide multiple users with efficient shared access to a large scale knowledge-base server. Parallel algorithms are presented for answering multiple inference, assert and retract queries on both, single and multiple inheritance hierarchies. In addition to theoretical time complexity analysis, empirical results obtained from extensive testing of a prototype implementation are presented.

1 INTRODUCTION

As outlined in [9, 27, 28], massively parallel architectures are essential for computationaly intensive AI applications. Since knowledge representation is an essential part of AI [22, 29], several researchers have studied parallel architectures for implementing knowledge bases [1, 9, 11, 12, 16, 18, 20, 23, 24, 27, 28]. The parallel knowledge representation systems presented in the literature have, however, either been based on special purpose parallel architectures or support only the parallelisation of one query at a time. The latter implies the (economically infeasible) dedication of a massively parallel computer to one single user (e.g. [11, 18]).

This paper is concerned with the design and implementation of a traditional frame based knowledge representation systems [2, 3, 13, 17, 19] on a general purpose massively parallel architecture. The considered architecture is a fine grained hypercube multiprocessor like the 64K processor Connection Machine [15, 27]. We show that, using such a widely available parallel system, it is possible to provide efficient shared access of multiple users to a large scale knowledge base server; see Figure 1.

* The first author's research is partially supported by the Natural Sciences and Engineering Research Council of Canada (Grant A9173). The second author is currently on leave from the University of Sao Paulo (Brazil), project BID/USP, and partially supported by CAPES/COFECUB (Grant 503/86-9). The third author's research is partially supported by the Bell-Northern Research *Graduate Award Program.*

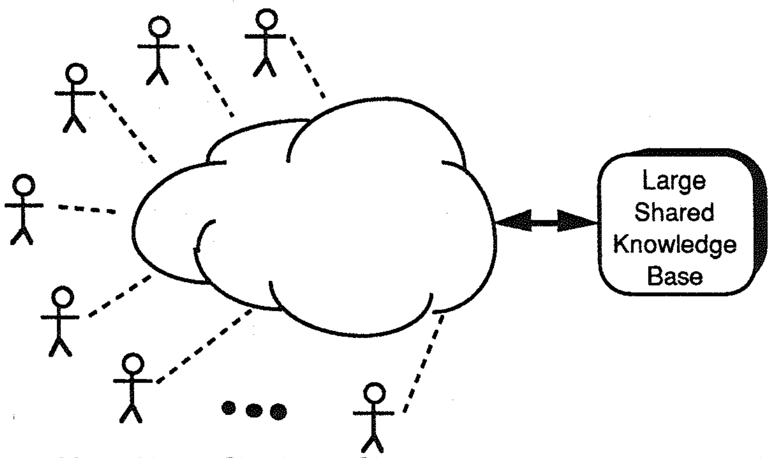

Figure 1. Many Users Sharing a Single Large Knowledge-Base Server.

We consider a parallel implementation of a standard frame based knowledge representation system which answers elementary queries such as top-down and bottom-up inference and assert/retract queries [11] . Such a system could be utilized as the foundational layers of a truly parallel reasoning system, see Figure 2. That is, it could be used as a parallel implementation of Layers 1 and 2 in Figure 2. Parallelization of the higher level layers has already been extensively studied [10, 14, 23]. This could lead to an architecture where Layers 3 and 4 reside on multiple workstations being connected to a central SIMD hypercube multiprocessor which supports layers 1 and 2.

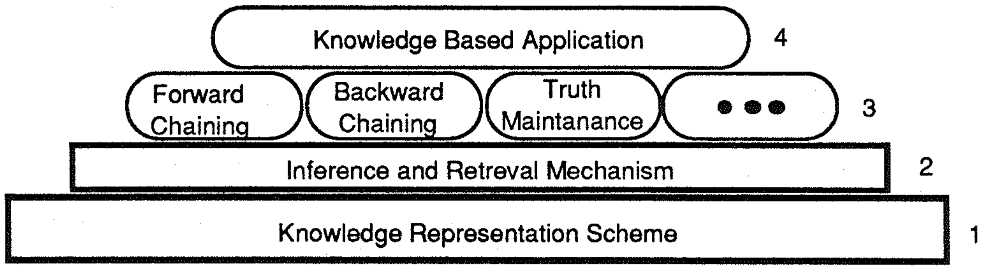
Knowledge Based Application
4
Forward Chaining
Backward Chaining
Truth Maintanance
●●●
3
Inference and Retreval Mechanism
2
Knowledge Representation Scheme
1

Figure 2. A Layered View of a Knowledge-Base Application.

Mores specifically, we show in this paper how to execute in parallel a set of inference and assert/retract queries on a knowledge base (with n frames) stored on a SIMD hypercube multiprocessor (with N=n processors).

In Section 3 we study single inheritance hierarchies with implicit storage [26]. We consider, in Section 3.2, multiple bottom-up inference queries on single inheritance hierarchies. We show that m≤N bottom-up inference queries [11] can be answered, in parallel, in time $O(\log n \log\log^2 n + h \log n)$[1] [or $O(\log n \log\log^2 n + h \log n)$ if frames can have an umbounded number of children], where h is the height of the inheritance hierarchy. In Section 3.3, we present a heuristic algorithm for answering multiple top-down inference queries [11]. Our experimental results, obtained from extensive testing of a prototype implementation, show that a nearly optimal (100%) processor utilization is obtained for a 70% load factor (number of processors divided by number of queries). In our experiments, the utilization never dropped below 75%, regardless of the load factor and other parameters. Our system adapts flexibly and automatically to varying work loads in a close to optimal way (providing a nearly constant product of response time and number of queries). In Section 3.5 we study assert and retract queries, and show that they can be executed in essentially the

[1] Note that, in contrast to [Evett, 1989 #53], our time complexity results also account for the inter processor communication time.

same time complexity as top-down inference queries. Note that, our system can process all four kinds of queries simultaneously.

In Section 4 we generalize our results to multiple inheritance hierarchies with explicit storage [26]. We outline how multiple top-down inference queries, bottom-up inference queries, as well as assert and retract queries can be answered in parallel for a multiple inheritance hierarchy stored on a hypercube multiprocessor. The time complexities for these operations are at most a $O(\log\log^2 n)$ factor larger than the complexities of the respective operations on our parallel single inheritance system.

2 PRELIMINARIES

2.1 FRAME-BASED KNOWLEDGE REPRESENTATION

Semantic nets alternatively known as frame-based systems have been widely studied [1, 2, 3, 9, 11, 13, 17, 19, 24, 25, 26], and several general purpose knowledge representation tools have been designed based on them [2, 3, 11, 13, 17, 25]. There are many advantages to a frame based approach for knowledge representation as detailed in [13, 19].

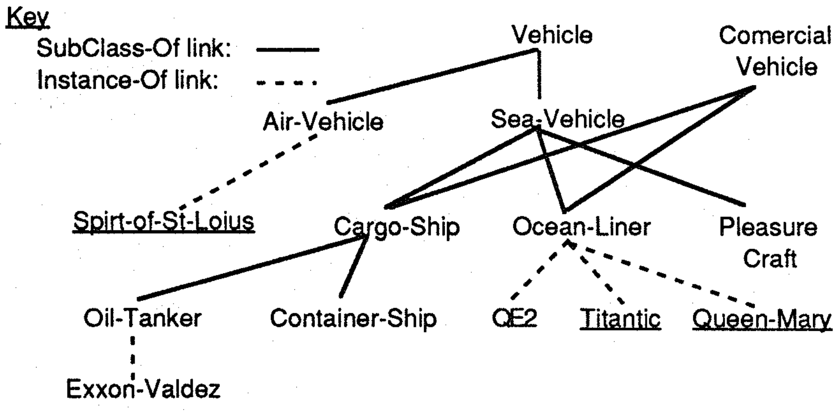

Figure 3. An Example of a Single Inheritance Hierarchy

A frame language provides the designer of knowledge based systems with an easy way to describe the domain objects to be modeled and their relationships. In a frame-based system each frame is used to describe an individual object or a class of objects. For example in Figure 3, the class Ocean-Liner is used as a prototype to describe all of the properties that are common to all Ocean liners, such as the fact that they carry paying passengers. The instance QE2, on the other hand, is a frame that represents an individual instance of the class ocean-liner. It specifies knowledge about a particular ocean liner, the QE2, such knowledge might include the number of passengers that the liner caries, or the QE2's transatlantic crossing time.

Both classes and instances are represented by frames. Each frame consists of a series of slots, where each slot is used to represent a single fact about a particular class or instance. Some slots many be explicit while others may inherit their values from their predecessors in the hierarchy (implicit).

In this paper we will first focus on parallel single inheritance frame based systems. In such systems the inheritance hierarchy can be represented by a k-nary tree. Later, in Section 4, we will show how our approach can be extended to handle multiple inheritance; i.e., the inheritance hierarchy has a more general lattice structure.

2.1 HYPERCUBE MULTIPROCESSOR

A hypercube multiprocessor is a set $P_1, \ldots, P_p$ of p processors connected in a hypercube topology; i.e., P_i and P_j are connected by a communication link if and only if the binary representations of i and j differ in exactly one bit. In a hypercube, there is no shared memory. The entire storage capability consists of constant size local memories, one attached to each processor (thus, $s=O(p)$).

2.3. MULTI-WAY SEARCH ON A TREE

Before presenting our parallel inference algorithms, we introduce some notations and previous results on hypercube algorithms which will be used in the remainder.

Let $T = (V, E)$ be a tree of size k, height h, and out-degree O(1), and let U be a universe of possible search queries on T. A *search path* for a query $q \in U$ is a sequence $\text{path}(q)=(v_1, \ldots, v_h)$ of h vertices of T defined by a *successor* function f: $(V \cup \{\text{start}\}) \times U \Rightarrow V$ (i.e., a function with the property that $f(\text{start},q) \in L_1$ and for every vertex $v \in V$, $(v,f(v,q)) \in E$). A *search process* for a query q with search path $(v_1, \ldots, v_h)$ is a process divided into h time steps $t_1<t_2<\ldots<t_h$ such that at time t_j, $1 \le j \le h$, query q is *matched* with node v_i. A *match* of a query q and a node v_i at time t_j is defined as a situation where there exists a processor which contains a description of both, the query q and the node v_i. Note, however, that we do not assume that the search path is given in advance; we assume that it is constructed during the search by successive applications of the functions f. Given a set $Q = \{q_1,\ldots,q_m\} \subseteq U$ of m queries, $m=O(k)$, then the *multi-way search problem* consists of executing (in parallel) all m search processes induced by the m queries. In [7, 8] it was shown that the multi-way search problem can be solved on a hypercube multiprocessor of size max{k,m} in time $O(\log k \log\log^2 k + h \log k)$. It follows from [4, 7, 8] that for trees with unbounded out-degree, as well as arbitrary graphs, the time complexity increases to $O(\log k \log\log^2 k + h \log k \log\log^2 k)$.

Consider the problem of changing the tree T during the execution of a multi-way search. That is, during the search leaves may be added to and subtrees may be deleted from T, and queries may duplicate or delete themselves when reaching a node of T. This problem is referred to as the *dynamic multi-way search problem.* In [5] it has been shown that this problem can be solved on a hypercube multiprocessor of size max{k,m} in the same time $O(\log k \log\log^2 k + h \log k)$. It follows from [4, 7, 8] that for trees with unbounded out-degree as well, as arbitrary graphs, the time complexity increases to $O(\log k \log\log^2 k + h \log k \log\log^2 k)$.

3 A PARALLEL FRAME BASED KNOWLEDGE SERVER SUPPORTING SINGLE INHERITANCE HIERARCHIES WITH IMPLICIT STORAGE

In this section, we will study the efficient hypercube implementation of a knowledge base server supporting a single inheritance hierarchy. We first describe, in Section 3.1, how to store a frame based system on a hypercube multiprocessor, and then, how this representation can be effectively used for inference. We will be interested in answering two basic types of elementary queries: bottom-up and top-down inference queries [11]. These query are elementary and intended to serve as a base upon which more complex query types can be defined by higher level inference mechanisms (as depicted in Figure 2).

In the following Sections 3.2 and 3.3, we show how multiple bottom-up and top-down inference queries can be processed efficiently in parallel. To simplify exposition, we will describe our inference methods for both query types separately; it is however easy to see that both type of queries can be processed simultaneously.

3.1 STORING AN INHERITANCE HIERARCHY ON A HYPERCUBE MULTIPROCESSOR

We require a scheme for distributing an inheritance hierarchy over the local memories of a hypercube. Consider the *level numbering* of the frames of an inheritance hierarchy as indicated in Figure 4. For the remainder we will assume that each frame with level number i, together with its links and data, is stored at processor P_i.

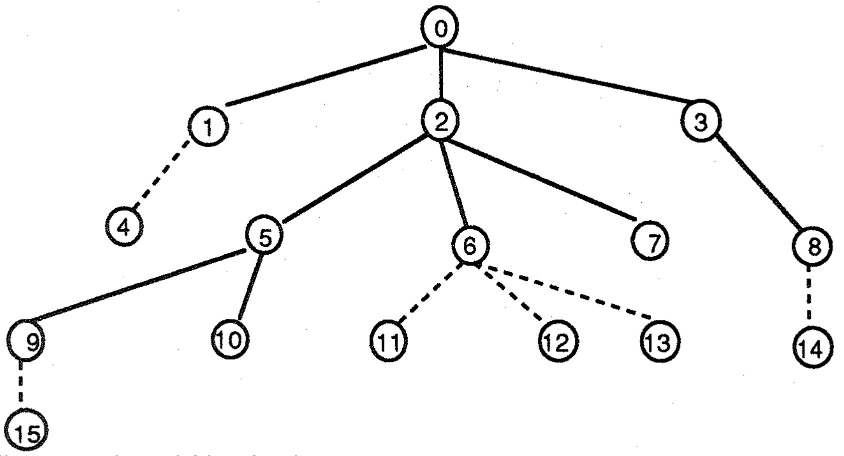

Figure 4. Level Numbering of the Nodes of an Inheritance Hierarchy

In many systems (e.g. [17, 25]), inheritance is "precompiled" such that little or no searching is required to find the value of a slot, even if the value derived from some superclass via inheritance. While we will adapt such an approach for our parallel multi inheritance knowledge base server (Section 5), we will use an implicit representation for the single inheritance system to be discussed in this section. That is, in order to store a frame at a processor, we store only the explicitly valued slots. For implicitly valued slots, there is no reference necessary, since these values will be determined by the inference mechanism.

Frame Name: Oil-Tanker Frame Index: 9 Frame Type: Class
Slots
Name: Cargo Explicet Value: Oil Inherited Value: Unknown
Name: Max-Speed Explicet Value: Unknown Inherited Value: Unknown
Links
SuperClass-Index: 5 SubClass-Indices: None Instances-Indices: 15

Figure 5. A Frame record representing the class Oil-Tankers

3.2 Answering Multiple Bottom-up Inference Queries

We first consider the parallel implementation of multiple bottom-up queries. Each bottom-up query, q(X), is of the form "Does frame X meet conditions a through z" or "What are the values of slots a through z of frame X?". "Is the Exxon-Valdez an vehicle with color black and current-location Alaska?" or "What is the weight the QE2?" are examples of bottom-up queries based on Figure 3. Bottom-up queries are always about a particular instance or class frame but, since we are using an implicit representation, may require the examination of all superclasses of that frame.

Queries will be represented by records of the form depicted in Figure 6. Each "Current Value" field of a query's slot is used to store the value of the respective slot at the frame the query is currently examining.

```
Query Type: Bottom-Up Root Name: Sea-Vehicles Root Index: 2
Slot1:      Name: weight              Current   Value: 2089
Slot2:      Name: paying-Passagers    Current   Value: Unknown
Condition: (weight > 1000) and (paying-passagers = 0)
Completed: False
```

Figure 6. A query record representing the bottom-up query "Identify all instances of Sea-Vehicles with weight > 1000 tons and paying passengers = 0"

Consider an inference hierarchy with n frames, stored on a hypercube multiprocessor with N processors as described in Section 3.1 (w.l.o.g., n = N), and a set of m≤N bottom-up inference queries where each query is stored at one arbitrary processor. For the remainder, *frames(i)* and *query(i)* refer to the query and frame (currently) stored at processor PE(i).

```
Algorithm 1: Answering Multiple Bottom-Up Inference Queries
1)  Match each query q(X) with the frame X it refers to.
2)  As long as there is still a processor PE(i) storing an unanswered query(i),
    repeat the following:
    2a) Every PE(i): If any slot in frame(i) is explicitly valued, and query(i) refers to the same
        slot but is currently unvalued, set the value of that slot of query(i) to the value given in
        the frame.
    2b) Every PE(i): If all necessary slots referred to in query(i) have been instantiated, check
        the condition, report the result, and delete the query.
    2c) Use multi-way search to advance all query(i) with Completed = False one step along
        their path (in the inheritance hierarchy) towards root; i.e., match them with the parent
        of the frame currently examined.
```

Figure 7. Hypercube Algorithm for Answering Multiple Bottom-Up Inference Queries

Figure 7 outlines a hypercube algorithm for answering, in parallel, m bottom-up inference queries. Our method makes extensive use of the multi-way search method outlined in Section 2.3. Note, in particular, the process of advancing all query(i) with Completed = False one step along their path towards root, i.e. matching them with the ancestor of the frame currently examined (Step 2c). As indicated in Section 2.3, it has been shown in [7, 8] that the problem of advancing all m queries one step along their path (in the inheritance hierarchy) towards root [i.e., matching them with the ancestor of the frame currently examined] can be solved on a hypercube multiprocessor of size n in time O(log n) if the number of children of each frame is bounded by a small constant. For unbounded number of children, each advancement all m queries takes time O(log n $\log\log^2 n$) [4, 7, 8].
The remainder of Step 2 consists of simple local, O(1) time, operations. From [7, 8] and [4] it also follows that the initial match in Step 1 can be executed in time O(log n $\log\log^2 n$).

Summarizing, we obtain that all m≤N bottom-up inference queries can be answered, in parallel, in time O(log n $\log\log^2 n$ + h log n) [or O(log n $\log\log^2 n$ + h log n) if frames can have an umbounded number of children], where h is the height of the inheritance hierarchy.

3.3 Answering Multiple Top-Down Inference Queries

Top-down inference queries, q(X), are of the general form "Identify all frames in the subtree (of the inheritance hierarchy) rooted at X such that conditions *a* through *z* are true". For example, "Identify all instances of Sea-Vehicles with weight > 1000 tons and paying passengers = 0", or "Identify all classes who are subclasses of Vehicle and have less-than 10 paying passengers" are top-down inference queries based on the hierarchy in Figure 3.

```
Algorithm 2: Answering Multiple Top-Down Inference Queries
1)  Match each query q(X) with the frame X it refers to.
2)  As long as there is still a processor PE(i) storing a query with Completed = False, repeat the following:
    2a) Every PE(i): If any slot in frame(i) is explicitly valued, and query(i) refers to the same slot but is currently unvalued, set the value of that slot of query(i) to the value given in the frame.
    2b) Every PE(i): If all necessary slots referred to in query(i) have been instantiated, check the condition, and set Completed to True.
    2c) Use multi-way search to advance all query(i) with Completed = False one step along their path (in the inheritance hierarchy) towards root; i.e., match them with the ancestor of the frame currently examined.
3)  Match each query q(X) with the frame X it refers to.
4)  Split each query q(X) into two tokens, a search token search-token and a control token control-token. Each token contains a copy of the original query. Each search token at frame X is responsible for searching the subtree rooted at the first child of frame X; each control token at frame X is responsible for searching (or having searched) the subtrees rooted at the other children of frame X.
5)  As long as there is still a processor PE(i) storing an unanswered query q(X), repeat the following:
    5a) Count the number F of free processors.  A free processor is a processor that is currently not supporting any search token.
    5b) Each token t calculates the number a(t) of assistants it could currently use.  A control token can always use as many assistants as it has remaining subtrees to search.  A Search token can always use as many assistants as there are unsearched subtrees at the frame it is current currently visiting. F new search tokens (assistants) are created and matched with the existing (search and control) tokens in order of the level numbering of their frames, each receiving a(t) assistants until all new tokens are distributed.
    5c) For each token that has been allocated assistants in Step 4, assign a child whose subtree has not been searched yet to each assistant, match the assistants with those children, create for each a corresponding control token, and have them search the respective subtrees.
    5d) Execute "Process-Search-Tokens" as shown in Figure 9.
    5e) Use multi-way search to advance all search tokens.
```
Figure 8. Hypercube Algorithm for Answering Multiple Top-Down Inference Queries

Again, queries are represented by records of the form depicted in Figure 6. Each "Current Value" field of a query's slot is used to store the (implicit) value of the respective slot at the frame the query is currently examining.

Figures 8 and 9 outline our hypercube algorithm for answering multiple top-down inference queries. Again, we assume an inheritance hierarchy of n frames, stored on a hypercube multiprocessor with N processors as described in Section 3.1 (w.l.o.g., $n = N$), and a set of $m \le N$ top-down inference queries where each query is initially stored at one arbitrary processor. Figure 8 shows the general structure of the algorithm. Steps 1-3 are similar to our bottom-up inference algorithm. The result of these steps is that each query, q(X), has for all the slots which are specified in it, explicitly stored the implicit values at frame X. Was is left to do in the remaining steps is to search, for every q(X), the subtree rooted at X. To this end, a search token and control token are created for every query. Each search token, for a query q(X), traverses (independently and in parallel) in preorder the subtree rooted at frame X and determines the answers to be reported; the details are described in Steps 5d and 5e, together with Figure 9. Each control tokens remains at the root of the respective subtree to be traversed, indicates to the respective search token the end of its traversal, and creates new assistant processes in the same way as search tokens do. (Note that, the number of control tokens never exceeds the number of search tokens.) The main idea leading to a near optimal speedup (as will be shown in Section 3.4) is to re-use processors released by queries which need to traverse smaller subtrees to improve the performance of the search processes for the larger subtrees. This rescheduling mechanism is described in Steps 5a-5c. After each "round", i.e. parallel advancement of all search tokens by one edge in the preorder of their subtree, processors from finished traversal processes are given to unfinished traversal processes. Every token determines, from the outdegree of the frame it is currently visiting, how many "assistants" it could currently utilize, i.e. ask them to search those subtrees independently and in parallel. For the distribution of available processors, the following heuristic is used: higher priority is given to those search tokens with smaller level number, i.e. tokens that have (in the expected case) the largest subtrees still to be searched.

For observing the correctness of the algorithm note that, the above algorithm searches for every query q(X) the entire subtree rooted at frame X, and that at every time a frame is examined, all inherited values are present. Due to the rescheduling procedure, the performance analysis for this algorithm is more complicated than in Section 3.2 and will be discussed separately in the next section.

```
Procedure "Process-All-Search-Tokens"
5d) Every PE(i) storing a search token:
    α)  For every slot j in frame(i) which is explicitly valued, if query(i) refers to the same
        slot then
            if   the last node visited by the search token was the parent of frame(i)
            then the inherited value for slot j of frame(i) is the current value of
                 slot j of the search token; the explicit value of slot j (if any) of frame(i)
                 becomes the current value of slot j of the search token; check the query
                 condition and report the result (if condition =true).
            else the explicit value of slot j (if any) of frame(i) becomes the
                 current value of slot j of the search token.
    β)  If the search token has not yet arrived at its control token
        then select, as frame to be visited next, the next node in the preorder traversal
        else if the search token's corresponding control token has additional subtrees to
             be searched
             then start traversing one of those subtrees
             else delete both, the search token and the corresponding control
                  token, and release the processor.
```

Figure 9. Processing of Search Tokens

3.4 ANALYSIS AND EXPERIMENTAL RESULTS

In the previous sections we introduced two inference algorithms. In the case of the bottom-up inference algorithm it was possible to get a worst case bound on the algorithm's time complexity. In the case of the top-down inference algorithm worst case analysis is more difficult. As in Section 3.2, advancing all m tokens one step along their path can be executed on a hypercube multiprocessor of size n in time O(log n) (if the number of children of each frame is bounded by a small constant) or in time O(log n $\log\log^2 n$) (for unbounded number of children) [4, 7, 8]. The problem lies in determining the number of such parallel steps required by our algorithm, since at the heart of the method is a heuristic rescheduling scheme that reallocates processors to queries. The challenge is to quantify how effective this reallocation technique is.

In order to test our mechanism, we have implemented a prototype system and have performed extensive tests using randomly generated hierarchies and sets of top-down inference queries. We considered the following input parameters:

n = number of frames = number of processors,
m = number of queries,
k = max. number of children of a frame.

Figure 10 shows the result of our experiments for n=16,000. The graph on the left depicts results for hierarchies with unbounded k, while the graph on the right shows results for hierarchies with a small value of k (k=8). The x-axis in each diagram represents m, the number of queries, ranging from 1 to 16,000 in 1% increments. For each value of m, 1000 experiments where performed, each with a new randomly generated hierarchy and set of queries. The two curves show the average number of parallel steps as well as the average speed up. The speed up was measured by comparing the number of parallel steps with the total number of steps necessary for sequentially processing the same query set on the same hierarchy. It measures the utilization of the massive parallel architecture and, as our results show, a nearly optimal utilization is obtained for a 75% load factor (number of processors, n, divided by number of queries, m). The utilization never dropped below 75%, regardless of the load factor and other parameters.

The shape of the curves in Figure 10 can be explained by two opposing effects. If there are only a few queries to be processed (small load factor), these can not immediately request enough assistants (due to the constraints of the hierarchy) to utilize all processors. On the other hand, it is important for large subtrees to receive assistants early in the traversal process. Hence, if the number of queries is close to the number of processors, there are no (or only very few) assistants available until the smaller trees have been traversed. Therefore, it becomes likely for the larger trees, that late arriving assistants can not be efficiently applied to the traversal.

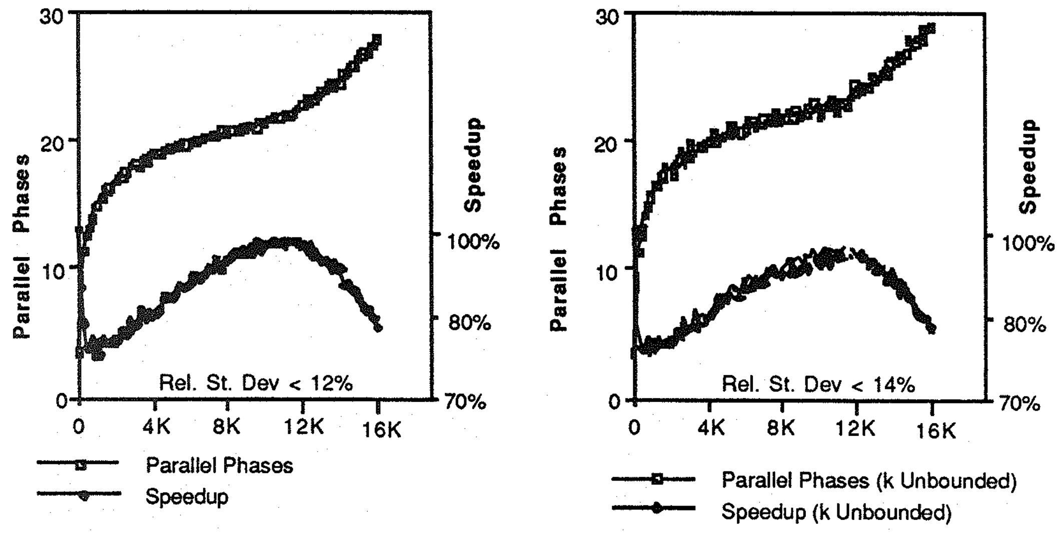

Figure 10. Experimental results for Top-Down inference

3.5. Dynamic Knowledge Representation: Assert and Retract Queries

In Section 3 we have described how m inference queries can be answered efficiently on a static frame based inheritance hierarchy of n nodes. We will now outline how assert and retract queries can also be answered in addition to inference queries, thereby producing a truly dynamic knowledge representation scheme.

Assert queries are queries that add knowledge to our representation. There are three basic types of assertions: assertion of a new slot value, assertion of a new slot (with initial value), or lastly, assertion of a new class or instance frame (complete with slots and values). Retract queries fall into three analogous types: retraction of a slot value, retraction of a slot, and retraction of a class or instance frame.

Assert and retract queries are executed simultaneously to our search queries. From the semantic viewpoint, however, all insert and retract submitted in one round (set of inference/assertion/retraction queries to be processed in parallel) will be performed only after all inference queries in that round have been answered. In addition, assertion and retraction queries processed in parallel must be prioritized. (For example, it is possible that several assert queries may attempt to change the value stored in a particular slot and frame in the same round. Which if any of these changes should have a lasting effect?) Our approach is to assign query priorities based on their position in the set of submitted queries, i.e. queries submitted towards the end of the list of queries (to be processed in parallel) are considered to be executed (logically) after those submitted earlier in the list.

Our strategy for processing assert and retract queries is as follows:

- Match the queries with the respective frames, using [7, 8] and [4].
- Use the concentrate/distribute operations in [21] to remove redundant assert and retract queries.
- Use the parallel tree update procedure in [6] to insert/delete the required frames.

From [4, 6, 21] it follows that the above can be computed, for all assert and retract queries in parallel, in time $O(\log n \log\log^2 n)$

4. MULTIPLE INHERITANCE KNOWLEDGE BASES WITH EXPLICIT STORAGE

While we used an implicit storage scheme for the single inheritance system described above, we will apply an explicit storage mechanism for obtaining a hypercube implementation of a parallel multiple inheritance knowledge base server. That is, we assume that in every frame, each slot which is not explicitly valued contains pointers to all possible slots from whom it can inherit its value, together with a function computing from these values the actual inherited contents. [13, 17, 25]

With such an explicit storage system, bottom-up inference queries can be easily answered by matching the queries with the respective frames, and then matching them with the frames referred to by the pointers in the respective slots. Hence, all bottom-up inference queries can be answered in time $O(\log n \log\log^2 n)$ [4, 7, 8].

For each top-down inference queries q(X), the problem reduces to matching the query with the respective frames X, and then traversing the subtree of all frames who have X as their super class. Despite the fact that this subtree is not any more in level ordering, as for top-down inference queries in Section 3, the traversal algorithm presented in Section 3 can essentially applied to this problem as well. The only difference is that for advancing all search tokens by one step in their preorder traversal, the algorithm in [4] needs to be applied; this results in a time complexity of $O(\log n \log\log^2 n)$ per parallel advancement instead of $O(\log n)$ as in [7, 8]. Otherwise, the same analysis and experimental results as shown in Section 3.4 apply again.

The execution of assert and retract queries becomes obviously more involved than in the implicit storage scheme. In addition to the update/insertion/deletion of frames described in Section 3.5, all possible points to those slots need also to be updated. We observe, though, that for each update/insertion/deletion of X frame X it suffices to traverse the subtree either of all ancestors or of all descendents of X, and update the pointers in those frames' slots. Hence, we obtain a parallel (multiple inheritance) assert/retract algorithm by adding to the assert/retract algorithm in Section 3.5 the same multiple subtree traversals as described in the previous paragraph. That is, again the same analysis and experimental results as shown in Section 3.4 apply.

REFERENCES

[1] L. Bic, "Processing of semantic nets on dataflow systems," *Artificial Intelligence*, Vol. 27, 1985, pp. 219-227.

[2] D. G. Bobrow and T. Winograd, "An overview of KRL, a knowledge representation language," *Cognitive Science*, Vol. 1, 1977, pp. 3-46.

[3] R. J. Brachman and J. G. Schmolze, "An overview of the KL-One Knowledge Representation system," *Cognitive Science*, Vol. 9, No. 2, 1985.

[4] R. Cypher and C. G. Plaxton, "Deterministic sorting in nearly logarithmic time on a hypercube and related computers," to appear in Proc. *ACM Symposium on Theory of Computing*, 1990.

[5] F. Dehne, A. Ferreira, and A. Rau-Chaplin, "Parallel branch and bound on fine grained hypercube multiprocessors," to appear in *Parallel Computing*.

[6] F. Dehne, A. Ferreira, and A. Rau-Chaplin, "Parallel branch and bound on a hypercube multiprocessor," in Proc. *IEEE Int. Workshop on Tools for Artificial Intelligence*, Herndon, VA, 1989, pp. 616-622.

[7] F. Dehne, A. Ferreira, and A. Rau-Chaplin, "Parallel fractional cascading on a hypercube multiprocessor," to appear in Proc. *Allerton Conf. on Communication, Control and Computing*, Monticello, Ill., 1989.

[8] F. Dehne and A. Rau-Chaplin, "Implementing data structures on a hypercube multiprocessor and applications in parallel computational geometry," to appear in *Journal of Parallel and Distributed Computing*.

[9] J. G. Delgado-Frias and W. R. Moore, "Parallel architectures for AI semantic network processing," *Knowledge-Based Systems*, Vol. 1, No. 5, 1988, pp. 259-265.

[10] M. Dixon and J. d. Kleer, "Massively parallel asumption-based truth maintenance," in Proc. *Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988, American Association for Artificial Intelligence, pp. 199-204.

[11] M. Evett and J. Hendler, "Parallel knowledge representation on the Connection Machine," in Proc. *Parallel Computing 1989*, Leiden, The Netherlands, 1989.

[12] S. E. Fahlman, G. E. Hinton, and T. J. Sejnowski, "Massively parallel architectures for AI: NETL, Thistle and Boltzman machines," in Proc. *AAAI Annual Conference on Artificial Intelligence*, 1983, pp. 109-113.

[13] R. Fikes and T. Kehler, "The role of frame-based representation in reasoning," *Communications of the ACM*, Vol. 28, No. 9, 1985, pp. 904-920.

[14] A. Gupta, "Parallelism in production systems,", Carnegie-Mellon, 1986.

[15] W. D. Hillis, *The Connection Machine*(Ed.), MIT Press, USA, 1985.

[16] K. Hwang, J. Gosh, and R. Chowkwanyun, "Computer architectures for artificial intelligence," *Computer*, Vol. 20, No. 1, 1987, pp. 19-27.

[17] IntelliCorp, "*KEE: Core Reference Manual*," 1986.

[18] B. Israel and J. Hendler, "A highly parallel implementation od a marker passing passing algorithm," Tech. Report No. CS-TR-2089, Dept. of Computer Science, University of Maryland, College Park, 1988.

[19] M. Minsky, "A framework for representing knowledge," in P. Winston (Ed.), *In The Psychology of Computer Vision*, McGraw-Hill, New York, 1975, pp. 211-277.

[20] D. I. Moldovan and Y.-W. Tung, "SNAP: a VLSI architecture for artificial intelligence," *Journal of Parallel and Distributed Computing*, Vol. 2, No. 2, 1985, pp. 109-131.

[21] D. Nassimi and S. Sahni, "Data broadcasting in SIMD computers," *IEEE Transactions on Computers*, Vol. 30, No. 2, 1981, pp. 101-106.

[22] A. Newell, "The knowledge level," *Artificial Intelligence Magazine*, Vol. 2, No. 2, 1981, pp. 1-20.

[23] J. Rice, "The advanced architectures project," *Artificial Intelligence Magazine*, Vol. Fall, 1989, pp. 27-39.

[24] P. S. Sapaty, "A wave language for parallel processing of semantic networks," *Comput. Artificial Intelligence*, Vol. 5, No. 4, 1986, pp. 289-314.

[25] M. J. Stefik, M. Bobrow, D. G. Mittal, and L. Conway, "Knowledge programming in LOOPS: Report on an experimental course," *Artificial Intellegence*, Vol. 4, No. 3, 1983, pp. 3-14.

[26] D. S. Touretzky, *The Mathematics of Inheritance Systems*(Ed.), Morgan Kaufmann Publishers, Inc, Los Altos, CA, 1986.

[27] L. Uhr, *Multi-Computer Architectures for Artificial Intelligence*(Ed.), John Wiley & Sons, 1987.

[28] B. W. Wah and G.-J. Li, "A survey on special purpose computer architectures for AI," *SIGART News*, Vol. 4, No. 96, 1986, pp. 28-46.

[29] W. A. Woods, "What's important about knowledge representation?," *Computer*, Vol. 15, No. 10, 1983, pp. 22-29.

School of Computer Science, Carleton University
Bibliography of Technical Reports

SCS-TR-129	**Eva: An Event Driven Framework for Building User Interfaces in Smalltalk** Jeff McAffer and Dave Thomas, November 1987.
SCS-TR-130 Out of print	**Application Frameworks: Experience with MacApp** John R. Pugh and Cefee Leung, December 1987. Available in an abridged version in the Proceedings of the Nineteenth ACM SIGSCE Technical Symposium, February 1988, Atlanta, Georgia.
SCS-TR-131	**An Efficient Window Based System Based on Constraints** Danny Epstein and Wilf R. LaLonde, March 1988.
SCS-TR-132	**Building a Backtracking Facility in Smalltalk Without Kernel Support** See Third International Conference on OOPSLA, San Diego, Calif., Sept. '88. Wilf R. LaLonde and Mark Van Gulik, March 1988.
SCS-TR-133	**NARM: The Design of a Neural Robot Arm Controller** Daryl H. Graf and Wilf R. LaLonde , April 1988.
SCS-TR-134	**Separating a Polyhedron by One Translation from a Set of Obstacles** Otto Nurmi and Jörg-R. Sack, December 1987.
SCS-TR-135	**An Optimal VLSI Dictionary Machine for Hypercube Architectures** Frank Dehne and Nicola Santoro, April 1988.
SCS-TR-136	**Optimal Visibility Algorithms for Binary Images on the Hypercube** Frank Dehne, Quoc T. Pham and Ivan Stojmenovic, April 1988.
SCS-TR-137	**An Efficient Computational Geometry Method for Detecting Dotted Lines in Noisy Images** F. Dehne and L. Ficocelli, May 1988.
SCS-TR-138	**On Generating Random Permutations with Arbitrary Distributions** B. J. Oommen and D.T.H. Ng, June 1988.
SCS-TR-139	**The Theory and Application of Uni-Dimensional Random Races With Probabilistic Handicaps** D.T.H. Ng, B.J. Oommen and E.R. Hansen, June 1988.
SCS-TR-140	**Computing the Configuration Space of a Robot on a Mesh-of-Processors** F. Dehne, A.-L. Hassenklover and J.-R. Sack, June 1988.
SCS-TR-141	**Graphically Defining Simulation Models of Concurrent Systems** H. Glenn Brauen and John Neilson, September 1988
SCS-TR-142	**An Algorithm for Distributed Mutual Exclusion on Arbitrary Networks** H. Glenn Brauen and John E. Neilson, September 1988
	SCS-TR-143 to 146 are unavailable.
SCS-TR-147	**On Transparently Modifying Users' Query Distributions** B.J. Oommen and D.T.H. Ng, November 1988
SCS-TR-148	**An O(N Log N) Algorithm for Computing a Link Center in a Simple Polygon** H.N. Djidjev, A. Lingas and J.-R. Sack, July 1988 Available in STACS 89, 6th Annual Symposium on Theoretical Aspects of Computer Science, Paderborn, FRG, February 16-18, 1989, Lecture Notes in Computer Science, Springer-Verlag No. 349
SCS-TR-149	**Smallscript: A User Programmable Framework Based on Smalltalk and Postscript** Kevin Haaland and Dave Thomas, November 1988
SCS-TR-150	**A General Design Methodology for Dictionary Machines** Frank Dehne and Nicola Santoro, February 1989

SCS-TR-151	**On Doubly Linked List ReOrganizing Heuristics** D.T.H. Ng and B. John Oommen, February 1989
SCS-TR-152	**Implementing Data Structures on a Hypercube Multiprocessor, and Applications in Parallel Computational Geometry** Frank Dehne and Andrew Rau-Chaplin, March 1989
SCS-TR-153	**The Use of Chi-Squared Statistics in Determining Dependence Trees** R.S. Valiveti and B.J. Oommen, March 1989
SCS-TR-154	**Ideal List Organization for Stationary Environments** B. John Oommen and David T.H. Ng, March 1989
SCS-TR-155	**Hot-Spot Contention in Binary Hypercube Networks** Sivarama P. Dandamudi and Derek L. Eager, April 89
SCS-TR-156	**Some Issues in Hierarchical Interconnection Network Design** Sivarama P. Dandamudi and Derek L. Eager, April 1989
SCS-TR-157	**Discretized Pursuit Linear Reward-Inaction Automata** B.J. Oommen and Joseph K. Lanctot, April 1989
SCS-TR-158 (revised)	**Parallel Fractional Cascading on a Hypercube Multiprocessor** Frank Dehne, Afonso Ferreira and Andrew Rau-Chaplin, May 1989 (Revised April 1990)
SCS-TR-159	**Epsilon-Optimal Stubborn Learning Mechanisms** J.P.R. Christensen and B.J. Oommen, June 1989
SCS-TR-160	**Disassembling Two-Dimensional Composite Parts Via Translations** Doron Nussbaum and Jörg-R. Sack, June 1989
SCS-TR-161 (revised)	**Recognizing Sources of Random Strings** R.S. Valiveti and B.J. Oommen, January 1990 Revised version of SCS-TR-161 "On the Data Analysis of Random Permutations and its Application to Source Recognition", published June 1989
SCS-TR-162	**An Adaptive Learning Solution to the Keyboard Optimization Problem** B.J. Oommen, R.S. Valiveti and J. Zgierski, October 1989
SCS-TR-163	**Finding a Central Link Segment of a Simple Polygon in O(N Log N) Time** L.G. Alexandrov, H.N. Djidjev, J.-R. Sack, October 1989
SCS-TR-164	**A Survey of Algorithms for Handling Permutation Groups** M.D. Atkinson, January 1990
SCS-TR-165	**Key Exchange Using Chebychev Polynomials** M.D. Atkinson and Vincenzo Acciaro, January 1990
SCS-TR-166	**Efficient Concurrency Control Protocols for B-tree Indexes** Ekow J. Otoo, January 1990
SCS-TR-167	**A Hierarchical Stochastic Automaton Solution to the Object Partitioning Problem** B.J. Oommen, January 1990
SCS-TR-168	**Adaptive List Organizing for Non-stationary Query Distributions. Part I: The Move-to-Front Rule** R.S. Valiveti and B.J. Oommen, January 1990
SCS-TR-169	**Trade-Offs in Non-Reversing Diameter** Hans L. Bodlaender, Gerard Tel and Nicola Santoro, February 1990
SCS-TR-170	**A Massively Parallel Knowledge-Base Server using a Hypercube Multiprocessor** Frank Dehne, Afonso Ferreira and Andrew Rau-Chaplin, April 1990
SCS-TR-171	**Parallel Processing of Quad Trees on the Hypercube (and PRAM)** Frank Dehne, Afonso Ferreira and Andrew Rau-Chaplin, April 1990