LATEX documents for the Machine Intelligence
and Pattern Recognition series
W.j. Maas

©Elsevier Science Publishers B.V. All rights reserved

defProviding Computationally Effective Knowledge Representation via Massive Parallelism

Matthew P. Evett*, William A. Andersen† and James A. Hendler‡

Department of Computer Science, University of Maryland, College Park, MD 20742, USA

Abstract

PARKA is a frame-based knowledge representation system implemented on the Connection Machine. PARKA provides a representation language consisting of concept descriptions (frames) and binary relations on those descriptions (slots). The system is designed explicitly to provide extremely fast property inheritance inference capabilities. In particular, PARKA can perform fast "recognition" queries of the form "find all frames satisfying m property constraints" in $O(d + m)$ time—proportional only to the depth (d) of the knowledge base (KB), and independent of its size. For conjunctive queries of this type, PARKA's performance is measured in tenths of a second, even for KBs with more than 100,000 frames. We show similar results for timings on on large IS-A networks derived from the Cyc commonsense KB, and for queries involving knowledge structure pattern matching in support of case-based planning. With such run-time performance, PARKA is possibly the "fastest knowledge representation system in the world".

1. Introduction

Currently, AI is experiencing a period of soul-searching. Critics contend that the promise of the AI techniques of the 80's evaporated because those techniques did not deliver. It wasn't that their formalisms and theory were unacceptable (in fact, they worked fine for relatively small, contrived domains). Real-life domains, however, are orders of magnitude larger, and the run-time performance of these earlier AI techniques is often completely

*evett@cs.umd.edu
†waander@cs.umd.edu
‡hendler@cs.umd.edu

unacceptable for such domains (and even much smaller ones); that is, these techniques, while quite useful, are *computationally ineffective* [18].

The field of knowledge representation (KR) is one area where the techniques of the 80's did not deliver acceptable performance. Most existing KR systems require orders of magnitude more run-time than do humans for certain classes of queries. One of the most striking examples of such a class is that of *recognition*, the problem of identifying those frames that satisfy a given set of property constraints. For example, "Find all x such that x is yellow, alive, and flies." Human agents seem to answer such questions (at least the existential versions) almost effortlessly, usually within only a few tenths of a second. Shastri connotes this effortlessness by calling such human inferencing "reflexive"[19].

Existing KR systems efficiently solve the converse problem—retrieving the properties of any given frame—but cannot do the same for recognition. In general, they answer recognition queries by traversing the entire knowledge base (KB), collecting the set of frames satisfying the given constraints. Their run-time for recognition queries is no better than linear in the size of the KB, i.e., $O(n)$. Some systems can require minutes to answer such queries.

Of course, some KR systems do provide computationally effective recognition queries in certain situations. On the other hand for any existing KR system there are bound to be many computationally ineffective query classes. Recognition queries are merely an example of how many systems of the 80's failed to live up to their advance billing.

The poor performance of existing KR systems is especially unfortunate, because KR is a core component of many AI applications. Because these applications can't perform any better than their weakest link, their overall performance consequently suffers.

For several years, it has been our goal to design a KR system with computationally effective inferencing on knowledge bases large enough to support real-world, commonsense reasoning. We believe such a system can serve as a basis for the implementation of realistic AI applications requiring rapid response time. Our system, PARKA, is a symbolic, frame-based KR system that uses the Connection Machine's (CM) massive parallelism to deliver sufficient run-time performance.

In this paper, we discuss the design and implementation of PARKA. Then, we present a short analysis of the expected and observed run-time performance of several different PARKA operations upon very large knowledge bases. We argue that PARKA's performance demonstrates that it can offer computationally effective knowledge representation on realistically large knowledge bases.

2. Description of PARKA

PARKA was designed as a general-purpose KB package, for use by other AI systems. For PARKA, we chose to use a frame-based symbolic representation paradigm for several reasons. First, frame systems have been used in KR for years, and remain a common paradigm in contemporary AI research. Second, many other representation schemes are merely notational variants ([8, 2] are just two). Our PARKA research and results should be extensible to those paradigms. Third, semantic nets readily lend themselves to the data-level parallelism design so important in efficient parallel implementation.

Most KR systems have two primary goals: expressiveness and formality. The authors of these systems want to express as many semantic concepts as possible with an unambiguous, rigorous formalism. For them, run-time efficiency is of secondary importance. They emphasize classic search and rule-based approaches, consequently suffering run-time that is at best linear, and at worst exponential, in the size of the problem. While these methodologies have a place in AI, their application to large KB's leads to unacceptable run-time performance, and it is expected that realistic KB's will be very large indeed—on the order of 10's or 100's of millions of frames ([14, 22], etc.) Such KB's would be orders of magnitude larger than any existing today, and would certainly render existing serial KR systems completely unusable.

We designed PARKA with run-time performance rather than expressiveness as the primary goal of the system. We built only as much representational capability into the system as would still allow computationally effective inferencing. Restricting expressiveness to achieve tractability is an undesirable but necessary practice that has become fairly common in the KR community. It is unavoidable because many operations, are, in general, NP-hard or even undecidable for term-subsumption languages that are sound and complete. Recognition, for example, is NP-hard, though it is a special case of classification, which can itself be undecidable [16]. So, although PARKA's semantics are roughly based upon those of NETL [8] and KL-ONE [2], we avoided semantic constructs lacking a computationally effective implementation. We believe PARKA's superb run-time performance on even very large KBs more than compensates for its slighlty restricted expressiveness when compared to that of other, serial, KR systems.

Our bias toward run-time performance over general expressiveness derives from our interest in memory-based reasoning[1] (MBR). We are strong

[1]We use the term "MBR" in a broader sense than do Stanfill & Waltz [22], to include

supporters of the MBR paradigm and its importance in future AI work. We believe that memory intensive operations form the pillar of most intelligence tasks, sharing this belief with many others ([10–12, 22] are just a few). This idea runs counter to those in the AI community who emphasize search, rule-based and theorem-proving approaches, couched in conventional logics. As we point out above rule-based serial systems suffer run-time that is at best linear and at worst exponential in the size of the problem, leading to unacceptable run-time performance on realistically large KBs. In contrast, the ability to access quickly a large collection of exemplars of previously categorized episodes in memory is the hallmark of the MBR paradigm. PARKA was designed to directly support exactly this type of fast access to such KBs.

2.1. Design Specifics

PARKA is a basic frame system: each frame corresponds to a concept represented in the KB and the collection of relations to which it belongs. Relations among concepts are represented as directed graphs whose arcs (or links) are stored as frame pointers in the *slots* of frames. Properties of frames are represented as relations, the domains of which are the frames having the property, and the ranges are the corresponding property values. The KB can also be viewed as the network formed by the frames and the links that connect them, similar to a semantic network ([8], e.g.)

Ontological relations (related to the study of "natural kinds", in philosophy) among frames are encoded via the IS-A ("is a") relation. The IS-A relation is intimately related to the calculation of property inheritance inferences (see below) and, as such, is accorded special status in PARKA. The subgraph consisting of all IS-A links and frames is referred to as the *IS-A hierarchy* of the net. Nomenclature similar to that of tree structures refers to elements of the hierarchy: for any node in the subgraph, adjacent node(s) representing superclass(es) are referred to as *parents*; those representing subclasses are called *children*. Likewise, nodes above and below any node on the IS-A hierarchy are called its *ancestors* and *descendants*, respectively. All PARKA IS-A hierarchies are rooted and acyclic.

A small subset of a frame network is shown in figure 1. The frames are shown as ovals. The properties of each frame are represented by the arcs emanating from them. Unlabelled arcs are IS-A links.

---

such paradigms as case-based reasoning.

Fig. 1. Subset of a Frame Network

2.2. Inheritance Mechanism

PARKA employs a property inheritance mechanism on its IS-A hierarchy. A frame is said to be *explicitly-valued* for a given property if it is incident on a property link of that type—that is, if the frame contains a slot by that name. Any frame not explicitly-valued for a given property *inherits* the value of its nearest ancestor(s) (using a variant of Touretzky's *inferential distance ordering* (IDO) scheme) that is explicitly-valued for that property. If there is more than one such node (which is possible because PARKA allows multiple inheritance), the inherited value is either ambiguous, or is arbitrarily chosen from among the possible candidates.

The first version of PARKA [5, 7] handled multiple-inheritance—as have many previous frame-based KR systems ([2] is just one)—by using inheritance path length to disambiguate multiple inheritance paths. This paradigm suffers from redundancy, ambiguity, and other problems, and has been soundly criticized in the philosophical community as being epistemologically inadequate. Many of these criticisms have been detailed by Touretzky [24] and Brachman [1].

To address the problems of multiple inheritance, the current implementation uses a top-down, path-based, credulous inheritance mechanism based on the IDO metric to disambiguate multiple inherited values: Assume the frame in question, X, is not explicitly valued for the given property, P. Let $\mathbf{B}$ be the set of ancestors $\{B_1, B_2, ...\}$ of X that are explicitly valued for P. X takes B_i's value for P as its own, provided B_i is an element of $\mathbf{B}$ such that there is no $B_j (j \neq i)$ that is an IS-A descendant of B_i. If more than one element of $\mathbf{B}$ meets this criterion, X is said to be ambiguously valued for property P.

Unfortunately, many retrieval operations involving top-down, path-based inheritance mechanisms, including IDO, have been shown to be NP-hard [17]. To calculate these operations in a timely manner, we adopted a slightly weaker ordering scheme for inheritance disambiguation. Again, let $\mathbf{B}$ be the set of ancestors of X that are explicitly valued for property P. X

takes B_i's value for P as its own, where B_i is that element of $\mathbf{B}$ with the largest topological number. The topological number, topo(Z), of a frame Z, is defined inductively: topo(RootNode) $= 0$ and for all other frames, Z, $\text{topo}(Z) = 1 + \max(\text{topo}(y)), \forall y \in \mathbf{C}$, where $\mathbf{C}$ is the set of frames that are parents of Z.

Though PARKA's disambiguation mechanism is not quite as powerful as complete IDO, it enjoys many of the same advantages and is considerably stronger than a simple path-length based scheme. It does not suffer from the problems of redundancy noted by Touretzky [24] and in only one case does our inheritance scheme differ from IDO: in full IDO, if there are two explicitly valued ancestors, X and Y, but X is also an ancestor of Y, then Y is "more specific" than X, and so its property value is chosen. Our topological disambiguation scheme, however, may arbitrarily disambiguate among the two ancestors, even when neither is an IS-A ancestor of the other. In the vast majority of cases, though, PARKA's mechanism is equivalent to IDO. Our experience with PARKA's encoding of the Cyc commonsense KB (see section 3.3) revealed no cases in which PARKA disambiguates an inheritance relation that is ambiguous via IDO. If Cyc turns out to be typical of future KBs, this shortcoming of topologicial disambiguation will have little impact on most inferences.

2.3. Implementation

PARKA's internal representation of a frame consists of a block of processors, one for each of that frame's IS-A parents. These processors are contiguous across the CM's processor address space. One processor of each block is distinguished as a referent for all other frames pointing to the represented frame.

The slots of each frame are encoded in a slot table, stored in one of the processors of that frame's block. The table is a list of pairs $(\langle\text{property}\rangle, \langle\text{propertyValue}\rangle)$, each component being a processor address. Figure 2 illustrates the internal representation on the CM of part of the net shown in figure 1. The large rectangles represent the distinguished processor of each block of processors corresponding to a particular frame. The smaller rectangles represent the remaining processors of those blocks. The square-bracketed values represent the address of the processor representing the property of that corresponding name.

To determine if a frame, Y, is explicitly valued for a given property, P, PARKA determines the (explicit) value of P for every frame in the KB. This process is called "slot retrieval" and is accomplished by scanning through every frame's slot table in parallel, seeking an entry corresponding to P. If a matching entry is found in Y's slot table, the corresponding value

Fig. 2. Internal CM representation of asmall subset of a frame network

stored there is the address of the processor representing the frame that is the value of property P for frame Y. This value is stored in a temporary (parallel) variable that will be returned as the value of P for all frames in the KB.

In general, slot retrieval is proportional to the size of the largest slot table in the KB. Because property values tend to be scattered across inheritance paths, these tables are typically quite small. The largest slot table in our encoding of the Cyc KB had only 43 entries. Even so, PARKA maintains a cache of the most recently accessed properties to accelerate explicit property look-up.

3. Performance

To demonstrate that PARKA can provide computationally effective knowledge representation, we implemented and timed a number of retrieval operations on very large KBs. Tests included inheritance queries on very large pseudo-random KB's, conjunctive queries on the Cyc KB, and structure matching retrievals on a large KB supporting a case-based planner.

3.1. Inheritance Queries

Almost all KB queries involve some inferencing along the IS-A hierarchy because almost all involve calculating the inherited value of a set of properties for a set of frames. PARKA uses several data structures to make such inheritance inferencing very fast, including (as described above) using multiple processors to represent frames having multiple parents. PARKA uses an activation wave propagation algorithm to calculate the value for a given property of every frame in the KB in time independent of the size of the network.

At the start of the algorithm, the IS-A root frame is "activated", forming a nascent activation wave. At each iteration, this wave is passed downward along IS-A links. The activation wave propagates synchronously; all nodes in the current "wave front" simultaneously activating the incident nodes

At propagation step i, $0 \leq i \leq d$, where d is the depth of the IS-A hierarchy:

1. Frames with a topological value of i (i.e., those at topological level i) that are explicitly valued for the property set their wave value to k, where k's j high order bits are i, k's low order bits are the frame's property value, and $j = \lceil \log d \rceil$.
2. Every frame (processor) not explicitly valued for the property in question that has parent nodes at topological level $i - 1$ "pulls" down the value of the activation wave from those parents.
3. Non-explicitly valued frames at topological level i choose as their own activation wave value the largest of those pulled down from the parent frames. Because the high order bits of each wave value are the topological level of the origin node, the selected value conforms to the inheritance scheme outlined in section 2.2.

Fig. 3. The Basic Inheritance Algorithm

that have not yet been activated. Each such iteration is a propagation step. Because PARKA is implemented on the CM, each propagation step is accomplished with a single parallel operation. The algorithm is described in detail in figure 3.

The number of propagation steps required to calculate a property value is equivalent to the depth, d, of the network's IS-A hierarchy, i.e., the longest explicit IS-A path in the KB. Consequently, PARKA's run-time for queries such as "what things are black?", is $O(d)$, and is independent of the size of the KB[2]. We refer to such queries as "top-down", and they are the bane of most serial KR systems, requiring $O(n)$ time to effect, where n is the size of the KB. Serial systems use indexing schemes to mitigate this computational morass, but indexing can be unsatisfactory for a variety of reasons (as we discuss in [11]) including that typically it is infeasible to explicitly index all properties.

The comparison between serial and parallel run-times is more striking when realizing that for realistic networks $d \approx \log(n)$. It is commonly believed that such network shallowness will persist and probably be accentuated as net size increases. Our PARKA implementation of the Cyc

[2]Run-time figures throughout this paper presume that the number of frames is no more than some small multiple of the number of processors.

Fig. 4. Run-time performance of inheritance queries on 8-level networks of varying sizes.

commonsense KB (see section 3.3) enjoys a similarly shallow IS-A topology.

To determine how PARKA would compare against a serial representation system, we created a serial version of PARKA, called SPARKA ("serial-PARKA"). To make the comparison of the timings between PARKA and SPARKA as fair as possible, we implemented SPARKA as a severely stripped-down version of a more complete serial implementation (detailed in [21]). It has very little functionality other than for simple property inheritance calculations, but is optimized to effect those calculations as quickly as possible.

We tested our analytical predictions of PARKA's run-time performance of simple property inheritance queries by timing PARKA's response to sample queries on topologies of varying size and depth. Then, we timed SPARKA on the same queries and networks. The networks were quite large—up to 128K nodes[3]. Because encoding such large networks by hand was not possible, we developed algorithms for generating pseudo-random networks with certain topological characteristics. These techniques are described in [5, 7]. Our experience with the Cyc KB (see section 3.3) has affirmed our belief that the topologies used to measure PARKA's performance reflect those of realistic KBs.

Figure 4 shows PARKA's run-time for frame networks of depth 8, and of varying size. This timing suite isolates the effect of network size on run-time. These timings support our supposition that PARKA's computation of inheritance queries is independent of network size[4]. Thus, PARKA's performance should scale up to arbitrarily larger KBs. The serial system's run-time, on the other hand, was linear with respect to network size. The figure contains best-fit curves to highlight this linear relation.

[3] Timings were made on a "quarter" CM-2, consisting of only 16K processors.

[4] Actually, we observed a correlation between PARKA's run-time and the size of the networks. We examined this degradation away from our theoretical performance predictions (the results of this study are detailed in [4]). The run-time of these operations degrades linearly with processor utilization, though the constant of linearity is very small relative to the number of processors.

The networks used in the timings were of two topological types: trees (each frame having exactly one parent) and directed graphs (each frame having between one and four parents.) We used different types of topologies to demonstrate that PARKA's run-time was independent of upward IS-A fan-out.

A comparison between the performance figures of SPARKA and PARKA in figure 4 demonstrates that the latter remains computationally effective even for very large KBs, while the former's performance is unacceptable for large networks. We anticipate that this contrast will become increasingly stark for the much larger KBs in applications involving real-world domains. (Lenat & Guha suggested that such KBs will contain tens of millions of nodes [14].)

3.2. Recognition Queries on the Cyc KB

The ability to solve recognition queries has driven much of PARKA's design. The problem of recognition is well-known in the field of KR (e.g., [26]) and is the problem of answering KB queries of the form: "find all frames x such that $P_1(x, c_1) \wedge P_2(x, c_2) \wedge \ldots \wedge P_p(x, c_p)$," where $P_i(z, c_i)$, $\forall i$, is a unary predicate true for all frames, z, that have value c_i for property P_i. E.g.: "What object is most characterized by this list of property values? ...." and "what have trunks, tusks, and are big and gray?" Such queries are extremely time-consuming for serial-based systems, often running in time no better than $O(pn)$, even on systems employing a highly constrained description language.

PARKA's ability to execute inheritance inferences quickly makes it particularly suitable to recognition. Because PARKA determines which objects have a given value for a given property in $O(d)$ time, PARKA determines which objects satisfy a set of p property constraints in no more than $O(dp)$ time, where d is the depth of the net.

But PARKA does even better, using a pipelining technique to evaluate recognition queries in time $O(d+p)$. Because each wave propagation step of an inheritance inference occurs "in lockstep"—all frames at the same topological level calculating their property value simultaneously—pipelining can be added to the basic process outlined in figure 3. At each propagation step i, all frames at topological level j, such that $i-(p-1) \leq j \leq i$, retrieve from their parent(s) the wave activation value having to do with the $(j-i)$–th element of the set of properties being inferred. In other words, at the time when the nodes in topological level j are calculating their value for the lth of the p properties, the nodes at topological level $j-1$ are computing the $(l-1)$th property, and those at the $(j+1)$th level the $(l+1)$th. Thus, the complete propagation requires $d+p-1$ propagation steps. Full implementation details can be found in [4].

PARKA's $O(d+p)$ performance compares very favorably with recognition queries on serial systems, which require $O(pb^d)$ time for the same queries, where b is the average fan-out of the IS-A hierarchy. We tested these predictions by timing PARKA's performance for recognition queries containing a varying number of conjuncts on an implementation of the Cyc KB. The results support our run-time analysis and are discussed in the next two sections.

3.3. Using Cyc for Validation of PARKA

We tested our run-time predictions for recognition queries by timing PARKA's performance on a version of the Cyc KB [14] converted for use with PARKA. Our motivation for using Cyc to evaluate PARKA's performance is twofold. First, we want to validate PARKA's inference mechanisms on a KB of large size and realistic topology. Because Cyc is the largest and most comprehensive commonsense KB in existence, it is an obvious choice. A second and more exciting motivation is that we envision some future version of Cyc being built on top of a massively parallel substrate, like PARKA, to make its reasoning services fast enough to be used by an intelligent agent operating in the world in real time.

The Cyc KB consists of a set of objects (called "units" in Cyc) and assertions involving those objects. Predicates are explicitly represented as units in Cyc. To represent assertions from the Cyc KB in PARKA, we considered only binary predicates (slots) and mapped the Cyc units to PARKA frames[5].

The inference mechanisms provided by Cyc range from the very simple, such as the "more general predicate" mechanism such as

$$\text{brother}(\text{John}, \text{Mary}) \Rightarrow \text{sibling}(\text{John}, \text{Mary})$$

to theorem proving using general "wffs", to unsound inference mechanisms such as analogy.

PARKA implements only some of the inference capabilities provided by Cyc, particularly those having to do with property inheritance. We represented only that subset of the Cyc KB involving the ontology of collections, properties, and the inheritance of properties. Cyc distinguishes between "is a subset of" and "is an instance of" relations, but for our tests these relations were collapsed into a single IS-A relation.

[5] The conversion process was particularly straightforward because the version of Cyc (Cyc-8) we were using had a frame system as its underlying implementation. The current version of Cyc (Cyc-9), however, has a different, assertion-based implementation.

The copy of the Cyc KB that we used for our tests contained a total of 26216 units, 8591 (33%) of which were collections, and 17623 (67%) instances. Of the instances, 4031 (15% of the total) were predicates. We recovered approximately 120,000 property assertions. To accommodate a KB of this size, we used a 16,384 processor CM-2 with a virtual processor ratio of 4:1, simulating 65,536 processors. The maximum depth of the KB along the IS-A relation was 23, (i.e., shallow relative to KB size, as expected.)

3.4. Performance of Recognition Queries

To test recognition query performance, we timed queries similar to those used by Cyc to find units "similar" to a given unit. Units are considered similar if they share the same values for a number of properties exceeding some threshold. First, we selected a Cyc unit (#%Burma-1986) with a relatively large number (22) of local assertions (i.e., explicitly-valued properties), assigning an arbitrary ordering to those properties. We then ran recognition queries in PARKA to identify those frames that matched at least 50% of the first n slots ($1 \leq n \leq 22$) of #%Burma-1986. The recognition queries themselves, then, involved between 1 and 22 conjuncts. The run-time performance of these queries is plotted in figure 5.

As figure 5 clearly shows, the time required to perform recognition queries grows only linearly in the number of conjuncts, p, and overall performance, even for a query of 22 conjuncts, is excellent. The run-time matches the $O(d + p)$ performance predicted by analysis. This performance compares very favorably with recognition queries on serial systems, which require $O(pn)$ time for the same queries, where n is the size of the KB. Recognition queries in PARKA are independent of KB size, and should scale up to arbitrarily larger domains. Indeed, in [11] we report sub-second run-time performance of recognition queries in a case-based planning system using KB's of over 10,000 frames. We achieved run-time performance 10,000 times better than that of a highly optimized serial version of PARKA [6].

This experiment was designed not only to demonstrate the $O(d + p)$ complexity of conjunctive recognition queries in PARKA, but also to show that PARKA can supply fast matching for analogy-related functions, a task that traditionally has been difficult for serial systems. For example, the Cyc query that most nearly corresponds to those of figure 5 finds only

[6] We purposely avoid the term "speed-up" here because of its connotations in the technical literature of parallel computation. We ran the serial version on a Macintosh IIcx computer, the performance of which will naturally pale in comparison to that of a CM. Even accounting for the disparate computational platforms, though, the performance differential is striking.

Fig. 5. Run-time of recognition queries of various sizes on the PARKA implementation of the Cyc KB

an arbitrary subset of the matching units. By exhaustively matching the probe against the entire KB simultaneously, PARKA finds all appropriate matches.

3.5. Structure Retrieval for Case-Based Planning

As another part of our demonstration of PARKA's computational effectiveness as a KR system for very large AI applications, we implemented a general structural matching mechanism. PARKA's structural retrieval mechanism has been used to support CaPER, a case-based planning system [11].

Structure retrieval in PARKA is a generalization of recognition, and is closely related to the problem of computing subgraph isomorphism. The problem of structure retrieval is to find the set of all m-tuples $(x_1, x_2, \ldots, x_m)$, where x_i are frames, such that

$$P_1(x_{f(1)}, x_{g(1)}) \wedge P_2(x_{f(2)}, x_{g(2)}) \wedge \ldots \wedge P_l(x_{f(l)}, x_{g(l)}) \\ \wedge Q_1(x_{f(1)}, c_1) \wedge \ldots \wedge Q_k(x_{f(k)}, c_k)$$

where P_m, $1 \leq m \leq l$, are binary predicates between specified tuple components, and Q_m, $1 \leq m \leq k$, are unary predicates on the specified tuple component, and where f and g are functions mapping $\{1, 2, \ldots, l\}$ to $\{1, 2, \ldots, m\}$ and h is a function mapping $\{1, 2, \ldots, k\}$ to $\{1, 2, \ldots, m\}$.

The problem can be viewed as that of finding all subgraphs (m-tuples) in a KB frame network that match a specified probe graph (defined as the conjunction of the binary and unary predicates). The KB is viewed as a graph, where the vertices are frames, and the edges are the 2-tuples comprising the binary relations represented by slots. A probe graph consists of a set of variables, a set of unary constraints, and a set of binary constraints. The unary constraints restrict the domains of the variables. The binary constraints specify relations (in terms of inherited property values, etc.) that must hold between frames bound to the variables. The algorithm returns all subgraphs that simultaneously satisfy all constraints. Note that this algorithm only returns exact matches. Future implementations will provide inexact matching capabilities with programmable similarity metrics for analog retrieval.

PARKA uses the pipelining technique outlined in section 3.2 for the retrieval of all components of the probe graph having to do with property inheritance, set membership, and subclass/superclass membership. The algorithm uses a parallel relational join algorithm combined with parallel constraint propagation to combine these results.

We have not yet completed a theoretical analysis of the PARKA's run-time performance of structure retrieval. Though the general problem of subgraph isomorphism is known to be in NP, empirical evidence indicates that the topology of actual knowledge bases is such that the worst case is never encountered and that average case performance will be consistently good.

We compared PARKA to a serial KR system on the same KB to demonstrate the advantages provided by the massively parallel PARKA system. For the serial system we used an optimized serial version of PARKA. The serial version was highly-indexed to allow for fast retrievals of all those frames being valued for a given property.

We timed several structure retrieval probes used by CaPER to retrieve plan descriptions from five case bases constructed for a simplified transportation logistics domain. These case bases consisted of between 20 and 100 cases (respectively between 220 and 1100 subplans and between 3,000 and 10,000 frames). The case bases were generated automaically using the Nonlin planner [23]. The most complicated query on this case base required 1.5 seconds of CM time to compute. Future implementations are expected to reduce retrieval times to under 500 milliseconds for comparable probes through the use of more pipelining and caching techniques. The PARKA timings for this and several other probes on these KBs are shown in figure 6. The same query probes were run on the serial implementation of PARKA; the results are shown in figure 7.

As can be seen, the run-time for the serial structure retrieval algorithm was exponential in the size of the KB. The run-time of the parallel algorithm, on the other hand, appeared to grow only logarithmically in the size of the KB. More detailed analysis will be necessary to fully account for the observed performance.

Because retrieval probes can be specified over any relation in the KB, this algorithm allows cases to be retrieved on arbitrary features, not restricted to any set of predetermined indexes. Because of this, there is no need for a privileged set of indexing features by which cases can be found in memory. Most serial CBR systems rely on elaborate indexing techniques to identify matching cases rapidly. This freedom from a need for *a priori* computed

Fig. 6. Structure retrieval parallel run-time performance

Fig. 7. Structure retrieval serial run-time performance

indexing features is one of PARKA's strengths. Because of it, CaPER is not hindered from retrieving potentially useful cases that share unindexed features with the target problem (other benefits are discussed in [11].)

We know of only one other massively parallel system that provides something like structure retrieval. DeMara and Kitano describe a generic pattern matching algorithm [10] running on Moldovan's SNAP system [15]. The "pattern matching" problem shares some similarities to that of structure retrieval, but is considerably different. The algorithm listed there is for finding matches against 4-node subgraphs, and it does so very quickly due to the special-purpose hardware on which SNAP is implemented. Our algorithm works for arbitrarily specified graphs, and runs on general-purpose hardware.

4. Future & Related Work

PARKA is designed for use by AI applications needing a general purpose KR system. By using PARKA, an AI application can leverage the speed and power of massive parallelism into its own performance, without having to adopt any parallel programming constructs on its own. The CaPER system discussed in section 3.5 is such an application. CaPER was designed and implemented by a programmer having no knowledge of the internal workings of PARKA or of the CM. Yet the PARKA-based CaPER system is orders of magnitude faster than the CaPER implementation that is based on a standard serial representation syststem.

Currently we are conducting a detailed examination of the efficacy of PARKA in CaPER. In particular, we are identifying those operations that would have simplified CaPER's use of PARKA and those that would have permitted the implementation of certain desirable functions that were unattainable given PARKA's current external interface. Having identified these operations, we will determine what, if any, of them should be implemented in future versions of PARKA so as to facilitate the use of PARKA by other AI systems.

Other potential uses for PARKA include acting as a massively parallel knowledge server, and as part of a knowledge-mining system. We also plan to examine how PARKA might be more fully integrated into the Cyc representation system, proper. Our current research is mainly concerned with an implementation of a version of PARKA on the MIMD CM-5. Preliminary results show that we should be able to represent KBs of over 1M frames on a 1K-processor CM-5 and obtain run-time performance nearly an order of magnitude better than the results in this paper. Because the CM-5 is a MIMD machine (though we use it as a SPMD machine), we can use several inferencing techniques that aren't possible on the SIMD CM-2. In particular, we plan to use an active messaging scheme (in the sense of [25]) to increase the flexibility of PARKA's memory association schemes, and to increase the use of pipelining in inferencing.

As for related work in this field, Fahlman was one of the first to propose a massively parallel implementation of a frame description language (FDL) with NETL [8]. NETL, though, was never implemented.

In the past several years, a few parallel KR systems have been implemented. Shastri has proposed a massively parallel, connectionist FDL implementation [18]. A recent implementation of a similar system is reported in [19].

Kolodner describes PARADYME [13] as a "case memory", implemented as a frame system on the CM. PARADYME offers simple top-down inheritance, multi-valued and facetted slots. PARADYME was designed expressly for representing "events" for use by a case-based reasoner. As such it is not a general-purpose frame-system, though it might be adapted into such. In PARADYME the representation of frames is distributed across many processors: two processors are used for each (frame, slot, value) triplet.

In [9], Geller describes the design and implementation of an FDL on the CM-2. The system uses a special mapping of frames to processors to obtain constant-time property inheritance queries on certain classes of IS-A network topologies. Geller's system also allows constant-time "update operations" which maintain constant-time inheritance for future queries by modifying the frame to processor mapping appropriately.

Moldovan's SNAP system [15] uses special purpose hardware to provide extremely fast marker passing operations in a semantic network. SNAP's representation language is relatively crude, but can be used as the basis for implementing more complex representation schemes.

These systems and others are described and critiqued in much greater detail in [4].

5. Conclusion

Using KBs of more than 100,000 frames, we have shown that PARKA computes several types of KR retrievals in time independent of the size of the KB and dependent only on network depth. The run-time of these operations is in the tenths of seconds. This performance compares very favorably to serial representation systems; for structure retrieval operations we recorded speed-ups of more than 10,000 over a highly-optimized serial representation system. Because our empirical evidence so far has supported our analytical claims, we believe that PARKA's performance will scale up to even larger KBs.

We argue that PARKA's performance for these operations has demonstrated that it can offer computationally effective knowledge representation on realistically large KBs. This in turn, demonstrates that massively parallel KR systems can serve as a basis for large AI applications in real-world domains.

6. Acknowledgments

We wish to thank the Systems Research Center at the University of Maryland for their support in this research. We also thank the University of Maryland Institute for Advanced Computer Studies (UMIACS) and Thinking Machines Corp. for the use of their Connection Machines. The support staff at both institutions were very helpful during the development of PARKA.

This work has been supported by AFOSR grant 01-5-28180, ONR grant N0014-88-K-0560, and NSF grant 1R1-8907890.

References

[1] R.J. Brachman. I Lied about the Trees, *AI Magazine*, Vol. 6, (No. 3), Fall 1985.

[2] R.J. Brachman and J.G. Schmolze. An Overview of the KL-ONE Knowledge Representation System, *Cognitive Science*, Vol. 9 (1985).

[3] R. DeMara and H. Kitano. PACE Benchmark for Parallel AI Architectures, Tech. Rep. IMPACT-92-004-USC/CMU, and PKPL 92-5, Department of

EE - Systems, University of Southern California, and School of Computer Science, Carnegie Mellon University, 1992.

[4] M.P. Evett. *PARKA: A System for Massively Parallel Knowledge Representation*, Ph.D. thesis, Dept. of Computer Science, University of Maryland, College Park, 1993 (forthcoming).

[5] M.P. Evett, J.A. Hendler, and L. Spector, Parallel Knowledge Representation on the Connection Machine, *Journal of Parallel and Distributed Computing*, 1993 (forthcoming).

[6] M.P. Evett and J.A. Hendler. Degradation of Interprocessor Communication Operations on the Connection Machine, Tech. Rep., Department of Computer Science, Univ. Maryland, College Park, 1993 (forthcoming).

[7] M.P. Evett and J.A. Hendler. An Update of PARKA, a Massively Parallel Knowledge Representation System, Tech. Rep., CS-TR-2850, Department of Computer Science, Univ. Maryland, College Park, February, 1992.

[8] S.E. Fahlman. NETL: A System for Representing and Using Real World Knowledge, MIT Press, Cambridge, MA, 1979.

[9] J. Geller. Advanced Update Operations in Massively Parallel Knowledge Representation, Tech. Rep. CIS-91-28, Dept. Computer and Information Science, New Jersey Institute of Technology, Newark, NJ, 1991.

[10] K.J. Hammond, Case-based planning: A framework for planning from experience, *Cognitive Science*, Vol. 14 (1990), pp. 384-443.

[11] B. Kettler, J.A. Hendler, W.A. Andersen, and M.P. Evett. Massively Parallel Support for Case-based Planning, *IEEE Expert*, 1993 (forthcoming).

[12] H. Kitano and T. Higuchi, Massively Parallel Memory-Based Parsing, *Proceedings of IJCAI-91*, 1991.

[13] J.L. Kolodner and R. Thau. Design and Implementation of a Case Memory, Tech. Rep. RL88-1, Thinking Machines Corp., Cambridge, MA, Aug. 1988.

[14] D.B. Lenat and R.V. Guha, *Building Large Knowledge-Based Systems*, Addison Wesley, Reading, Mass., 1990.

[15] D. Moldovan, W. Lee, and C. Lin, SNAP: A Marker-Propagation Architecture for Knowledge Processing. Tech. Rep. CENG 89-10, Dept. Electrical Engineering-Systems, University of Southern California, Los Angeles, CA, Sept. 1989.

[16] B. Nebel. Terminological Reasoning is Inherently Intractable, *Artificial Intelligence,* Vol. 43, (No. 2), 1990.

[17] B. Selman and H. Levesque. The Tractability of Path-Based Inheritance, *Proceedings of IJCAI-89*, Morgan-Kaufman, San Mateo, CA, 1989.

[18] L. Shastri, Massive Parallelism in Artificial Intelligence, Tech. Rep. MS-CIS-86-77 (LINC LAB 43), Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 1986.

[19] L. Shastri and V.G. Ajjanagadde. From Simple Associations to Systematic Reasoning: a Connectionist Representation of Rules, Variables and Dynamic Bindings Using Temporal Synchrony. Technical Report MS-CIS-90-05, Department of Computer and Information Science, Univ. of Pennsylvania (Revised January 1992). To appear in *Behavorial and Brain Sciences*.

[20] L. Spector, J.A. Hendler, and M.P. Evett, Knowledge Representation in PARKA. Technical Report 2410, Department of Computer Science, University of Maryland at College Park, 1990.

[21] L. Spector, W. Andersen, J. Hendler, J., Kettler, B., Swartzman, E., Woods, C., and Evett, M., Knowledge Representation in PARKA - Part 2: Experiments, Analysis, and Enhancements. Technical Report 2837, Department of Computer Science, University of Maryland at College Park, 1992.

[22] C. Stanfill, and D. Waltz, Toward Memory-Based Reasoning, *Communications of the ACM*, Vol. 29 (No. 12), December 1986, pp. 1213-1228.

[23] A. Tate. Project Planning Using a Hierarchic Non-linear Planner. Research Report No. 25, Department of Artificial Intelligence, University of Edinburgh, 1976.

[24] D.S. Touretzky. The Mathematics of Inheritance Systems. Morgan Kaufmann, Los Altos, CA, 1986.

[25] T. Von Eicken, D. Culler, S. Goldstein and K. Schauser. Active Messages: a Mechanism for Integrated Communication and Computation, Tech. Rep. UCB/CSD 92/#675, Computer Science Division, EECS, University of California, Berkeley, CA.

[26] R. Wilensky. Some Problems and Proposals for Knowledge Representation. Tech. Rep. UCB/CSD 86/294, University of California, Berkeley, May 1986.