Technical Research Report

Advances in High Performance Knowledge Representation


T.R. 96-67

Sponsored by
the National Science Foundation
Engineering Research Center Program,
the University of Maryland,
Harvard University,
and Industry
Parka on MIMD-supercomputers *

Kilian Stoffel, James Hendler, Joel Saltz and Bill Andersen
Computer Science Dept./UM Institute for Advanced Computer Studies
University of Maryland, College Park

We have ported the SIMD Parka knowledge representation system to generic MIMD machines. The system has been recoded in C and supported using runtime optimization packages developed in the High Performance Systems Software Laboratory at the University of Maryland. New “scanning” algorithms have been developed for inheritance and recognition inferences. These algorithms have been tested with both random networks and on a recoding of the ontology of the CYC knowledge base as well as on large planning case-bases. Tests show that the new version is significantly faster than the SIMD system, and that it promises to scale well to knowledge bases orders of magnitude larger than CYC.

1. Introduction

In the past, research in the field of (very) large knowledge bases has primarily concentrated on defining KR languages and analyzing their complexity. This research has resulted in a wide range of well-studied languages, with much known about their theoretical performance. Less work, however, has been spent examining details of efficiently implementing such languages, or on the scaling properties of these KR systems using large, real-world, applications. Up until recently, this negligence has largely been justified with the excuse that very large KBs don’t yet exist, and that the community could postpone dealing with these issues. However, this excuse is no longer valid. Current research has resulted in the development of extremely large KBs including not only the “common sense” KB of Lenat’s CYC [15], but also (much larger) KBs including machine-readable dictionaries [14], large ontologies [9], and very large case-bases for AI planning systems [13]. Despite this, most of the current KR systems are not able to accommodate these KBs, which may contain millions of assertions about many thousands of objects.

The Parallel Understanding Systems (PLUS) Laboratory at the University of Maryland has been working for a number of years on parallel support for very large knowledge bases. The main system to come out of this laboratory was the Parka system, which was the SIMD implementation of a frame-based AI KR language on the CM-2. The system was shown to be extremely fast, and to perform extremely well on very large knowledge bases

*This research was supported in part by grants from NSF(IRI-9306580), ONR (N00014-91-1451), AFOSR (F49620-93-1-0065), the ARPA/Rome Laboratory Planning Initiative (F30602-93-C-0039, DABT-63-95-C-0037), the ARPA I3 Initiative (N00014-94-10907). Dr. Hendler is also affiliated with the UM Institute for Systems Research (NSF Grant NSF EEC 94-02384).
(having on the order of $10^8$ frames). Details of the Parka system and its implementation, as well as a comparison of Parka to other efforts in VLKBs, can be found in [8,6].

There were two main problems with Parka in its SIMD form. First, there was the obvious problem that SIMD computers are currently less popular than MIMD platforms. Second, the ability to scale to much larger knowledge bases than what could be maintained in the memory of the CM2 (i.e. we needed at least one (virtual) processor per node) was constrained by the hardware. This led us to reimplement Parka on a more conventional MIMD platform, also moving from *Lisp to C.

In this paper, we describe this reimplement effort. In particular, the goal of this project is to reimplement Parka for generic MIMD computers. To reach this goal, we use the CHAOS-Library developed by Dr. Saltz and his students at the High Performance Computing Software Systems Laboratory at the University of Maryland [17]. The language is being implemented in as portable a manner as possible so as to work on a number of different MIMD computers.

We will first overview the UM Parka KR project and present some of the applications for which we are developing very large KBs. We then describe in the next two sections Parka's basic data structures and their current implementation. In Section 5 we will analyze the potential parallelism in the system. The central element of the Parka system is the inference engine which is described in Section 6. We present some performance results in Section 7 and we conclude with a summary of the contribution of this work and future research directions.

2. Parka Overview

The Parka system is a frame-based AI language (sometimes called a “property/class” system in today’s literature) which was designed to be supported efficiently using high-performance computing techniques. The goal of the project is, and has always been, to develop a fairly traditional AI language/tool that can scale to the extremely large size applications mandated by the needs of today’s information technology revolution.

More specifically, Parka allows the user to define a frame-based knowledge base with class, subclass, and property links used to encode the ontology. Property values can themselves be frames, or alternatively can be string, numeric values, or specialized data structures (used primarily in the implementation). The language allows exceptions, in the form of multiple-inheritance, and provides extremely efficient (and efficiently parallelizable) algorithms for performing inheritance using a true inferential-distance-ordering calculation [11]. Parka has also been shown to effectively compute recognition, and also to handle extremely complex “structure matching” queries – a class of conjunctive queries relating a set of variables and constraints and unifying these against the larger KB. While it is difficult to exactly compare KR languages, a very loose categorization would put Parka as more expressive than Classic [3], due to the presence of exception handling.

2*Lisp is the parallel version of Lisp which runs on the Thinking Machine Corporation CM-2 and CM-5 multiprocessors.

3Although it has been shown that IDO with fully general exceptions (i.e. cancellation links) is exponentially hard, we’ve demonstrated that IDO with multiple inheritance exceptions can be computed in polynomial time and is efficiently parallelizable [18].
although slightly less expressive than Loom [16] due to the lack of extensive numerical capabilities. A full description of the language, and more details on past results can be found in http://www.cs.umd.edu/projects/plus/Parka.

Parka is being used in a number of research initiatives at the University of Maryland. Some of the projects using Parka include

- A Knowledge Discovery in Databases application for a large medical knowledge base. In particular, it has been noted that data mining systems can benefit from the use of KBs to provide semantic information for datamining [4]. We are exploiting this sort of information by using medical knowledge in the datamining of an OB-Gyn patient database containing records on over 20,000 patients. The current Parka KB has over 1.2 million assertions [20].

- Several Case-based planning applications including the memory-intensive case-based planning system, Caper, developed by Brian Kettler in his recent doctoral thesis [13]. The Caper KBs, the largest of which contains over 2,000,000 assertions, are described later in this paper (section 4). Parka is also being used to support a logistics planning project, jointly being developed with MITRE Corp. [10], with a KB currently containing over 650,000 assertions.

- A recent project involves the use of Parka as a hybrid knowledge and database for storing and retrieving designs and process plans for mechanical products, as part of on-going effort to develop a new hybrid variant/generative approach to process planning in manufacturing. Parka’s content-based retrieval of designs and plans is important in order to achieve the desired functionality of the process planning system, and will represent a significant advance over current “variant” approaches to process planning, which (among other things) involves retrieving process plans from databases based on fixed-length alphanumeric keys. The KB for this project is currently under construction, but is expected to dwarf those described previously, since current part databases are extremely large, containing information about the machining of thousands of parts.

3. Parka’s basic data structures

The need for portability has influenced the choice of the environments to implement Parka:

Compiler: The compiler used was a standard ANSI C one. This compiler exists on all of the computers we currently access, particularly including the IBM SP-2 and the CRAY T3D.

Communication Library: Our implementation is based on functionality provided by the CHAOS run-time support package which provides scheduling libraries that have been ported to a wide range of MIMD supercomputers. (Work on the Parka project has actually led to a new scheduling approach which is being integrated into the CHAOS libraries[19]).

---

4We report KB sizes in “assertions,” basically the number of links in the semantic network corresponding to the frames, as this accurately reflects the total number of relations between items in the KB and corresponds directly to the “concepts” in a description logic representation.
We have focused on the design and implementation of a very general system that can easily be ported to the different supercomputers. This means that currently, no special optimizations are done using the specific characteristics of any one machine. (Future work will include optimizing on several important architectures.)

3.1. The data structures of the KB
To use CHAOS and other generic tools, it was necessary to reimplement the Parka system to be based on arrays, rather than on the linked lists of the earlier *Lisp implementation. There are three major structures used:

Frames: The primary data structure used in the new implementation is a frame. A frame is an array of a variable size. Each element of the array points to an element of the Network Array described below. This pointer defines the properties of the frame. If an element points to a "normal" frame then the element represents an ISA-link. If the pointer goes to a frame that is characterized to be a property, then the link defines an explicitly valued property of the frame. This semantic difference allows us to separate the links into two sub-arrays, one for ISAs and one for properties.

A property frame (property) has a different structure than a normal frame. The basic structure of a property frame is again an array. This array contains two sub-arrays of the same size. For each non-ISA property in every normal frame, there is a pointer to a property frame. In the property frame, there are two links which are created, each in one of the sub-arrays. The link created in the first sub-array points back to the frame for which the property was defined. The second link points to the value for the property of that frame. In addition, a property frame also contains the arrays of a normal frame (see Figure 1). (This allows properties themselves to be frames with hierarchies and other properties, which was not allowed in the earlier SIMD system.)

Network-Array: The network array is an array which holds all of the normal and property frames in the system as sub-arrays. (This array can be quite large for very large KBs.)

Frame-Table: The frame table is an array of strings. In each element of the frame table the name of a frame is stored. The name corresponds to the name of the frame located at the same position in the Network Array. (e.g. the fifth element of the frame table contains the name of the fifth element of the network). This allows a mapping from "human" to machine descriptors, for use in querying, etc.

This representation of the KB has several considerable advantages.

- The representation of the KB as an array (of arrays) allows us to use standard communication libraries like CHAOS. In addition, computation over array-based data structures is a well-studied area for MIMD Systems.

- The separation of the properties from the ISA links allows a fast parallel implementation for inheritance algorithms.
Figure 1. Frame Arrays in MIMD Parka

- The replacement of the single property link by a frame permits efficient implementations of different parallel algorithms such as recognition and structure matching.

- A separate name space (Frame Table) can be used to calculate starting points for scan operations without the presence of the whole KB.

4. Current Status

We have currently reimplemented all of the major algorithms of the Parka system – the creation of frame-based semantic networks, the performance of inheritance operations thereon, the execution of recognition queries and structure matching queries, all of which we explain describe in the following sections.

4.1. Construction of the Frames, the Network and the Frame Table

There are three methods for creating and/or loading the networks on a parallel machine.

1. The KB is defined directly by a program using commands such as `define_frame`, `define_property`, `define_isa` and `assert`.

2. A KB can be read in from an ASCII file containing these same commands.

3. If a KB is once constructed then the memory can be dumped on the disk. The dump files can be read in again later. It is possible to generate either a dump of the whole network or a distributed dump for multiprocessors. In the latter case, a separate dump for each node is created.

All three methods can be used to generate one network on a single processor or a network being distributed over the nodes of the parallel computer. There are two predefined
methods that distribute the network in blocks or stripes. (It is also possible to use other methods to distribute the network which are predefined or user-defined.)

In the first two methods (where a network is created) a consistency check for the whole network is performed. For the third method it's assumed that the network is correct.

4.2. Inheritance

For the efficient performance of inheritance, we require a fast algorithm for scanning the ISA hierarchy. The efficiency of this algorithm is crucial to supporting very large knowledge bases because, as we argue in [8], most interesting inferences in large knowledge bases are dependent on the efficiency (and correctness) of the inheritance algorithm.

4.2.1. The Scanning Algorithm

The scanning algorithm is dependent on the distribution of the Network Array among the processors. The current implementation supports the two predefined types (block and cyclic) of distributions. The user is free to define other distributions, but all of these distributions must be static.

In principle, there are two approaches as to how the scan algorithm could work:

1. To expand the ISA hierarchy of a frame, a processor has to find out on which node the frame is located. If the frame is on the local node the ISA hierarchy can directly be expanded. If the frame is installed on a remote processor, then the local processor could ask for a copy of the frame from the remote processor and could then expand the hierarchy. This is essentially a "load balancing" solution, as frame information is propagated amongst processors.

2. The local processor has again to locate the frame in question and expand it directly if its local. Otherwise it will inform the remote processor to expand the corresponding frame. This approach (which we use) has more of a scheduling burden (coordinating communication) but needs less explicit load-balancing.

In the second approach, the work load is dependent on the distribution of the Network Array. Each frame is expanded on the processor which hosts that frame. The first approach does not contain such an implicit distribution of the work load. Instead, it would require that an explicit distribution of the work must be defined.

However, the amount of work to realize during the expansion of a frame is very small (only a few instructions) compared with the communication overhead. It would be very hard to implement an efficient load balancer because of the supplementary network load produced by the balancer. For this reason we chose the second approach for our implementation. Each frame is expanded by the processor on which it is stored. This leads to the following basic scan algorithm as used in this implementation:

1. locate the starting frame
2. look up the ISA array of the frame
3. send msgs to all processors which host a frame defined in the ISA array
4. receive all msgs from remote processors
5. for each received frame goto step 2

This program is executed in SPMD mode on each processor of the parallel computer. Based on this scan algorithm it's now possible to define an inheritance algorithm.

4.2.2. Implementation of the Scanning Algorithm

Our programming model attempts to have as much of a data-parallel flavor as possible. This places restrictions on the types of task-parallelism we allow. We list these restrictions below.

1. The task-graphs (network array) are static and acyclic.

2. Each data item (frame) is updated by a unique task and all data-items updated by a given task are on the same processor. This allows an owner-computes strategy.

3. The same task-graph is repetitively used for different data-sets. This allows runtime preprocessing optimizations.

These restrictions are not severe and the previously described implementation of the Parka system satisfies these restrictions.

The nodes for the task-graph are distributed across processors based on a user-specified distribution and the computation is distributed using the owner-computes rule. Since the task-graph computation is iterative, certain optimizations can be performed once in a preprocessing step and reused. One such optimization is to perform a topological sort of the DAG, thus dividing it into levels. As shown in Figure 2a, the synchronization requirements can now be confined to the levels thus increasing the computation granularity. This method has been suggested in [17], and was used to parallelize sparse matrix codes with loop carried dependencies. However global synchronization does not work very well because the synchronization constraints affect the load balancing. All processors have to wait for the slowest processor at each level. The total computation time is thus \( \sum_{i=1}^{\text{levels}} \left( \max_{j=1}^{\text{processors}} \text{Comp}(\text{proc}_j, \text{level}_i) \right)^5 \).

The alternative approach, considered in [5] is to use better distribution mechanisms to increase locality and then use low-latency active messages to communicate the data that does need to go off-processor. The arrival of data automatically triggers computation, thus the synchronization is implicit. The synchronization requirements of this method are very relaxed and thus the load-balance is better. The total computation time is \( \max_{j=1}^{\text{processors}} \left( \sum_{i=1}^{\text{levels}} \text{Comp}(\text{proc}_j, \text{level}_i) \right) \). While the computation time is better than the level-synchronized scheme, the communication/synchronization time may be worse since more messages are sent, even though the amount of data sent is the same. [5] found that high efficiencies could be achieved on the CM-5 using its very low overhead communication support. A drawback is that a data flow programming model must be used.

Our approach is to use the level-based data-parallel model but relax the synchronization constraints by using split-phase synchronization mechanisms. This allows processors to continue processing incoming data from the next level while waiting for the slowest

\(^5\text{Where Comp}(\text{proc}_j, \text{level}_i) \text{ is the time used by processor } j \text{ to compute level } i.\)
DO K = 1, num_levels
  MPI_scatter( level(k-1))
  MPI_gather( level(k))
  COMPUTE( level(k))
END DO

DO K = 1, num_levels
  id = FZY_scatter( level(k-1))
  WHILE ( FZY_recv(id, data)
    == LEVEL.NOT_DONE) )
    COMPUTE( level(k))
END DO

a) Collective synchronization
b) Fuzzy synchronization

Figure 2. Collective/Fuzzy synchronization

processor from the previous level. Since each processor does not progresses to the next level until it has processed all nodes at the current level, the skew is limited to a maximum of one level. This is most useful when a processor can have a high load at one level and a low load at the next level. In such a case, that processor can catch up without slowing the others down.

Figure 2b shows how a DAG could be parallelized using such a fuzzy barrier. Each processor sends out all the outgoing data at the end of each level, but processes each incoming data-chunk as soon it arrives. The condition NOT_DONE remains true until all the incoming messages from a level have arrived, or when a termination condition has been detected. Thus even if processor A is on level X, processor B can begin to process data on level X+1 (until it eventually waits to receive A’s message). Though, we do not discuss it here. the termination detection check can be incorporated to implement branch-and-bound algorithms. The computation time is thus better than that of the level-based scheme with no extra communication overhead.

Figures 3 and 4 shows the results of using fuzzy synchronization and the level-based runtime techniques on the inheritance network. These experiments used an inheritance network with 500,000 classes and instances (i.e the DAG has 500K nodes) and several links between each (for a total of about 1.5M links). As can be seen from 3, fuzzy synchronization provides significant benefits over using collective synchronization. Figure 4 shows that high efficiencies can be obtained over a wide range of platforms using these techniques. The efficiencies are around 70%.

4.2.3. The Inheritance Algorithm

The inheritance algorithm we use is based on the Touretzky IDO algorithm as modified in Parka. The theoretical worst case complexity of this algorithm is \( O(d \times n/k) \) where \( d \) is the depth of the ISA-hierarchy, \( n \) is the number of frames in the network and \( k \) is the number of nodes of the parallel computer.

The idea of our algorithm is that in one pass through the ISA-hierarchy, we can simultaneously create a list of all properties (direct and inherited) for each node and also a list of all properties “overwritten” by more specific properties (therefore being eliminated from the first list) The difference of these two lists contains the properties inherited according to IDO. We discuss the efficiency of this algorithm in section 7.2.2 (Table 1).
5. Potential Parallelism in KB

In this section, we discuss the kind and degree of MIMD-parallelism we exploit. The parallelism of a KB is hidden in the ontology. In our case the ontology is represented as a directed acyclic graph. In such a graph the parallelism usable by the scanning algorithm is defined by the branch out of each frame in the network (i.e. the number of ISA links per frame).

A network in which each frame has only one ISA link defined has no parallelism exploitable by the scanning algorithm. Each frame has to be completed before the descendant frame can be attacked. Such a graph imposes serial execution of the scanning algorithm. On the other hand, a graph with depth two and a large amount of ISAs defined for the root frame has the highest possible degree of parallelism for the scanning algorithm.

This algorithm therefore seems to be well adapted for AI KBs. Generally, these are not very deep but often very bushy. The fact that the parallelism is low for narrow ISA hierarchies is not very harmful. In addition, in several types of queries there are multiple scans to perform. These scans can be combined and executed as one single larger scan. Such a method is used to implement recognition queries (see Section 6.1) and in more complex inference methods like the structure matcher (see section 6.2).

6. Complex Inferencing in MIMD Parka

As mentioned previously, we have built several other inference classes on top of the inheritance mechanism. In this section, we discuss the two most important: recognition queries and structure matching inferences.
Figure 4. Performance of parallelized Inheritance Network code: Timings shown include the native communication systems for the SP2, the Paragon and the CM5 use the native communication system as well as MPI and PVM times for the SP2 and T3D machines respectively.

6.1. Recognition

We have implemented a new version of Parka's recognition algorithm – its means for handling conjunctive property queries. The algorithm is based on the special structure of the property frames and again uses the scanning algorithm. Essentially, we handle these queries as a set of inheritance queries and some join operations, thus we can maintain the complexity of \(O(d \times n/k)\).

To process the conjunctive property query, we split the conjunction into the conjunctive elements. This produces a list of simple expressions of the form \((prop1, x, y)\) (where \(x\) and \(y\) can both be variables, or one of the two can be a constant). So we have to find all the frames which have an explicit or an inherited definition for the given property and their value in respect to the property.

It is easy to find all explicitly labeled frames for a given property. The sub-array of the property frame with the back pointers contains all the pointers to the frames which are directly labeled for this property. The other sub-array of the property frame holds the pointers to the value for the property. On one hand, it’s possible to create a list with all the frames for which the property is directly defined and which have the searched value. On the other hand, a list with all the frames which do not have the value being searched for, but which are able to be “overwritten” with the correct values during the inheritance can also be created.

The basic idea of the recognition algorithm is that all the frames which have the required property value are activated and start to send information downward along the ISA links.

\(6\) The SIMD algorithm for performing this was one of the main contributions of the original Parka system[7].
During this scan, all frames in the ISA hierarchy will inherit the values of the property. If the inherited information arrives on a node which is directly labeled for a property, but with a different value, then all the descendants of this node are commanded to remove the value they inherited from the same node as this node. Thus, it's possible to correctly handle all inherited properties according to IDO in one single scan.

We can find the solution to each sub-query defined by the elements of the original conjunctive property query in one single IDO run (scan run). The results of this run is a number of sets containing the frames and their value for the analyzed property. Frames which are elements of all the sets and their corresponding values are the solutions of the original conjunctive query.\(^7\)

The potential degree of parallelism of this algorithm is high, because there are multiple starting points (all the frames with direct correct property values). At the same time it's possible to send multiple properties across the ISA hierarchy during one scan step which results in a large scan graph with a high degree of parallelism.

6.2. Structure Matcher

The second even more complex inference method is the structure matching algorithm. It allows general relation-based structures to be retrieved from a knowledge base. The algorithm takes a conjunctive query similar in form to the recognition query, but with binary constraints (i.e., with two variables). A set of these conjuncts defines a graph structure, where variables or constants are the nodes and predicates are the arcs. The algorithm compares the probe structure with the knowledge base and returns a set of all satisfying matches.

Our description of the problem of structure matching follows that given in [21]. A knowledge base defines a set, \( P \), of unary and binary predicates. Unary predicates have the form \( P_i(x) \) and binary predicates have the form \( P_j(x_1, x_2) \), where each \( x_i \) is a variable on the set of frames in the KB. An existential conjunctive expression on these predicates is a formula of the form \( \exists x_1, \ldots, x_m : P_1 \land P_2 \land \ldots \land P_n \), where \( n \geq 1 \). Our task is to retrieve all structures from memory which match a given conjunctive expression. Therefore, we would like to find all such satisfying assignments for the \( x_i \).

We can view the problem of matching knowledge structures in two ways. The first is as a subgraph isomorphism problem\(^8\). We view variables as nodes and binary predicates as edges in a graph. We want to find structures in memory which "line up" with the graph structure of the query expression. The other way to view the matching problem is as a problem of unification or constraint satisfaction. If we can find a structure in memory which provides a consistent assignment to the variables \( x_i \) (i.e., unification), then that structure matches the conjunctive expression.

6.3. Overview of the algorithm

The structure matching algorithm operates by comparing a retrieval probe, \( P \), against a knowledge base (KB) to find all structures in the KB which are consistent with \( P \). This match process occurs in parallel across the entire knowledge base. A Parka KB

---

\(^7\)This can be formalized as a set of joins in the relational calculus.

\(^8\)More specifically, this is a problem of subgraph isomorphism with typed edges, the edges being the relations in the KB between frames.
consists of a set of frames and a set of relations (defined by predicates) on those frames. Most relations are only implicitly specified and so must be made explicit by expanding the relation with the appropriate inference method. By computing inherited values for a relation, all pairs defining the relation are made explicit. We currently allow only unary and binary relations⁹.

A retrieval probe is specified as a graph consisting of a set of variables $V(P)$ and a set of predicates (or constraints) $C(P)$ that must simultaneously hold on frames bound to those variables. The result of the algorithm is a set of $k$-tuples, where each $k$-tuple encodes a unique $1 - 1$ mapping of frames to variables in $V(P)$, that unifies with the description of the structure in memory with $C(P)$. Figure 5 shows a simple example of the structure matching algorithm; given a knowledge base represented as a semantic network and a probe graph, the algorithm finds all matching subgraphs in the network which are isomorphic to the probe.

![Fragment of semantic network](image)

**Fragment of semantic network**

![Probe graph](image)

**Probe graph**

![Structure Matcher](image)

**Structure Matcher**

![Matching subgraphs](image)

**Matching subgraphs**

Figure 5. A simple example of structure matching.

The set of frames that can bind to each variable is initially restricted by a set of constraints indicated by unary predicates. Each unary constraint may only constrain the

⁹Future versions of Parka may support higher-order predicates.
values of one variable. Examples of these constraints are "X is a dog" or "the color of X is yellow". We allow set theoretic combinations of the unary constraints, for example "X is a dog and the color of X is yellow", or "X is a dog but X is not yellow." The domains for each variable are maintained throughout the match process and are further restricted as more constraints are processed.

Constraints between frames bound to variables are specified by a set of binary constraints. For example, we can say "the color of X must be Y", or "X is a part of Y", for some X and Y in V(P). Binary constraints are processed by "expanding" the binary relation given in the constraint. By expansion we mean that all pairs participating in a relation R in the KB are made explicit by invoking the inference method for the associated predicate. The pairs allowed to participate in the expanded relation are restricted to those in the domains of the variables related by R. For example, a binary constraint may be expressed as (Color X Y). In this case the values for each concept in the domain of X are computed for the color predicate and pairs that have values outside the domain of Y are excluded.

Two additional binary predicates, "eq" and "neq" are provided to provide co-designation and non-co-designation of variables. These constraints act as a filter, eliminating any tuples from the result for which the constrained variables are(not) bound to the same frame. (Other binary predicates could be supported as well, such as inequality and other numeric constraints. However, in the current implementation, we do not provide support for these constraints.)

The result of a structure match is a set of k-tuples, where each tuple corresponds to a satisfying assignment of the k variables. Alternatively, the result can be viewed as a relation. Initially, the matcher begins with an empty set of relations. During the match, several intermediate relations may be constructed. Simple binary relations result from the expansion of a binary constraint. These are later fused (via a relational join operation) or filtered (via co-designation or non-co-designation) until a single relation remains. The algorithm selects binary constraints to process using a greedy algorithm based on a simple cost model stored in the meta-data.

7. Testing

To realize the tests it was necessary first to program a network generator. The networks generated by this program were afterwards used to test the implemented KB.

7.1. Network generator
The network generator is able to generate two kinds of KB formats:

1. ASCII file
2. memory dump files

There are three characteristics of the network that can be defined:

1. the branching factor in the network
2. the number of levels
3. the number of levels with the maximal size

Further it is possible to randomly generate properties for different frames. There is a parameter to define the density of these properties. This allows us to duplicate some of the tests made on the original Parka system on the SIMD CM-2 computer (cf. [8]).

7.2. Results on IBM SP2 (16 processors)
7.2.1. Degree of Parallelism

As mentioned above, the performance of the scan algorithm depends on the structure of the ISA hierarchy. To test this, we created two networks, both with 2,500 frames. The first network is 2,500 levels deep and each level consists of one frame. The second network has two levels, a first where one frame is inserted and and a second with 2,499 frames. For the sequential algorithm the amount of work is the same for both networks. This is also true for the parallel scanning algorithm but the big difference is that for the first network there is little opportunity for parallelism – one level has to be executed after another. In the second network it’s different. All the frames on the second level can be treated in parallel if they are distributed among the processors.

To execute this first test, we used 4 processors on an IBM SP2. A cyclic network distribution was used. Thus, processor 1 owned the frames \{0, 4, 8, \ldots\}, processor 2 owned the frames \{1, 5, 9, \ldots\}, etc.

The execution of the scan algorithm on the first network took 0.4 sec. The scan through the second network only 0.008 sec. Note that although the theoretical speedup should be no more than 4, we have a speedup of 50! This is related to the communication system of the SP2 (and most other MIMD machines). For the first network we have to communicate 2499 short messages, in the second one long message. The throughput of the communication system is much higher on a small number of long messages then on a large number of short messages.

This result seems to be very promising for real KB, because most of the existing KBs have a very flat ontology. For example, in a recoding of the ontology of CYC for Parka (see section 7.2.3), the deepest subgraph we found was only 27 levels deep, whereas maximum branch out was over 1,000.

7.2.2. Scaling and Speedup

As a second test, we want to analyze how the scanning algorithm performs if the configuration of the computer changes. The results presented here are again based on generated networks. The branching factor was fixed to 3. The depth of the ISA-hierarchy was varied from 9 to 12, generating networks of about 29,500, 88,500, 265,700, and 797,000 frames respectively. (As a point of comparison, our encoding of the ontology of CYC (Section 7.2.3) has about 32,500 frames.) A property to find was placed on the root node and the inheritance was done by one of the leaf nodes (this corresponds to the worst case for the previous Parka implementation). The absolute values represented in Figure 6 and Table 1 are extremely fast, however we note that they are not so important because the programs were not optimized for the SP2 where the tests were done. What’s interesting is the comportment of the algorithms. There are two points that seem to be interesting:

1. the scalability in terms of processor number
<table>
<thead>
<tr>
<th>nbr. nodes/netsize</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>29524</td>
<td>0.115</td>
<td>0.060</td>
<td>0.033</td>
<td>0.021</td>
<td>0.017</td>
</tr>
<tr>
<td>88573</td>
<td>0.345</td>
<td>0.175</td>
<td>0.091</td>
<td>0.050</td>
<td>0.032</td>
</tr>
<tr>
<td>265720</td>
<td>1.037</td>
<td>0.523</td>
<td>0.265</td>
<td>0.137</td>
<td>0.088</td>
</tr>
<tr>
<td>797161</td>
<td>1.689</td>
<td>0.735</td>
<td>0.388</td>
<td>0.201</td>
<td></td>
</tr>
</tbody>
</table>

Table 1
Inheritance

Figure 6. Scanning Performance
2. the scalability in terms of network size

On both measures, we see that the algorithms perform quite well. In addition, we believe further speedup would be available if we optimized the algorithms for this particular platform.

7.2.3. Results for the CYC Ontology

We have performed a number of tests on KBs that were not randomly created. The largest one we had used previously in testing the SIMD version of the system was the CYC ontology. As previously reported in [7], we encoded the ontology of version 8 of MCC's CYC system into Parka. In our version, this ontology has about 32,000 frames and about 150,000 assertions relating them (property and ISA links). The biggest difference between the generated networks and CYC is that the test network had only a single property on the root, thus making all "subgraphs" equal to the entire KB. In CYC the ISA hierarchies under particular properties are only subnetworks of the full 32,000 frames.

We hoped to directly compare the SIMD and MIMD versions using queries on the CYC system, but most of the tests we'd run previously were, basically, too easy for the new version. In particular, the subnetworks in CYC are all much smaller than the smallest test network described in the previous section. For this reason simple queries like "What's the color of frame X" could not be used to show parallel effects – their time was on the order of 50-100μseconds on a single processor! Thus, instead of timing single inheritance queries, we generated a new class of recognition-like queries that were designed to stress the new system.

To start with, we tried the timing of recognition queries in some of CYC's largest subtrees. The results were quite promising, but again too fast for showing parallel speedups. For example, using the scan algorithms, we executed some recognitions with more than twenty conjuncts and had single processor response times of under 1/100 of a second (compared with about 1 second for the SIMD system).

To further stress the scanning algorithms and to explore speedups, we designed a new set of inferences specifically to that purpose. In particular, we used queries that would include great amounts of both inheritance and recognition. These queries were of the following form: "Give me all the frames which have one or more properties in common with frame X". To be sure to get "slow" times, we ran these queries on frames in big ISA sub-hierarchies of CYC. Two of the biggest, plant and animal, were chosen for testing, since they had very large numbers (in the tens of thousands) of other frames which shared at least one property.

The execution time for the queries "Give me all the frames with the same properties as Animal" and "Give me all the frames with the same property as Plant" are presented in Figure 7, which compares these times to the optimal speedup curve.

As one can see the recognition algorithm behaves very well. The efficiency is about 75%. That's very high with respect to the small amount of CPU time needed for the inheritance algorithm. The queries represented in the Figure 7 are much more complex than standard queries in CYC.
Figure 7. Recognition-Performance in CYC

7.3. Results for UM-Translog

A second domain used for testing the MIMD implementation was on knowledge bases that were created as part of doctoral thesis work on the development of a system for memory intensive case-based planning [12]. Part of this project involved the automatic seeding of large case memories by a generative planner. One domain used in this work was the “UM Translog” domain, a logistics planning domain developed for the evaluation and comparison of AI planning systems. Case-bases of various size were created, with each containing a number of plans, derivation information, and planning related ontologies.\(^\text{10}\)

To measure the performance of the structure matcher we used the UM-Translog KB in different sizes (10 cases, 100 cases and 200 cases).

The results presented in this section are especially interesting for two reasons. On one hand we are presenting the timings for the structure matcher, the most complex retrieval algorithm implemented in Parka to date. On the other hand we are also able to show, that the new Parka system has the capability to handle very large KBs on a single Sparc workstation.

The queries we used to test the system are presented in detail in Appendix A. These are all queries used by the case-based planning system during its problem solving. They were chosen at random from a large number of stored queries. The results are summarized in Table 2. Table 2a presents the timings for the six queries on a SPARC 20 using the three different KBs, and Table 2b presents the timings on 1, 8 and 16 nodes of an SP2 using the 200 case KB (the largest of the three). The actual sizes of these KBs are shown

\(^{10}\)A full description of the domain, the complete specification of the planning operators used, and the case-bases themselves are available on the World-Wide Web at http://www.cs.umd.edu/projects/plus/UMT/.
Table 2
UM-Translog Timings[msec] (serial and parallel).

<table>
<thead>
<tr>
<th>Query</th>
<th>20 CB</th>
<th>100 CB</th>
<th>200 CB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1020</td>
<td>4740</td>
<td>6990</td>
</tr>
<tr>
<td>2</td>
<td>195</td>
<td>1305</td>
<td>1635</td>
</tr>
<tr>
<td>3</td>
<td>225</td>
<td>1470</td>
<td>1725</td>
</tr>
<tr>
<td>4</td>
<td>630</td>
<td>3570</td>
<td>4590</td>
</tr>
<tr>
<td>5</td>
<td>675</td>
<td>3600</td>
<td>4605</td>
</tr>
<tr>
<td>6</td>
<td>405</td>
<td>585</td>
<td>645</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query</th>
<th>1</th>
<th>8</th>
<th>16</th>
<th>1:8</th>
<th>1:16</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3041</td>
<td>546</td>
<td>313</td>
<td>69.6</td>
<td>60.7</td>
</tr>
<tr>
<td>2</td>
<td>713</td>
<td>129</td>
<td>75</td>
<td>69.1</td>
<td>59.4</td>
</tr>
<tr>
<td>3</td>
<td>753</td>
<td>135</td>
<td>79</td>
<td>70.0</td>
<td>56.6</td>
</tr>
<tr>
<td>4</td>
<td>1997</td>
<td>361</td>
<td>205</td>
<td>69.1</td>
<td>60.9</td>
</tr>
<tr>
<td>5</td>
<td>2003</td>
<td>360</td>
<td>206</td>
<td>69.5</td>
<td>60.8</td>
</tr>
<tr>
<td>6</td>
<td>284</td>
<td>52</td>
<td>29</td>
<td>68.2</td>
<td>61.2</td>
</tr>
</tbody>
</table>

Table 3
Sizes of the UM-Translog KBs.

<table>
<thead>
<tr>
<th>cases</th>
<th>frames</th>
<th>structural links</th>
<th>property links</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>11612</td>
<td>26412</td>
<td>114359</td>
</tr>
<tr>
<td>100</td>
<td>59915</td>
<td>130558</td>
<td>800481</td>
</tr>
<tr>
<td>200</td>
<td>123173</td>
<td>266176</td>
<td>1620456</td>
</tr>
</tbody>
</table>

in Table 3, where frames is the number of nodes in the DAG, structural links are those in the ISA ontology, and property links are all others (i.e. the number of the edges in the DAG is equal to the structural links plus the property links). (We believe the 200 case case-base to be the largest meaningful semantic net created to date, as can be seen it contains about 1.8M assertions concerning over 123,000 entities.)

As can be seen, the sequential timings range from under a second for the simplest query to about 7 seconds for the most complex. On the parallel system all queries were executed in under one second, with the simplest query taking only 29 milliseconds on 16 processors, and the most complex taking only 303 millisecond. Table 2b also shows the efficiencies of the parallel algorithm. Even in the current non-optimized form, the efficiency averages about 69.3% for eight processors and 59.9% for 16 processors.

8. Future work

We are currently working on pushing the current implementation and developing new algorithms and applications. Algorithmically, current work is focused on doing empirical research on how these algorithms scale to much larger networks and greater numbers of processors. In particular, the current implementation is done on a small IBM SP2 machine (16 "wide" nodes). The next step will be to test these programs on bigger SP2s to see how the scaling continues in larger networks and on larger machines. We also will do work on optimizing communication patterns for the SP2 implementation (which should
significantly improve absolute performance) as well as porting this implementation to the Cray T3D machine, which has significantly faster communication times. We expect to be able to run extremely large networks (potentially billions of frames) extremely efficiently on this machine. First tests of the basic scan algorithm seem to support this assumption.

In addition, we are exploring the use of I/O to allow the use of secondary storage for processing extremely large knowledge bases where even the online memory of the large parallel machines is not adequate. The CHAOS runtime support package is being extended to support efficient I/O, and we are also beginning an examination of the special I/O requirements for the storage and caching needs of much larger knowledge bases (containing 100s of millions of assertions). The SP2 at Maryland is being configured to contain about 200 Gigabytes of disk storage, making it uniquely suited for exploring massive knowledge and database applications.

In regard to applications, we are currently exploring the use of knowledge based techniques in the guidance of data-mining applications. In data mining applications, a knowledge repository is searched for information with the aim of obtaining as much information as possible. One weakness with current data-mining systems, however, is their lack of access to semantic knowledge about the domain being searched. Storing such semantics, however, requires much larger knowledge bases that function significantly faster than most of today's systems. Such systems will be needed for doing significant knowledge discovery against large, heterogeneous data corpora such as medical data systems or the World Wide Web. Parka promises to provide support for such knowledge bases. We are currently exploring the development of such hybrid knowledge and databases in the areas of medical informatics, military transportation logistics, and health care financing.

9. Conclusion

In summary, we have ported the SIMD Parka system to more generic MIMD machines. The system has been recoded in C and supported using runtime optimization packages developed in the high performance computing laboratory at Maryland. New “scanning” algorithms have been developed for inheritance, recognition, and structure matching inferences. These algorithms have been tested with both random networks and on two very large knowledge bases (CYC and the UM-translog case bases). Tests show that the new version is significantly faster than the SIMD system, and that it promises to scale well to knowledge bases of the order of magnitude needed for complex information technology applications.

REFERENCES


A. UM-Translog Queries

;;; QUERY 1: "find all plans for getting a package from a LOC to a LOC in DIFF city connected by a direct route"
(query! ':and
  (#!instanceOf     ?plan  #!E-Plan)
  (#!instanceOf     ?goal  #!CE-Located-At)
  (#!everyInstanceOf ?pkg   #!Package)
  (#!everyInstanceOf ?loc   #!Place)
  (#!everyInstanceOf ?dloc  #!Place)
  (#!instanceOf     ?ocity  #!City)
  (#!instanceOf     ?dcity #!City)
  (#!instanceOf     ?sit   #!Initial-Situation)
  (#!instanceOf     ?sval  #!Situated-Value)
  (#!instanceOf     ?pkg-pvar #!Plan-Variable)
  (#!instanceOf     ?dest-pvar #!Plan-Variable)
  (#!everyInstanceOf ?route  #!Route)
  (#!plan-Goal=     ?plan  ?goal)
  (#!thing=         ?goal  ?pkg-pvar)
  (#!var-Value=     ?pkg-pvar  ?pkg)
  (#!location=      ?goal  ?dest-pvar)
  (#!var-Value=     ?dest-pvar  ?dloc)
  (#!situation=     ?plan  ?sit)
  (#!location-sit=  ?pkg  ?sval)
  (#!situation=     ?sval  ?sit)
  (#!sslot-Val=     ?sval  ?loc)
  (#!in-City        ?loc   ?ocity)
  (#!in-City        ?dloc  ?dcity)
  (#!neq            ?ocity  ?dcity)
  (#!route-Origin=  ?route  ?ocity)
  (#!route-Destin=  ?route  ?dcity)
))

;;; QUERY 2: "find all plans using a particular train station, Hub-Ts"
(query! ':and
  (#!instanceOf     ?act   #!P-Move-Vehicle)
  (#!instanceOf     ?plan  #!E-Plan)
  (#!everyInstanceOf ?train  #!Train)
  (#!instanceOf     ?p-varp #!Plan-Variable)
  (#!instanceOf     ?pvara  #!Plan-Variable)
  (#!vehicle=       ?act  ?p-varp)
  (#!var-Value=     ?p-varp  ?train)
  (#!destin=        ?act  ?pvara)
  (#!var-Value=     ?pvara  #!Hub-Ts)
  (#!action-Of-Plan ?act  ?plan)
))

;;; QUERY 3: "find all top-level plans using a Regular truck (uses IDO and transitive inheritance)"
(query! ':and
  (#!everyInstanceOf ?truck1 #!Regular-Truck)
  (#!instanceOf     ?act   #!P-Move-Vehicle)
  (#!instanceOf     ?plan  #!E-Plan)
  (#!instanceOf     ?plant #!E-Plan)
(\#instanceOf ?case \#Case)
(\#instanceOf ?pvar \#Plan-Variable)
(\#\var-Value= ?pvar ?truck1)
(\#vehicle= ?act ?pvar)
(\#action-Of-Plan ?act ?plan)
(\#tmember-Of ?plan ?plant)
(\#plan-of-Case ?plant ?case)
)

;;; QUERY 4: "find all plans for getting an object from a LOC to a LOC in SAME city"
(query! (,:and
(\#instanceOf ?plan \#E-Plan)
(\#instanceOf ?goal \#CE-Located-At)
(\#everyInstanceOf ?pkg \#Package)
(\#everyInstanceOf ?loc \#Place)
(\#everyInstanceOf ?doctype \#Place)
(\#instanceOf ?city \#City)
(\#instanceOf ?sit \#Initial-Situation)
(\#instanceOf ?sval \#Situated-Value)
(\#instanceOf ?pkg-pvar \#Plan-Variable)
(\#instanceOf ?dest-pvar \#Plan-Variable)
(\#plan-Goal= ?plan ?goal)
(\#thing= ?goal ?pkg-pvar)
(\#\var-Value= ?pkg-pvar ?pkg)
(\#location= ?goal ?dest-pvar)
(\#\var-Value= ?dest-pvar ?doctype)
(\#situation-\before= ?plan ?sit)
(\#location-sit= ?pkg ?sval)
(\#situation= ?sval ?sit)
(\#slot-\value= ?sval ?loc)
(\#in-City ?loc ?city)
(\#in-City ?doctype ?city)
))

;;; QUERY 5: "find all plans for getting an object from a LOC to a LOC in DIFF city"
(query! (,:and
(\#instanceOf ?plan \#E-Plan)
(\#instanceOf ?goal \#CE-Located-At)
(\#everyInstanceOf ?pkg \#Package)
(\#everyInstanceOf ?loc \#Place)
(\#everyInstanceOf ?doctype \#Place)
(\#instanceOf ?cit \#City)
(\#instanceOf ?dcity \#City)
(\#instanceOf ?sit \#Initial-Situation)
(\#instanceOf ?sval \#Situated-Value)
(\#instanceOf ?pkg-pvar \#Plan-Variable)
(\#instanceOf ?dest-pvar \#Plan-Variable)
(\#plan-Goal= ?plan ?goal)
(\#thing= ?goal ?pkg-pvar)
(\#\var-Value= ?pkg-pvar ?pkg)
(\#location= ?goal ?dest-pvar)
(\#\var-Value= ?dest-pvar ?doctype)
)
(\texttt{\textbackslash !situation-\textbackslash !Before} = \texttt{\textbackslash ?plan \textbackslash ?sit})
(\texttt{\textbackslash !location-\textbackslash !sit} = \texttt{\textbackslash ?pkg \textbackslash ?sval})
(\texttt{\textbackslash !situation} = \texttt{\textbackslash ?sval \textbackslash ?sit})
(\texttt{\textbackslash !sslot-\textbackslash !Val} = \texttt{\textbackslash ?sval \textbackslash ?oloc})
(\texttt{\textbackslash !in-City} = \texttt{\textbackslash ?oloc \textbackslash ?ocity})
(\texttt{\textbackslash !in-City} = \texttt{\textbackslash ?dloc \textbackslash ?dcity})
(\texttt{\textbackslash !neq} = \texttt{\textbackslash ?ocity \textbackslash \textbackslash ?dcity})
)

;;;; QUERY 6: "find compatible packages and vehicles"
(query! (\texttt{\textbackslash :and})
  (\texttt{\textbackslash !everyInstanceOf \textbackslash ?pkg \textbackslash !Package})
  (\texttt{\textbackslash !ancestor} \texttt{\textbackslash ?ptype \textbackslash !Package})
  (\texttt{\textbackslash !ancestor} \texttt{\textbackslash ?vtype \textbackslash !Vehicle})
  (\texttt{\textbackslash !everyInstanceOf \textbackslash ?veh \textbackslash !Vehicle})
  (\texttt{\textbackslash !instanceOf \textbackslash ?pkg \textbackslash ?ptype})
  (\texttt{\textbackslash !instanceOf \textbackslash ?veh \textbackslash ?vtype})
  (\texttt{\textbackslash !pv-Compatible} = \texttt{\textbackslash ?ptype \textbackslash ?vtype})
))
Parka-DB: Integrating knowledge- and data-based technologies *

Kilian Stoffel, Merwyn Taylor, and James Hendler

Department of Computer Science
University of Maryland
College Park, Maryland 20742
stoffel,mtaylor,hendler@cs.umd.edu

Keywords: Implemented KR&R Systems: Reports, Evaluations, Updates, and (Parallel and Distributed Implementations)

Abstract

Real world applications are demanding that KR systems provide support for knowledge bases containing millions of assertions. We present Parka-DB, a high-performance reimplementation of the Parka KR language which uses a standard relational DBMS. The integration of a DBMS and the Parka KR language allows us to efficiently support complex queries on extremely large KBs using a single processor, as opposed to our earlier massively parallel system. In addition, the system can make good use of secondary memory, with the whole system needing less than 16MB of RAM to hold a KB of over 2,000,000 assertions. We demonstrate empirically that this reduction in primary storage requires only about 10% overhead in time, and decreases the load time of very large KBs by more than two orders of magnitude.

*This research was supported in part by grants from NSF(IRI-9306580), ONR (N00014-J-91-1451), AFOSR (F49620-93-1-0065), the ARPA/Rome Laboratory Planning Initiative (F30602-93-C-0039, DABT-63-95-C-0037), the ARPA I3 Initiative (N00014-94-10907). Dr. Hendler is also affiliated with the UM Institute for Systems Research (NSF Grant NSF EEC 94-02384).
1 Introduction

In the past, research in the field of (very) large knowledge bases has primarily concentrated on defining KR languages and analyzing their complexity. This research has resulted in a wide range of well-studied languages, with much known about their theoretical performance. Less work, however, has been spent examining details of efficiently implementing such languages, or on the scaling properties of these KR systems using large, real-world, applications. Up until recently, this negligence has largely been justified with the excuse that very large KBs don't yet exist, and that the community could postpone dealing with these issues. However, this excuse is no longer valid. Current research has resulted in the development of extremely large KBs including not only the "common sense" KB of Lenat's CYC [14], but also (much larger) KBs including machine-readable dictionaries [13], large ontologies [7], and very large case-bases for AI planning systems [12]. Despite this, most of the current KR systems are not able to accommodate these KBs, which may contain millions of assertions about many thousands of objects.

In this paper, we describe a new implementation of the Parka knowledge representation system which is based on a relational database system. The new implementation, Parka-DB, is therefore a KR system tightly coupled with a DBMS system. This new architecture allows us not only access to bigger KBs but at the same time improves the performance of Parka's inference engine in terms of memory utilization. In this way it allows the querying of KBs of virtually arbitrary size, while still remaining extremely efficient.

In this paper, we first overview the UM Parka KR project and present some of the applications for which we are developing very large KBs. We then describe the architecture of the hybrid knowledge and database implementation (Section 3). A series of empirical test results and performance analysis is then presented in section 4. Section 5 lists some important related projects, and we conclude with a summary of the contribution of this work and future research directions.

2 Parka Overview

The Parka system is a frame-based AI language (sometimes called a "property/class" system in today's literature) which was designed to be supported
efficiently using high-performance computing techniques. The goal of the project is, and has always been, to develop a fairly traditional AI language/tool that can scale to the extremely large size applications mandated by the needs of today's information technology revolution.

More specifically, Parka allows the user to define a frame-based knowledge base with class, subclass, and property links used to encode the ontology. Property values can themselves be frames, or alternatively can be string, numeric values, or specialized data structures (used primarily in the implementation). The language allows exceptions, in the form of multiple-inheritance, and provides extremely efficient (and efficiently parallelizable) algorithms for performing inheritance using a true inferential-distance-ordering calculation [10].\footnote{Although it has been shown that IDO with fully general exceptions (i.e. cancellation links) is exponentially hard, we've demonstrated that IDO with multiple inheritance exceptions can be computed in polynomial time and is efficiently parallelizable [18].} Parka has also been shown to effectively compute recognition, and also to handle extremely complex "structure matching" queries – a class of conjunctive queries relating a set of variables and constraints and unifying these against the larger KB. While it is difficult to exactly compare KR languages, a very loose categorization would put Parka as more expressive than Classic [4], due to the presence of exception handling, although slightly less expressive than Loom [15] due to the lack of extensive numerical capabilities. A full description of the language, and more details on past results can be found in http://www.cs.umd.edu/projects/plus/Parka.

One of the key features of Parka is that it has been shown to efficiently handle its inferencing on KBs containing millions of assertions. Early work on the system gained most of its efficiency through massive parallelism [6], however in recent years we've made increasing use of database management techniques to remove the need for parallelism (although still allowing for efficient parallelization). The version we describe in this paper uses DBMS technologies to support inferencing and data management. In particular, we will describe Parka-DB which was developed to run on generic, single processor (or parallel) systems with significantly less primary memory requirements than the previous versions.

Parka-DB is being used in a number of research initiatives at the University of Maryland. Some of the projects using Parka-DB include:

- A KDD [20] application for a large medical knowledge base. In particu-
lar, it has been noted that data mining systems can benefit from the use of KBs to provide semantic information for datamining [5]. We are exploiting this sort of information by using medical knowledge in the datamining of an OB-Gyn patient database containing records on over 20,000 patients. The current Parka-DB KB has over 1.2 million assertions.²

- Several Case-based planning applications including the memory-intensive case-based planning system, Caper, developed by Brian Kettler in his recent doctoral thesis [12]. The Caper KBs, the largest of which contains over 2,000,000 assertions, are described later in this paper (section 4). Parka is also being used to support a logistics planning project, jointly being developed with MITRE Corp. [9], with a KB currently containing over 250,000 assertions.

- A recent project involves the use of Parka as a hybrid knowledge and database for storing and retrieving designs and process plans for mechanical products, as part of ongoing effort to develop a new hybrid variant/generative approach to process planning in manufacturing. Parka's content-based retrieval of designs and plans is important in order to achieve the desired functionality of the process planning system, and will represent a significant advance over current "variant" approaches to process planning, which (among other things) involves retrieving process plans from databases based on fixed-length alphanumeric keys. The KB for this project is currently under construction, but is expected to dwarf those described previously, since current part databases are extremely large, containing information about the machining of thousands of parts.

3 Parka-DB

To support the demands of the extremely large KBs which are needed in applications such as those above, without requiring (but still allowing) the use of supercomputers, we have been using an increasing number of database algorithms to perform the inferencing in Parka. This led to a reimplementation of the system, which we call Parka-DB, in which the Parka KR sys-

²We report KB sizes in "assertions," basically the number of links in the semantic network corresponding to the frames, as this accurately reflects the total number of relations between items in the KB and corresponds directly to the "concepts" in a description logic representation.
stem is implemented directly using a relational database management system (RDBMS). The system thus blends the knowledge-based inferencing capabilities of Parka with the standard database capabilities of the RDBMS. In particular, Parka-DB uses the RDBMS as a runtime storage medium. Thus, since the RDBMS uses external devices to store data, Parka-DB can manage knowledge bases that are too large to be maintained in internal memory. In fact, Parka-DB can handle KBs that are as large as available disk space, and can make use of the RDBMS to efficiently manage the I/O between primary and secondary storage. This allows Parka-DB to scale to essentially arbitrarily large KBs while still outperforming other KR systems.

KB-DB Interface To maintain efficiency in inferencing, we differentiate two categories of concepts: structural and non-structural assertions. Structural assertions consist of the isa, instance, instanceof, and subcat relations that encode the class/subclass ontology in a Parka KB. Non-structural assertions are all other concepts in the KB.

This differentiation is important from a practical viewpoint – the structural assertions are used in property inheritance. In particular, Parka-DB must scan the structural assertions for computing inherited properties, and inheritance evaluation is a critical aspect of all queries issued to Parka-DB. Thus, the structural assertions are kept in internal memory for fast access. In all of the cases we’ve examined, the number of structural assertions is less (often much less) than the number of non-structural assertions. Therefore, we can keep these structural assertions in internal memory even for our largest KBs (we discuss actual memory requirements in section 4).

On the other hand, the knowledge bases we are using in the previously described applications require more memory than is available on all but the largest current computer systems (particularly supercomputers). They certainly don’t fit in the amount of internal memory expected to be found in “affordable” computer systems any time in the near future. Furthermore, we find that it is very unlikely that all of the data contained in the set of non-structural assertions will have to be processed for a given query (unlike the structural assertions, which are often all scanned). Therefore, the non-structural assertions are not required to be kept in primary memory at all times. For these reasons, Parka-DB stores non-structural assertions in specialized tables within the database. This allows the database system to handle
the loading of only that subset of non-structural assertions required to evaluate a given query, a process which DBMS systems manage efficiently. Thus, to summarize, by dividing data into two categories, Parka-DB can access the highly utilized structural assertions quickly, and rely on the database system to manage the much larger and less frequently accessed individual items of non-structural data efficiently.

To facilitate the efficient moving of data between the inferencing algorithms and the database, we have had to slightly modify the database system. This will be described in the next section, but first we wish to briefly clarify the interaction between the two components. Parka, as a KR system, is primarily concerned with answering queries. In the process of evaluating these queries, Parka issues two main requests to the database. Figure 1 illustrates the interaction between the modules. One request is a command to project data from a non-structural assertion table. The other is a request to perform a join over two tables.

In this manner, Parka-DB utilizes the processing power of the database for those services that a RDBMS can perform more efficiently, mainly join processing. In fact, Parka-DB's query algorithms have been designed such that many of the computations can be directly executed in the form of a join over two relational tables. Thus, the KR component of Parka-DB relies on the database not only for storage and retrieval services, but also for the intermediate processing of complex queries. In this manner, Parka-DB can also take direct advantage of database optimizations (for example, the use of parallelism for joins in a parallel DBMS).

Parka-REL Parka-DB is implemented on top of an RDBMS system we have created at the University of Maryland. This system, Parka-REL, offers the standard DDL interface for creating databases. Creating relational tables
is straightforward and can be achieved through direct function calls from a C program or through a parser. To offer efficient access to user data, the system provides users the opportunity to create indexes on attributes of relational tables. Once the tables have been defined and created, users can seamlessly insert and delete records. Parka-REL provides storage and retrieval functionality, and all of the standard database operations required in the framework of Parka-DB. Parka-DB communicates with Parka-REL directly through the RDBMS’s library of C function calls, rather than through SQL (although a limited SQL interface is provided for user access to the DB). The RDBMS, though limited in functionality, is very useful and supports Parka-DB adequately. We believe the limitations of Parka-REL are more than outweighed by the fact that Parka-REL can be made publicly available and thus Parka-DB distributed without the need for users to buy an expensive DBMS system. (We are, however, interested in some other database features not provided in Parka-REL, see Future Work.)

**Data Storage**  One of the most important aspects of our implementation is that although Parka-DB is in all ways a frame-based KR system, its knowledge base is actually stored as relational tables to take full advantage of Parka-REL’s processing services. In particular, each assertion type is associated with a relational table in Parka-REL. Frames in Parka-DB thus have their information distributed amongst several tables.

This contrasts with other attempts to use database technology for supporting knowledge bases (see section 5). Other systems [11, 16] store frames in a much more centralized manner – they attempt to gather as much information about a frame as possible when loading data from external storage. Parka-DB only loads into memory that information which is needed for processing a query. In fact, we believe that one reason Parka-DB is more efficient than these other systems is that its inferencing algorithms have been designed to take full advantage of this distributed relational(frame) data.

When evaluating complex queries on large knowledge bases the sizes of intermediate results can be enormous. Computing the results of such queries using only internal memory can place a big demand on system resources, especially if the data to be processed is also stored in internal memory (as in most KR systems). Parka-DB overcomes this problem by distributing the workload such that the KR component prepares relational tables for
the RDBMS component to process using external storage as it needs. The
RDBMS unit can consume available disk space while processing intermediate
results thus reducing the demand for valuable internal system resources.

4 Performance

As discussed above, the redesign of the previous version of Parka into Parka-
DB was primarily to make greater use of secondary storage over primary
memory (using the RDBMS). This clearly was expected to result in smaller
use of internal memory and faster KB loading times (since less is moved from
disk to memory) at the expense of greater querying times. In this section,
we describe how Parka has performed on these issues.

Memory use Although all the information needed by Parka could be kept
in Parka-REL and loaded into memory as needed, thus fully minimizing mem-
ory use, we keep the inheritance structure (structural assertions) in primary
memory for efficiency reasons (as discussed previously). For the biggest KB
we used in our tests (Caper-200, see below) the space required for this data
was only 1.1 M-Bytes of RAM.

Additional use of memory is made by the inference engine which must
maintain a small amount of information about each frame in the system.
The space currently used is 12 Bytes per frame, and the number of frames
is typically low compared to the number of assertions, so this also is not a
major use of space. Where there is enough physical memory to guarantee
this space Parka-DB executes most of the inferencing algorithms in memory.
I/O is only performed to save intermediate and final query results in the
DBMS.

Parka-REL also consumes physical memory to provide its services. The
RDBMS allocates memory for a buffer pool with 40 buffers, each using 4
K-Bytes. This is a small amount of memory and the performance of Parka-
REL would improve with more memory. A larger buffer pool and/or a bigger
page size would allow Parka-REL to store more data in physical memory
and reduce the amount of disk I/O, in situations where Parka-DB repeatedly
requests the same information. Thus, including the resources consumed by
the graphical query interface, Parka-REL, and Parka-DB, the total physical
memory consumed never exceeded 16 M-Bytes of physical memory even for
<table>
<thead>
<tr>
<th>Knowledge Base</th>
<th>#Frames</th>
<th>#struct. Links</th>
<th>#Asserts.</th>
<th>in-memory</th>
<th>Parka-DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caper-20</td>
<td>11103</td>
<td>26404</td>
<td>118615</td>
<td>28</td>
<td>0.29</td>
</tr>
<tr>
<td>Caper-100</td>
<td>56516</td>
<td>130526</td>
<td>807480</td>
<td>214</td>
<td>1.34</td>
</tr>
<tr>
<td>Caper-200</td>
<td>116297</td>
<td>266100</td>
<td>1635782</td>
<td>640</td>
<td>2.80</td>
</tr>
</tbody>
</table>

Table 1: Comparison of previous version and Parka-DB KB Loading time (in seconds)

the Caper-200 KB. As a comparison, holding this entire KB in the memory-based version would take 162MB of RAM, so we see a considerable advantage to the new version.

**KB Loading times**  In the previous version of Parka the process of loading a KB into memory was very expensive. Since all the data had to be loaded into primary memory, the KB was kept in the form of a flat ASCII file containing the frame descriptions — there was no advantage to storing intermediate structures, since everything had to be read in. Instead, special hashing and other functions were developed to make the loading as efficient as possible.

Table 1 shows the sizes (in terms of frames and assertions) of the three KBs we have used in testing the system. All three of these were generated for the Case-based planning work described in section 2 and discussed in [12]. The case-bases were created via a generative planning system, and contain information on the plans, sub-plan structure, and causal information used by the planner, as well as ontology information about the logistics-planning domain. Details on this work and the planning domain can be found in

http://www.cs.umd.edu/projects/plus/Caper/ and


As can be seen, even the biggest KB, which has over 2,000,000 assertions, can be loaded in under 3 seconds (over 200 times as quickly as when the whole thing is processed and loaded). This incredible improvement is created since in Parka-DB the KB is saved in a denser format which drastically reduces the number of I/O operations, since only a very small part of the KB is actually loaded.
<table>
<thead>
<tr>
<th>Query</th>
<th>Preproc.</th>
<th>Inferencing</th>
<th>I/O</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>KB</td>
<td>DB</td>
<td></td>
</tr>
<tr>
<td>Q1(Caper-020)</td>
<td>0.001</td>
<td>0.049</td>
<td>0.157</td>
<td>0.609</td>
</tr>
<tr>
<td>Q2(Caper-020)</td>
<td>0.001</td>
<td>2.258</td>
<td>0.652</td>
<td>3.978</td>
</tr>
<tr>
<td>Q1(Caper-100)</td>
<td>0.001</td>
<td>0.336</td>
<td>3.379</td>
<td>4.4867</td>
</tr>
<tr>
<td>Q2(Caper-100)</td>
<td>0.001</td>
<td>13.761</td>
<td>10.223</td>
<td>28.860</td>
</tr>
<tr>
<td>Q1(Caper-200)</td>
<td>0.001</td>
<td>0.590</td>
<td>12.329</td>
<td>15.575</td>
</tr>
<tr>
<td>Q2(Caper-200)</td>
<td>0.001</td>
<td>27.345</td>
<td>33.519</td>
<td>70.177</td>
</tr>
</tbody>
</table>

Table 2: Timings for query processing (seconds)

**Query Processing**  Obviously, by making greater use of (slower) secondary memory we lose efficiency since we are trading space in primary memory for time. In this section, we analyze this trade-off directly, examining how much of the query time is spent in I/O, how much in the KB, and how much in the DB. Our analysis was based on a number of queries generated by the Caper system during its case-based planning. In this abstract, we show two of these, the fastest and slowest to process. Others were in between, with most closer to the faster times. The first query, which contains 10 conjuncts concerning 5 variables, corresponds to the query “Show all plans in which a train was used to move a package through Hub-TS”, and the second query, with 8 variables and 15 conjuncts asks to “show all plans in which the package was a liquid package, was moved in a tanker truck, and was taken to a location that is of type ‘city-location’.”

The process of querying the KB is divided into three phases. The query interface provided to the user is the same independent of the version of Parka used. For this reason issued queries are first preprocessed for evaluation by the appropriate version of Parka. Once the preprocessing is done the query is processed by a sequence of KB and DB operations. Based on table 2 we see the following:

- **Query Preprocessing** With Parka-DB some preprocessing is needed. In the first phase, the consistency of the query is checked. The second phase is used to define a join tree that specifies the order in which expressions in queries

---

3 *Note to reviewers:* Full paper will include more queries and the actual queries will be shown using screen dumps of our graphical query interface - space limitations prohibit including these here.
are to be evaluated. During this process several optimization strategies are used to define a reasonable query evaluation order. Preprocessing is currently extremely fast (column 2), although we believe more time spent here can bring down the total, this is a direction of our current work.

- **Inferencing:** The timings for the inferencing part of query processing is divided into subparts, the amount of time spent by KB-operations (particularly IDO-inheritance) and the amount of time used by the DB-operations (selects, joins, ...). These timings are shown in column 3 and 4. The time spent by the DB-operations, in particular, could be reduced by using better strategies during preprocessing.

- **I/O Operations** The I/O operations are the overhead we must pay since the whole KB is no longer loaded into memory. The numbers in columns 5 and 6 show this time. The input cost is incurred by loading data into memory and by loading intermediate results of the query processing. The output cost is indicative of the processing time required by Parka-REL to write intermediate and final results. Its notable that the I/O process accounts for only approximately 10% of the total processing time. Based on these results we see that the additional I/O time caused by moving items between primary and secondary storage imposes only a small overhead on the system. In addition, significantly faster loading times are an added benefit for the database implementation.

5 Related Work

Several research projects have dealt with the use of secondary storage to support large frame-based systems. Karp and Paley [11] discuss integrating database storage technology into KR systems. They pull in much data about individual frames, rather than distributing the frames as we discussed earlier. Thus, their system functions more as a “persistent store” than as a query system such as ours. The IDI (Intelligent Database Interface) project [16] can also be viewed in a similar manner, although that work is meant to deal more with access to distributed DBs than as a KR system per se.

Another system dealing with disk-based knowledge is Haase’s Framer system [8]. Framer is an object oriented storage system designed to be used to implement a KR system. The system provides the basic routines for simple inferencing mechanisms, but provides neither a full KR language nor the sort
of conjunctive querying provided by Parka-DB. Another alternate approach is that of Thenetsys [17] which is a semantic network system that employs somewhat of a virtual memory approach to semantic network management. In Thenetsys, semantic networks are stored on disk. When a node from a network is requested, Thenetsys has to find room for the node and its related data. If memory is nearly full, Thenetsys has to remove some nodes of a semantic network from memory, much like a virtual memory manager. Again, this system will not support the same sorts of conjunctive queries as supported in Parka-DB, nor have either of these systems yet been shown to scale to the size of the KBs being used at Maryland.

In fairness to these other systems, one problem with demonstrating these (and other KB systems) on very large KBs is the difficulty in obtaining these. In an effort to make such comparisons possible, the case-based KBs described in this paper are being made publicly available on the WWW - See http://www.cs.umd.edu/projects/plus/Parka

6 Conclusion & Future Work

Real world applications are demanding that KR systems provide support for KB’s containing millions of assertions. We have presented Parka-DB, a high performance KR system based on implementing a standard KR using a relational DBMS system. The integration of a small DBMS (Parka-REL) with a standard KRS (Parka) allows us to efficiently support complex queries on extremely large KBs using a single processor. In addition, the system can make good use of secondary memory, with the whole system needing less than 16MB of RAM to hold a KB of over 2,000,000 assertions. We demonstrate empirically that this reduction in primary storage requires only about 10% overhead in time, and decreases the load time of very large KBs by more than two orders of magnitude.

Based on these results, we believe that the Parka-DB architecture is a cost-effective approach to scaling knowledge representation to very large KBs. Parka-DB is designed to consume as little internal memory as possible while using more (less expensive) disk space. Even though the price of RAM is falling, the price of RAM is still more expensive than disk space. By using less internal memory and more external memory, Parka-DB’s utilization of system resources is quite efficient. Considering the efficient use of system re-
sources and its high performance, Parka-DB offers a framework for knowledge representation that is adequate for managing huge KB's while not requiring extensive system resources.

We see two important directions to take this work. The first is to make more use of the database features than done to date. While Parka-REL has obvious advantages (it's home-grown and freely distributable), it doesn't have some features we believe might be of use in future KB systems. These include the security, concurrency, and user profiling features that many commercial RDBMS systems provide. Parka-DB has been designed to take advantage of such features if available, and we have begun discussions with various database researchers to explore the possibility of joint work in this direction.

The second direction is to improve the performance of our algorithms. As alluded to previously, more time spent in preprocessing should yield better performance. In recent experiments, we have found that we can improve efficiency by a full order of magnitude if we hand optimize the queries to minimize the size of internal results. This reduces the size of the inheritance scans (thus reducing the KB inferencing times) and also reduces the size of the relational joins (reducing the DB times). A CM-5 multiprocessor version of Parka [2] used heuristics to do just this sort of optimization, and we believe we can duplicate this work in the new framework at the cost of only a minimal amount of additional preprocessing time.

Secondly, Parka was originally designed for parallel supercomputing environments [6] and the algorithms have been proven to scale well to a wide range of supercomputing systems [19] guaranteeing a parallel efficiency better than 50% up to 16 processors for the case-bases described in this paper. In addition, we have recently prototyped a version of Parka's inferencing algorithms on the distributed Mosix system developed at Hebrew University [3] with speedups through more than 12 processors on an ethernet-based system. We are now working on porting Parka-DB to both parallel and distributed systems, and expect to see similar speedups.

Combining these two promising research directions with the low overhead of the I/O times shown in this paper, we are committed to two performance goals for the Parka-DB system: to be able to handle queries of the complexity of those shown in this paper in sub-second times for KBs of several million assertions (by the end of 1996), and to scale the system to handle these queries on KBs containing over one hundred million assertions (by the end of 1997).
References


