A parallel inference model for logic programming

by JIE-YONG JUANG and DANIEL CHENG
Northwestern University
Evanston, Illinois

ABSTRACT

In this paper, we describe a parallel inference model for logic programming on general-purpose multicomputers. In the model, input clauses are partitioned into subsets, and resolution is conducted on each subset concurrently. The partitions are dynamically adjusted via clause migration as inference proceeds. This allows each processor to work on virtually the whole clause set while a shorter resolution cycle is achieved. In the context of AND/OR tree space search, the parallel model explores another dimension of parallelism in addition to AND/OR parallelism. It implicitly forces multiple processors to jointly search the same path that leads to a refutation. Problem-solving heuristics can be incorporated in the parallel model systematically to determine clause partitions and guide inference. With a distance measure derived from problem-solving heuristics, a partition that has the best combination of clause subsets and a small rate of clause migration can be obtained using existing clustering algorithms. Clause migration decisions can also be made based on the distance measure. Last, we point out that all the mechanisms of the parallel model can be efficiently supported by a connection graph. The graph also simplifies the implementation of the subsumption strategy.
INTRODUCTION

Logic inference is a method of acquiring new knowledge from a set of known facts represented as clauses, and resolution is the most commonly used technique for logic inference. Logic programming languages such as Prolog have been successfully used in solving complex problems, especially in developing expert systems. In resolution, a theorem, that is, new knowledge, is formulated as a goal statement, and the known facts are treated as a set of consistent axioms. Then, a proof procedure is applied to show that the denial of the goal statement is false, a refutation exists. The procedure involves repeated cycles of matching, unification and resolution of two clauses. The matching process identifies a pair of clauses such that the positive form of a literal appears in one clause and the negative form in the other. These two literals are unifiable if their variables can be unified by substitution during the unification process. After the two literals are unified, a new clause called the resolvent is generated by canceling them and combing the remaining literals. The two clauses involved are called the resolvable pair in this paper. The proof procedure terminates when an empty clause is generated.

The resolution procedure is slow when it runs on today's computers. Many efforts have been made to speed up logic inference, and parallel processing has been identified as one of the most promising approaches. Previous work in this area is classified into five categories according to the level of parallelism they explore: (1) subrule level, (2) rule level, (3) search level, (4) language level, and (5) system level. At the subrule level (or the architecture level), emphasis has been placed on the exploitation of concurrency during unification because unification is the most often invoked module in logic inference. Pipeline architectures and construction of a dedicated unification hardware fall in this category. For the rule level, efforts have concentrated on parallel matching of clause pairs that have unifiable literals. Search-level parallelism mainly deals with the selection of clauses, from the whole clause set, to resolve, and is gaining attention in the field of logic programming. Sharing resources is the main subject in system-level parallelism. To facilitate the automatic exploitation of parallelism in logic inference, many parallel logic programming languages have been proposed. Their implementations are in progress.

However, the revelation of the low degree of parallelism in typical logic programs and the difficulty of sharing variables make intensive parallelism difficult to achieve. In this paper, we propose a parallel model of logic inference in which the clause set of a problem domain is decomposed into several distinct partitions. Logic inference on each partition is conducted at a different processor in a multiprocessor system simultaneously. Intermediate derivations in each partition are shared between partitions via clause migration. A formal method of partitioning the clause set is proposed, that includes sharing intermediate derivations via clause migration.

A PARALLEL INFERENCE MODEL FOR LOGIC PROGRAMMING

Our objective is to design a parallel logic inference scheme that will run on general-purpose multicomputers including multiprocessors and local area networks. Since no specialized hardware is assumed for such a scheme, the parallelism will be explored primarily at the search level. Major problems confronting existing schemes are investigated below to motivate our design.

Problems in Existing Parallel Inference Schemes

According to recent studies, parallelism at the subrule level and the rule level is limited. There was speculation, on the other hand, that search-level parallelism could offer a significant opportunity for a large degree of concurrency. Concurrency is a consequence of non-determinism in logic inference. That is, the order of resolution affects only the efficiency, not the correctness of the inference. Therefore, many clauses can be resolved simultaneously. Nevertheless, the non-determinism also has an adverse effect on logic inference. In each resolution cycle, the two clauses to be resolved can be selected out of C(n,2) possible combinations. If the selection is unrestricted, the resolution cycle can be very long when the clause set is large. To tackle this problem, two restrictions are imposed: (1) only Horn clauses are allowed in the clause set, and (2) either a top-down or a bottom-up proof procedure is used. The resulting inference procedure can be viewed as a search process in an AND/OR tree space. Each alternative branch in the AND/OR tree offers a possibility for parallelism since it represents a subtree that can be searched independently. The resulting inference procedure retains a high degree of parallelism, even though a strict sequence is imposed when it searches along a path. Nevertheless, due to sharing variables between AND branches and the small number of OR branches found in most existing programs, concurrency in AND/OR tree search is also limited in practice.

A procedure with a high degree of parallelism is able to exploit the potential of a multiprocessor since many processors will be kept busy for most of the time. For a conventional deterministic task, such as numerical analysis, better processor utilization implies greater speedup. Unfortunately, logic inference is non-deterministic. Keeping pro-
processors busy does not guarantee a speedup. In other words, many processors may not be doing useful work even though they may be busy all the time. To prevent a processor from doing unproductive work, schemes that use the dominance relation to eliminate unnecessary clauses were shown to be very effective.\textsuperscript{16, 17} Clause subsumption is a notable example.\textsuperscript{22} Unfortunately, it is difficult to implement in an AND/OR tree search procedure.

Problem-solving heuristics provide another important approach to improve the efficiency of logic inference. It may be used to guide the selection of resolvable clauses. However, in its current form, it is hard to incorporate problem-solving heuristics in an AND/OR tree search procedure.\textsuperscript{16}

\textbf{Partitioning Clause Set for Parallelism}

Since a matching process selects resolvable pairs from the whole clause set, resolution on a small clause set is usually much faster than on a large one. Another fact that can be observed is that a successful inference does not always involve all the clauses. These two observations suggest that inference can be conducted on all the possible subsets of clauses concurrently; each is carried out by a different processor. The inference terminates as soon as a processor finds a proof. If the processor that found the proof happens to work on a small subset, the inference time can be very short. Although such an approach toward parallel logic inference looks very promising at the first glance, it is impractical. For a set of \( n \) clauses, there are \( 2^n \) subsets. It is impossible to exhaust all the subsets even for a medium-sized clause set. However, this approach can be made practical by the following modification. Initially, clauses are partitioned into as many subsets as the number of processors available. In this way, we can run the procedure on a system with an arbitrary number of processors, and all the processors can be kept busy all the time. Nevertheless, a subset so obtained usually does not contain sufficient clauses for a successful inference. To cope with the problem, clauses are transferred from one subset to another as inference proceeds. The migration of clauses adjusts the partition dynamically so that a refutation can be found in a subset. Thus, clause migration is essentially a robust clause partitioning scheme. To clarify this concept of parallel logic inference, the basic steps involved are outlined in the following procedure:

\textbf{Procedure: Parallel-Logic-Inference}

\begin{verbatim}
Begin
  Partition the input clause set into subsets;
  Load each subset into a processor;
  All the processors run the following loop concurrently:
    Loop until success or there are no more resolvable pairs
      Run a local inference cycle;
      If the resolvent is an empty clause, then set success flag;
      Invoke clause migration if necessary;
      Split into two subsets and request for another processor if local subset becomes too large;
    end Loop;
  end.
End.
\end{verbatim}

Due to clause migration, a processor in the above procedure conducts inference on virtually the whole clause set, though it is in fact working only on a small subset of clauses. Thus, the parallel inference procedure can be viewed as a form of virtual inference.

In the context of an AND/OR tree search space, the parallel model allows three forms of parallelism. In addition to AND parallelism and OR parallelism, processors may also jointly work on the same search path via sharing intermediate inference results. This parallelism is implicitly achieved by clause migration, which does not exist in the top-down or bottom-up proof procedure.

The resolution procedure is non-deterministic, and the partitioning approach creates no shared variables. Hence, no synchronization between processors is necessary. All the processors can run concurrently in the above procedure. Furthermore, resolution and clause migration from one processor to another can also be carried out asynchronously.

\textbf{Issues in the Parallel Inference Procedure}

The proposed parallel inference model consists of three basic components: the initial partition, clause migration, and local inference. Each involves several unsettled issues.

The initial partition is the basis of the whole inference procedure. A proper partition will reduce the rate of clause migration and improve the inference speed. Thus, clauses have to be grouped in such a way that local inference is most productive and the clause migration rate is minimal. The main issues that have to be investigated in determining the initial clause partition are the proper size of a clause subset and how related clauses are grouped together. Factors related to the target machine, such as processor speed and interprocessor communication delay, should be taken into consideration. However, problem-solving heuristics are more important since logic inference is extremely problem-dependent.

Clause migration dynamically changes the partition of the clause set. Each processor has to determine which clause should migrate, when and where. Making such decisions involves not only static information but also information about the dynamic status of the inference. Thus, besides problem-solving heuristics, an efficient data structure for maintaining status information is important to making decisions about clause migration.

Local inference is the component that does the actual work of resolution. Like a conventional inference procedure, it repeats a resolution cycle that consists of matching a pair of clauses, unifying the matched literals, and resolving the pair. Issues that have to be investigated include how to reduce inference cycle time and how to identify the most promising pair. Inference cycle time may be reduced by keeping the local clause subset small and organizing it into an efficient data structure. The subsumption strategy can also help to reduce the inference cycle since it can eliminate unnecessary clauses, but it requires an efficient data structure because much searching is involved. Determining which pair to resolve, on the other hand, relies on problem solving heuristics. Ranking resolvable pairs requires an effective way to capture knowledge about the problem.
Incorporating problem-solving heuristics and employing efficient data structures are two keys to a successful design of the parallel logic inference procedure. Depending on the problem to be solved and available knowledge about solving it, different heuristics may be applied. This requires a parallel logic inference procedure that adapts different heuristics from problem to problem. Few existing parallel logic inference schemes provide mechanisms for this purpose. We will describe a systematic method for incorporating different problem-solving heuristics into the parallel inference procedure and a unified data structure for supporting clause migration, local inference and subsumption.

PROBLEM-SOLVING HEURISTICS

Among resolvable pairs, some will lead to a refutation, but not all. The best inference procedure, would be the one in which every resolution makes progress toward a proof. It is hard to achieve due to the nondeterministic nature of logic inference. This motivates the use of problem-solving heuristics to predict which resolvable pair is more likely to lead to a proof so that that number of inference cycles can be minimized. For example, in the set-of-support strategy, a set of clauses is “supported” and one of these clauses or their descendants should be included in each resolution. Accordingly, any resolvable pair with a supported clause is considered more useful than those without supported clauses. In the unit-preference strategy, clauses with fewer literals are resolved first since these clauses are more likely to generate an empty clause. The unit-preference strategy belongs to a general class of problem-solving heuristics, called ordering strategies.

An ordering strategy ranks clauses to determine the order in which resolutions are performed. These strategies can be adapted to the local inference part of the proposed parallel inference model, but are difficult to apply to the initial partition and clause migration parts.

Our objective is to provide a unified mechanism for incorporating problem-solving heuristics to guide local inference, conduct initial partitioning, and determine clause migration. To this end, the problem-solving heuristic is transformed into a preference measure from which the order of resolution in local inference can be obtained directly. The measure is then transformed into another measure that helps grouping clauses into appropriate subsets.

Resolution Preference

There may be several unifiable literals in a resolvable pair, and the pair can be resolved with respect to each of them. Resolution on different unifiable literals may have different effects on the efficiency of inference, though they operate on the same clause pair. A problem-solving heuristic based on clause ordering will not reflect this fact. Thus, for a more productive local inference scheme, the preference measure should lend itself to the ranking of unifiable literals instead of individual clauses.

It is easy to encode problem-solving heuristics using a preference measure. For example, if the set of support (SOS) strategy is used, links attached to clauses having support can be placed at a preference level an order of magnitude larger than others. If the kind of strategy used in the inference process defines a function value (e.g., a priority function, a weighting, or a literal number), preference measures can be assigned values in proportion to the function values.

Another important factor is the nature of the inference procedure. As Kowalski suggests, resolutions which lead to the simplification of the graph should be performed before others. With this regard, two connected literals, each of which has only this link attached, can be resolved upon without generating new clauses or links. Therefore, these links should be assigned a higher preference level than others.

Distance Between Two Clauses

The order of local resolution can be derived directly from the concept of preference. The grouping of clauses, however, cannot be determined with the same measure. Instead, it requires a measure that characterizes the similarity between clauses. We will call such a measure the distance between two clauses in this paper. Since the clause is the smallest unit in the initial partition and clause migration, a distance measure should be provided at this level.

The next question is how to transform the problem-solving heuristics into the distance measure. It can be done independently. However, in order to provide an unified transformation mechanism, the distance measure should be correspondingly derived from the preference measure. The advantage of doing so is that users only have to deal with the problem-solving heuristics, as in the conventional inference systems. Consequently, the partition and migration processes can be transparent to users.

For two clauses that are resolvable, their distance can be determined by a function, which combines the preference levels of those unifiable literals associated with them. The function can be provided by the system as a default. It can also be overwritten by the user with one that is more appropriate for the problem to be solved.

Two clauses may not be resolvable due to the lack of unifiable literals. However, their descendants may be resolvable. The distance between these two clauses, thus, can be obtained indirectly from other resolvable pairs. Consider an example in which Clauses Cl.1 and Cl.2 are not resolvable, but both are resolvable with Clause Cl.3. In this case Cl.2 will be resolvable with the resolvent of Cl.1 and Cl.3. Let D(x,y) represent the distance between Clause x and Clause y. Then, D(Cl.1, Cl.2) can be reasonably defined as the sum of D(Cl.1, Cl.3) and D(Cl.2, Cl.3). In general, more complex functions than a summation can be used. Nevertheless, a linear function is desirable because it is convenient to manipulate.

The distance between two non-resolvable clauses is infinite if none of their descendants are resolvable. The distance measure of a resolvent inherits that of its parent clauses with some adjustments.

Initial Clause Partitioning According to Distances

After a distance measure has been defined for every pair of clauses, partitioning can be directly converted into a cluster-
Determining Clause Migration Based on Distances

A clause should migrate from one subset to the other when it has a higher preference to be resolved at the remote site. The difference in preference is reflected in the distances between this clause and its local and remote counterparts. If the remote distance of the clause is less than its local distance, then this clause has preference to be transferred to the remote site. When a clause is not resolvable with any clause in the local subset, its local distance is infinite. Thus, apparently, it should migrate to another partition. Essentially, the criteria for clause migration are equivalent to the protocol that realizes a distributed adaptive clustering algorithm.

UNIFIED DATA STRUCTURE

The connection graph proposed by Kowalski was found to be efficient for supporting the parallel logic inference. In a connection graph, each literal is represented as a graph node, and nodes representing the literals of a clause are grouped together. Unification is then conducted to match every pair of literals that have the same predicate symbol and are complementary in sign. Unifiable pairs of literals are indicated by graph links, and each link is labelled by the most general unifier (MGU) of that link. The graph representation of the clause set in Figure 1(a) is shown in Figure 1(b). Such a graph is simple, yet provides a unified framework for managing resolvable pairs, facilitating subsumption, and supporting clause migration.

Managing Resolvable Pairs

In a blind clause matching, resolvable pairs of clauses are determined by searching over the whole clause set from scratch in every resolution cycle. The procedure is highly redundant since the set of resolvable pairs does not change significantly from iteration to iteration. With a connection graph, the unifiable clauses are identified beforehand and updated during the inference process. Because information about unifiable clauses is maintained, the matching process in each resolution cycle is immediately eliminated.

Facilitating Subsumption

Subsumption is effective but may incur a large overhead. It involves exclusive searching for candidates to be subsumed, and is invoked whenever a new clause is generated. With the connection graph, those candidates can be located immediately since all of them are exactly two hops away from the new clause.

Computing Distances

To compute the distance measure of two unresolvable clauses, we need to know all the indirect resolvable pairs that relate the two clauses. This is readily available in the connection graph since these pairs form a chain in the connection graph.

Supporting Clause Migration

Local and remote distances have to be collected in making clause migration decisions. Since distance is incremental with respect to the number of indirect resolvable pairs involved, only those clauses on the partition boundary are candidates to be exchanged between subsets. If the input clause set is organized into a connection graph, these clauses can be determined immediately, and potential migration paths are represented by links crossing over the partition boundary. Hence, the overhead of clause migration can be greatly reduced by a connection graph.

From the collection of the Computer History Museum (www.computerhistory.org)
AN EXAMPLE

An example is described in this section to illustrate the key features of the parallel model presented in previous sections. In order to directly compare the performance of this model with that of PROLOG systems, a Horn clause set is used, as shown in Figure 1a. From the input clause set, a connection graph is first constructed as shown in Figure 1b.

Our next task is to decompose the resulting graph through a clustering algorithm. For this example, let's define the preference measure of a pair of unifiable literals to be the sum of the literals in the two associated clauses.

If there exists only one link between two clauses, the distance measure is simply set to the preference measure of their unifiable literals; otherwise, the minimum of the preference measures of the links connecting them is taken. Accordingly, the distance between Cl.1 and Cl.2 is 4, between Cl.2 and Cl.4 is 6, and so on. For clauses having no unifiable links between them, the distance measure is defined to be the shortest path between them along the unifiable links. Thus, the distance measure between Cl.1 and Cl.4 is 10(4 + 6), between Cl.1 and Cl.9 is 13(4 + 5 + 4).

The complete distance measure between every pair of clauses is shown in Table I.

The clustering algorithm is then invoked to run on this distance matrix which results in three partitions of the initial graph, as shown in Figure 2, assuming three processors are available. From Figure 2, we observe that the partitions are suitably cut on unifiable links, which have the largest distance measures, as we expect. Also, notice that clauses are evenly distributed over partitions, which is a quite desirable situation. Each partition is, thereafter, loaded into one processor for execution.

The links crossing over partitions, external links, are the points where communication takes place between partitions.

### TABLE I—Distance matrix of the clause set in Figure 1

<table>
<thead>
<tr>
<th>Clause</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td>9</td>
<td>11</td>
<td>15</td>
<td>13</td>
<td>13</td>
<td>17</td>
<td>17</td>
<td>16</td>
<td>20</td>
<td>20</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>11</td>
<td>9</td>
<td>9</td>
<td>13</td>
<td>13</td>
<td>14</td>
<td>12</td>
<td>17</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>5</td>
<td>0</td>
<td>11</td>
<td>10</td>
<td>10</td>
<td>12</td>
<td>10</td>
<td>14</td>
<td>9</td>
<td>9</td>
<td>13</td>
<td>13</td>
<td>12</td>
<td>18</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>6</td>
<td>11</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>13</td>
<td>5</td>
<td>5</td>
<td>15</td>
<td>19</td>
<td>19</td>
<td>9</td>
<td>8</td>
<td>22</td>
<td>22</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>5</td>
<td>10</td>
<td>11</td>
<td>0</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>4</td>
<td>14</td>
<td>14</td>
<td>18</td>
<td>17</td>
<td>7</td>
<td>21</td>
<td>21</td>
<td>17</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>5</td>
<td>10</td>
<td>11</td>
<td>10</td>
<td>0</td>
<td>12</td>
<td>18</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>18</td>
<td>17</td>
<td>17</td>
<td>19</td>
<td>19</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>7</td>
<td>12</td>
<td>13</td>
<td>12</td>
<td>12</td>
<td>0</td>
<td>18</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>21</td>
<td>18</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>13</td>
<td>11</td>
<td>16</td>
<td>5</td>
<td>14</td>
<td>16</td>
<td>18</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>24</td>
<td>24</td>
<td>3</td>
<td>13</td>
<td>27</td>
<td>27</td>
<td>23</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>13</td>
<td>14</td>
<td>5</td>
<td>4</td>
<td>14</td>
<td>16</td>
<td>0</td>
<td>18</td>
<td>22</td>
<td>22</td>
<td>13</td>
<td>3</td>
<td>15</td>
<td>31</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>13</td>
<td>9</td>
<td>6</td>
<td>15</td>
<td>14</td>
<td>4</td>
<td>6</td>
<td>20</td>
<td>18</td>
<td>0</td>
<td>12</td>
<td>12</td>
<td>23</td>
<td>31</td>
<td>21</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>17</td>
<td>19</td>
<td>18</td>
<td>19</td>
<td>18</td>
<td>18</td>
<td>0</td>
<td>24</td>
<td>22</td>
<td>12</td>
<td>0</td>
<td>15</td>
<td>27</td>
<td>25</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>17</td>
<td>15</td>
<td>18</td>
<td>19</td>
<td>18</td>
<td>18</td>
<td>0</td>
<td>24</td>
<td>22</td>
<td>12</td>
<td>12</td>
<td>0</td>
<td>27</td>
<td>15</td>
<td>15</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>16</td>
<td>14</td>
<td>19</td>
<td>8</td>
<td>17</td>
<td>19</td>
<td>21</td>
<td>3</td>
<td>13</td>
<td>23</td>
<td>27</td>
<td>27</td>
<td>0</td>
<td>16</td>
<td>30</td>
<td>30</td>
<td>26</td>
</tr>
<tr>
<td>14</td>
<td>16</td>
<td>12</td>
<td>17</td>
<td>8</td>
<td>7</td>
<td>17</td>
<td>19</td>
<td>3</td>
<td>13</td>
<td>21</td>
<td>24</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>29</td>
<td>24</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>15</td>
<td>13</td>
<td>12</td>
<td>9</td>
<td>7</td>
<td>17</td>
<td>19</td>
<td>15</td>
<td>13</td>
<td>13</td>
<td>30</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>16</td>
<td>15</td>
<td>15</td>
<td>13</td>
<td>12</td>
<td>13</td>
<td>12</td>
<td>12</td>
<td>3</td>
<td>13</td>
<td>30</td>
<td>30</td>
<td>8</td>
<td>16</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>15</td>
<td>13</td>
<td>12</td>
<td>3</td>
<td>8</td>
<td>18</td>
<td>17</td>
<td>7</td>
<td>23</td>
<td>21</td>
<td>3</td>
<td>15</td>
<td>15</td>
<td>20</td>
<td>9</td>
<td>18</td>
<td></td>
</tr>
</tbody>
</table>

In this case, there are two between partitions A and B, two between partitions A and C, and none between partitions B and C. A copy of the MGU of each external link is maintained in each connecting partition.

In the execution of inference on each partition, the processor picks a link of its partition, and resolves upon this link to generate a new clause, the resolvent. It selects links in ascending order of preference measures. Ties are broken randomly, but in favor of internal links.

In Figure 3, a snapshot is shown after two steps of inference have been performed in each partition through the resolution of two links, indicated by the darkened lines in Figure 2 and selected according to the strategy, assuming equal execution time for these resolutions. Except for Cl.10, whose new MGU is updated through interprocessor communication, these beginning resolutions are completely done.
within each partition. Inside partition A, we further observe that Cl.6 ought to be subsumed by Cl.3. After the deletion of Cl.6 and its links, Cl.10 becomes isolated in partition A and is subject to migration. It migrates to partition C as its only link goes to partition C, shown in Figure 4.

During the migration of Cl.10, we assume two inference steps are also taking place in partition B and C, and one inference step in partition A. The resulting situation is shown in Figure 5. For resolution performed on clauses with external links, the new MGU is checked for compatibility, as well as, internal links. Any incompatible links are immediately deleted, as is the case, with the link between Cl.2 and Cl.3. If the external links are incompatible, they are deleted from both clauses through the communication protocol. If the new MGU is compatible with the old one, a copy of it is sent to the remote processor along the external link in order to keep the two copies consistent.

Cl.2 now becomes isolated in partition A, and is subject to migration. Since it has two links with partition B and one link with partition C, we migrate it to partition B to save the potential communication overhead. Figure 5 also shows the arrival of Cl.10 from partition A, which is thus available for resolution in partition C.

Here again, we assume that the migration of Cl.2 can be performed in approximately the same time one inference step is performed in partition B and C. Figure 6 displays the situation after these operations. Cl.7, in this case, must be migrated. Miscellaneous inference steps, thereafter, are shown in Figure 7 and Figure 8. In Figure 8, the goal statement is reached by the generation of an empty clause in partition B. Counting the migration process as one resolution step, it takes eight resolution steps to get the proof.

Presenting this example clause set to a logic inference system, a typical PROLOG, we get 28 inference steps. By carefully rearranging the order of these clauses, we can reduce it to 16 steps, and this turns out to be the best we can get from an uniprocessor inference system. Since most of the time inference problems involve a moderate number of clauses, optimal ordering of the clauses is usually not obtainable. On the other hand, the performance of the parallel model can be further improved if a more elaborated heuristic is encoded. The shorter resolution cycle in each step is another
speedup factor of the parallel model, that is implicitly illustrated in this example. Therefore, we conclude, that this parallel model will outperform uniprocessor inference systems, with the potential speedup factor in proportion to the number of partitions.

CONCLUDING REMARKS

We have described a parallel inference model for logic programming on general-purpose multicomputers. In this model, input clauses are partitioned into subsets, and resolution is conducted on each subset concurrently. All the available processors are fully utilized in this way. The partition is dynamically adjusted via clause migration as inference proceeds. This allows each processor to work on virtually the whole clause set while achieving a shorter resolution cycle. No synchronization is necessary between processors. In the context of AND/OR tree space search, the parallel model explores another dimension of parallelism in addition to the AND parallelism and OR parallelism. It implicitly allows multiple processors to jointly search the same path that leads to a refutation. Problem-solving heuristics can be incorporated in the parallel model systematically. They are encoded into a preference measure of unifiable literals to guide local inference such that the most promising clause is always resolved first. The preference measure is then translated into the distance measure between clauses. The distance measure allows closely related clauses to be grouped in the same subset using existing clustering algorithms. This partition so obtained has the best combination of clause subsets and a small rate of clause migrations. Clause migration decisions can be made based on the distance measure also. The optimal migration decision would be the one that realizes an adaptive clustering algorithm. Last, we find out that all the mechanisms of the parallel model can be efficiently supported by a connection graph. The graph also simplifies the implementation of the subsumption strategy.

Therefore, we conclude that this parallel model will outperform the uniprocessor inference systems.

REFERENCES
