A Survey of Parallel Data Mining.

Alex A. Freitas
CEFET-PR
Dep. de Informatica (DAINF)
Av. Sete de Setembro, 3165
Curitiba – PR, 80230-901, Brazil
alex@dainf.cefetpr.br

Abstract

With the fast, continuous increase in the number and size of databases, parallel data mining is a natural and cost-effective approach to tackle the problem of scalability in data mining. Recently there has been a considerable research on parallel data mining. However, most projects focus on the parallelization of a single kind of data mining algorithm/paradigm. This paper surveys parallel data mining with a broader perspective. More precisely, we discuss the parallelization of data mining algorithms of four knowledge discovery paradigms, namely rule induction, instance-based learning, genetic algorithms and neural networks. Using the lessons learned from this discussion, we also derive a set of heuristic principles for designing efficient parallel data mining algorithms.

1 Introduction.

A major issue in data mining is scalability with respect to the very large size of current-generation and next-generation databases, given the excessively long processing time taken by (sequential) data mining algorithms on realistic volumes of data. To mention just two examples, [Cohen 95] estimates that C4.5 with rule pruning would take 79 years on a 150-MHz processor in order to mine a database with 500,000 tuples. [Provost & Aronis 96] report that a sequential version of the RL algorithm is impractical (i.e. takes too long to run) on databases of more than 70,000 tuples.

Parallelism offers a natural and promising approach to cope with the problem of efficient data mining in large databases. Recently, there has been considerable interest in the parallel processing of data mining algorithms [Provost & Aronis 96], [Holsheimer et al. 96], [Shafer et al. 96], [Srivastava et al. 97], [Agrawal & Shafer 96], [Han et al. 97], [Kufrin 97], [Mackin 97], [McLaren et al. 97], [Neri & Giordana 95], [Flockhart & Radcliffe 95], [Freitas 97]. However, most of the literature focuses on isolated efforts to parallelize a single kind of data mining algorithm.

This survey aims at presenting a much broader view of the area of parallel data mining, by discussing the parallelization of several kinds of data mining algorithms/paradigms. More precisely, we discuss the parallelization of data mining algorithms in the following four knowledge discovery paradigms: rule induction, instance-based learning (or nearest neighbour), genetic algorithms and neural networks. In addition, we use the lessons learned from this discussion to derive some heuristic principles for designing efficient parallel data mining algorithms.

It should be noted that parallelism is not the only approach to speed up data mining algorithms. Other approaches – some of which can be used together with parallelism – are sampling, attribute selection, discretization of continuous attributes, restriction of the search space, algorithm/code optimization and distributed data mining. An overview of these approaches and their advantages/disadvantages over parallelism can be found in [Freitas &

Lavington 98], which also presents a detailed discussion on parallel data mining.

This paper is organized as follows. Section 2 reviews the distinction between data parallelism and control parallelism. Section 3 discusses parallel rule induction. Section 4 discusses parallel instance-based learning. Section 5 discusses parallel genetic algorithms. Section 6 discusses parallel neural networks. Section 7 discusses how to design efficient parallel data mining algorithms and points out some future research directions.

2 Data parallelism versus control parallelism.

This Section reviews the distinction between data parallelism and control parallelism, which is crucial for an understanding of the next Sections. In essence, data parallelism refers to the execution of the same operation or instruction on multiple large data subsets at the same time [Hillis & Steele 86], [Lewis 91], as illustrated in Figure 1. This is in contrast to control parallelism (or operation parallelism), which refers to the concurrent execution of multiple operations or instructions, as illustrated in Figure 2.

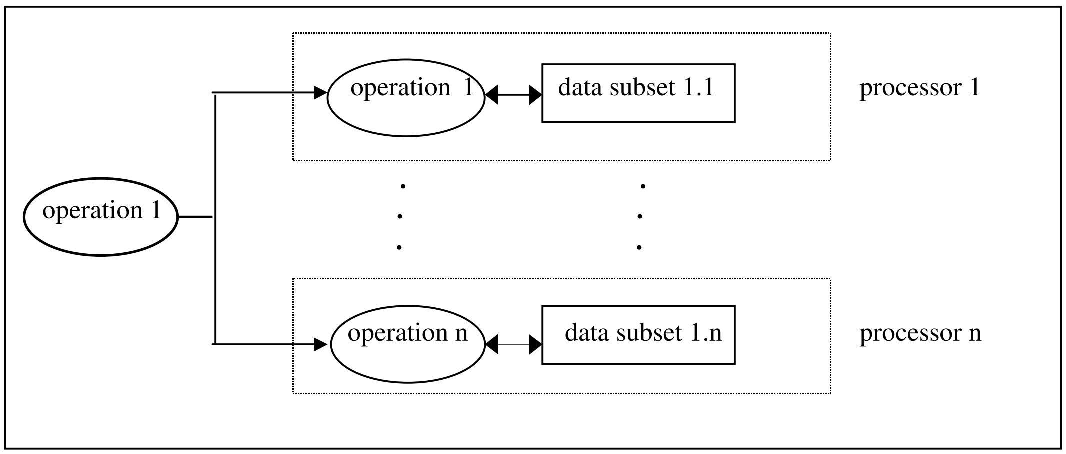

Figure 1: Data parallelism (1 operation executed on n processors, via data partitioning).

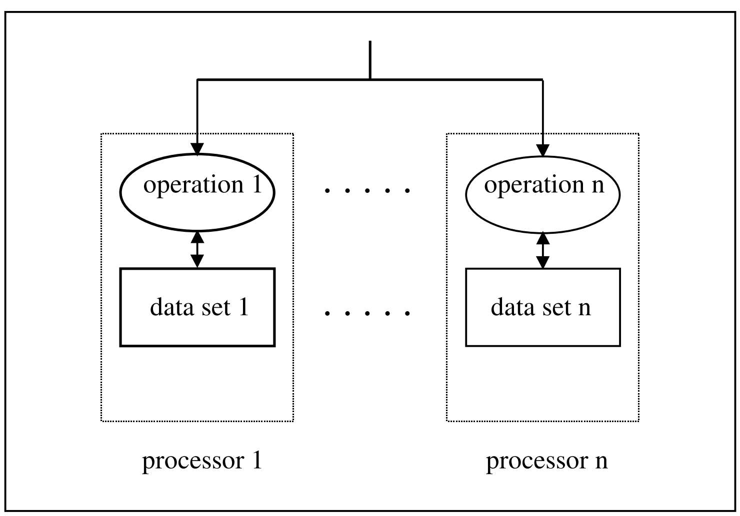

Figure 2: Control parallelism (n operations executed on n processors).

From a data mining viewpoint, data parallelism has three main advantages over control parallelism. First, data parallelism lends itself to a kind of automatic parallelization. The control flow of a data parallel program is essentially the same as the control flow of a sequential program – only the access to the data is parallelized in the former. Hence, a lot of previously-written sequential code can be re-used in a data-parallel fashion. This simplifies programming and leads to a development time significantly smaller than the one associated with control-parallel programming.

Second, data parallelism has a higher degree of machine-architecture independence, in comparison with control parallelism. Since the control flow of a data-parallel algorithm is still sequential (recall that only data handling is parallelized), there is no need to tailor the control flow of the algorithm to the underlying parallel architecture. This is in contrast with control parallelism, where this kind of tailoring is one of the major challenges of parallel programming [Quinn 87], [Akl 89]. Note that the problem of machine-architecture dependence is not completely eliminated in data parallelism. This problem is simply pushed down to a lower layer of software, hidden from the applications programmer, which leads to an increase in programmer productivity.

Third, intuitively data parallelism has better scalability for large databases than control parallelism [Hillis & Steele 86]. In most database applications (including data mining), the amount of data can increase arbitrarily fast, while the number of lines of code typically increases at a much slower rate. To put it in simple terms, the more data is available, the more the opportunity to exploit data parallelism. In principle we can add to the system a number of processor nodes (usually CPU + RAM) proportional to the amount of data increase, to keep the response time nearly constant. (In practice there will be some small increase in response time, due to the increase in inter-processor communication overhead associated with the extra number of processor nodes.)

Despite the above advantages of data parallelism, it should be emphasized that the exploitation of control parallelism is also useful in data mining. For instance, in the rule induction paradigm (see Section 3), a pure data-parallel approach would search the rule space in a sequential fashion, evaluating/modifying candidate rules one at a time. Hence, data parallelism does not address the problem of very large rule spaces. This problem is better dealt with by using control parallelism.

To summarize, one can say that data parallelism addresses the problem of very large databases (typically very many tuples), whereas control parallelism addresses the problem of very large search spaces (e.g. very many candidate rules). Note that data and control parallelism are not mutually exclusive. If a large enough number of processors is available, both types of parallelism can be exploited at the same time, which can greatly speed up the execution of data mining algorithms.

3 Parallel Rule Induction.

In a very high level of abstraction a Rule Induction (RI) algorithm can be viewed as an iterative heuristic search, where each iteration consists of three steps, as follows:
(1) select the “best” candidate rule (CR);
(2) expand the selected CR, generating new CRs;
(3) evaluate the new CRs.

These iterative steps are repeated until a satisfactory set of CRs is found [Holsheimer & Siebes 94], [Michalski 83]. Both steps (1) and (3) are based on a given CR-evaluation function, which computes the quality of a CR by accessing the database. Although the exact formula used to evaluate a CR varies a great deal among different data mining algorithms, most rule induction algorithms use a CR-evaluation function based on the *number of tuples* satisfying the CR’s antecedent (“if” part) and consequent (“then” part). In essence, *counting* database tuples satisfying a given set of conditions is a ubiquitous operation in CR-evaluation functions [Freitas 97]. Therefore, data parallelism can be naturally applied to the above step (3) of a RI algorithm, where a single CR is evaluated in parallel by multiple processors (each of them counting tuples in its local memory). This approach is particularly appealing, because the evaluation of CRs is the bottleneck of a RI algorithm mining a very large database.

In contrast, control parallelism is usually associated with the above steps (1) and (2). In addition, in the case of step (3), control parallelism can also be associated with the evaluation of multiple CRs at the same time, on different processors.

There is, however, an interesting phenomenon related to the scalability of control parallelism. Surprisingly, in some cases increasing the amount of search (creating more opportunity to exploit control parallelism) leads to *less* accurate discovered knowledge - see e.g. [Quinlan & Cameron-Jones 95a], [Murphy & Pazzani 94] and [Murthy & Salzberg 95]. In general, this anomaly does not occur in the case of data parallelism. Indeed, for a given number of attributes, increasing the number of tuples given to the data mining algorithm tends to increase (or at least not to reduce) the accuracy of the discovered knowledge.

We mention below several parallel Rule Induction (RI) systems, focusing on the differences between data and control parallelism. The remaining of this Section is divided into two Subsections. The first one discusses parallel RI systems running on general-purpose parallel machines, without DBMS facilities; while the second one discusses parallel RI systems running on parallel database servers, with DBMS facilities.

3.1 Parallel Rule Induction (RI) Systems without DBMS Facilities.

[Agrawal & Shafer 96] report experiments with parallel versions of the Apriori algorithm to discover association rules. In a data-parallel version, each processor independently computes local support counts for all the candidate itemsets of a given size by accessing only database tuples in its local memory. In contrast, in a control-parallel version, each processor computes support counts for its local candidate itemsets by accessing all tuples (in all processors' local memories). They observed that the data-parallel version achieved speed ups significantly greater than the control-parallel version.

More recently, [Han et al. 97] have proposed new parallel algorithms for mining association rules which improve upon some aspects of the parallel algorithms proposed by [Agrawal & Shafer 96]. In particular, some improvements proposed by [Han et al. 97] include a more efficient inter-processor communication scheme and a clever method for distributing candidate itemsets among processors.

[Anand et al. 95a], [Anand et al. 95b], [Anand et al. 94a] have developed the parallel rule induction algorithm STRIP, based on evidence theory. STRIP was designed to allow the exploitation of both data- and control-parallelism, discovering mainly summarization and association rules. Some experiments have shown that the exploitation of control parallelism can be very inefficient in some cases [Anand et al. 95b], but beneficial in others [Anand et al. 95a]. Overall, however, the exploitation of data parallelism seems to be safer and more promising in STRIP, when mining large databases.

[Provost & Aronis 96] have implemented a data-parallel version of the RL algorithm on a Connection Machine CM-2 with 8k processors. They achieved a speed up of about 1200 over sequential RL on a data set with 1,000,000 tuples.

[Cook & Holder 90] implemented a control-parallel version of the AQ algorithm on the CM-2. However, in their approach each Candidate Rule (CR) is stored in a distinct processor, which requires a number of processors exponential in the number of attributes - a not very cost-effective approach. In addition, all the training tuples have to be replicated in each processor, which considerably reduces the scalability of the algorithm.

[Kufrin 97] discusses the parallelization of several phases of the rule pruning method of

C4.5. For each phase, he discusses which approach (data parallelism or control parallelism) is more appropriate. In essence, control parallelism was used to select a subset of rules, while data parallelism was used in the phases of pruning rule conditions and removing individual rules from the rule set. The author reports good speed up results on a shared-memory parallel machine.

There has also been significant research about parallel versions of Top-Down Induction of Decision Trees (TDIDT) algorithms. We mention below some projects in this area.

[Pearson 94] implemented two versions of a parallel TDIDT algorithm on a 128-processor Fujitsu cellular array computer, both of them exploiting control parallelism. Overall, the speed ups achieved were relatively low (far smaller than 128, the number of processors). Furthermore, in both parallel versions all the tuples are replicated in each processor, which of course reduces the scalability of the system.

SPRINT is a TDIDT algorithm designed specifically for parallel execution, for which very good speed up and scalability results are reported [Shafer et al. 96]. However, this algorithm has a significantly increased memory usage, in order to store some special data structures. These data structures include lists of predicting-attribute values. These lists keep the attribute values in order, avoid the resorting of attribute values after the partitioning of a tree node. The tuples being mined are distributed across the processors, and each processor computes attribute lists referring only to its local tuples. When a tree node is partitioned, not only the tuples in that node but also the attribute lists are partitioned across the children nodes. Unfortunately, the partitioning of the attribute lists is performed by creating a hash table that requires that tuple ids of tuples in the current tree node be collected from all processors. Hence, the number of collected tuple ids is proportional to the number of tuples being mined, which limits scalability.

More recently, [Srivastava et al. 97] have proposed a new scalable TDIDT algorithm that avoids the specialized data structures used in SPRINT by performing clustering on the values of a continuous attribute, rather than sorting these values. The algorithm follows a hybrid data/control-parallel approach, by exploiting data parallelism in higher levels of the tree and control parallelism in lower levels of the tree (where there is less data to justify the exploitation of data parallelism).

Finally, in addition to the above domain-independent parallel algorithms, several domain-specific parallel classification algorithms, somehow relying on specific properties of their target application domain, have been reported in the literature. Some representative examples are in the domains of protein-folding [Lathrop et al. 90], [Hofacker et al. 96] and halo finding in cosmology [Pfitzner & Salmon 96].

3.2 Parallel Rule Induction (RI) Systems with DBMS Facilities.

The previous Subsection has discussed parallel rule induction in general-purpose parallel machines. However, nowadays there is a number of high-performance, cost-effective, commercially-available Parallel Database Servers (PDS). These machines offer the benefit of automatic parallelization, in that database queries are automatically parallelized and optimized by the machine. The application programmer only has to specify the query in a declarative style, saying “what” the query must do, rather than “how” to do it. Hence, the user has the benefit of reduced processing time without the complexity of parallel programming.

In addition, PDS offer several DBMS facilities that are also useful for large-scale data mining systems – e.g. a DBMS’ security-control mechanisms help to promote data-privacy control, which is an important issue in data mining [O’Leare 95]. Hence, it is desirable to exploit the DBMS facilities offered by state-of-the-art PDS. We mention below some projects that exploit such facilities.

The DAFS project includes a parallel version of a TDIDT algorithm [McLaren et al. 97]. DAFS is a PDS designed to support the overall knowledge discovery process (including pre-processing and post-processing for data mining). It follows a client/server approach, where the client is based on the well-known Clementine data mining tool [Shearer & Khabaza 96] and the server is a shared-nothing PDS. The parallel TDIDT algorithm implemented in DAFS essentially follows a data-parallel approach, which seems to be an adaptation of the data-parallel algorithm described in [Kufrin 95].

[Holsheimer et al. 96], [Holsheimer & Kersten 94] implemented a data-parallel version of a beam-search data mining algorithm on the Monet PDS. In essence, candidate rules (CRs) are evaluated by submitting queries to Monet, which processes the queries by exploiting *inter*-query parallelism – i.e. two or more queries are processed in parallel by multiple processors. Monet uses *vertical* partitioning of database relations (i.e. the set of *attributes* of a relation are distributed across the processors). This approach departures from conventional PDS, which use *horizontal* partitioning (i.e. the set of *tuples* of a relation are distributed across the processors).

One motivation to use more conventional PDS (typically based on horizontal partitioning) stems from the desire to integrate data mining and data warehouses [Pass 97], [Freitas & Lavington 98]. Indeed, most large data warehouses are implemented on the top of commercially-available PDS [IBC 95], to improve efficiency and scalability. [Freitas 97], [Freitas & Lavington 96] propose a primitive-based approach for integrating data mining and data warehouses. More precisely, this project defines a generic rule induction primitive that underpins the evaluation of CRs in a number of rule induction algorithms. By executing this primitive on a PDS, rule induction algorithms are significantly speeded up due to the exploitation of data parallelism in the execution of database queries. To support this claim, the authors report the results of some experiments with the application of the primitive in a data-parallel TDIDT algorithm.

There are also some data mining tools offering parallel implementations of TDIDT algorithms. One example is the HeatSeeker system [Mackin 97]. In this system the client offers a graphical interface used to control the TDIDT algorithm and the parallel database server is a distributed-memory White Cross machine. On the largest, high-end White Cross machines, a large number of processors can be used for exploiting data parallelism in a massively-parallel fashion.

Finally, [Anand et al. 94b] have used a hardware accelerator, namely the Ingres Search Accelerator (based on ICL's SCAFS), to speed up database queries. They observed that, for a given database size, the benefits associated with the reduction of processing time are inversely proportional to the number of tuples satisfying the query. Note that this is in contrast with conventional "software-based" PDS, where the benefits of using a parallel machine are proportional to the number of tuples satisfying the query.

4 Parallel Instance-Based Learning.

In the context of a classification task, the instance-based learning paradigm consists of two basic steps, as follows:

(1) compare a new tuple, to be classified, against all stored tuples, by computing a distance metric (similarity measure) between the new tuple and each stored tuple;

(2) the k nearest (most similar) stored tuples – where k is a user-specified value – are selected and their classes are used to predict the class of the new tuple according to a given class-conflict resolution scheme (e.g. pick the most frequent class among the nearest stored tuples).

The execution time of Step (2) is quite small and is entirely dominated by step (1) –

assuming that k << number of tuples, as usual. Hence, the target of parallelism is step (1). This step seems to offer an ideal opportunity for the exploitation of data parallelism, as follows. First of all, the data being mined is partitioned into p mutually exclusive and exhaustive data subsets, where p is the number of processors. Each subset is assigned to a distinct processor. Then each processor computes the distance (similarity) between the tuples in its local data subset and the new tuple to be classified. Note that each processor can perform its computation in a manner entirely independently from the other processors. Hence, inter-processor communication overhead is not a concern in the first step of the algorithm. (The second step requires some inter-processor communication overhead, but recall that the time taken by this step is much shorter than the time taken by the first step.)

The exploitation of data parallelism in IBL can also be done in a way that integrates data mining and data warehouses, based on the idea of performing generic data mining primitives on a Parallel Database Server (PDS) – see Section 3.2. [Freitas 97], [Freitas 97a] has defined a generic IBL primitive that underpins several IBL algorithms and has shown how to use that primitive to exploit data parallelism on a PDS, in order to significantly reduce the processing time of IBL algorithms.

The exploitation of control parallelism in instance-based learning is also possible – although often less profitable than data parallelism. In the control-parallel approach each processor is assigned the task of classifying a subset of new tuples. Each processor has access to a copy of all stored tuples. This allows each processor to classify each of its new tuples without communicating with other processors. However, this approach has reduced scalability, since the entire data being mined must be replicated across all processors. Hence, this approach is suitable when there are a large number of new tuples to be classified and the number of stored tuples is not very high.

So far we have discussed how to exploit parallelism in the computation of distance metrics in general, without considering any detail of the distance metric. In practice, depending on the IBL algorithm, the computation of a distance metric can require some operation that offers additional potential for the exploitation of parallelism. The typical example is the computation of attribute weights, which are used in many IBL algorithms to assign greater importance to attributes with greater predictive power.

Two IBL algorithms that parallelize the computation of attribute weights are discussed in [Stanfill & Waltz 86] and [Creecy et al. 92]. In both algorithms attribute weights are dynamically computed in parallel as the algorithm computes the distance between a new tuple and the stored tuples. In both cases data parallelism is exploited, as described above. Once the stored tuples are distributed across the processors, the computation of attribute weights is done in parallel by letting each processor compute partial attribute weight values from its local tuples and then by having the partial results somehow combined into the final attribute weights. Furthermore, both algorithms were implemented on the Connection Machine CM-2, by exploiting massive parallelism.

5 Parallel Genetic Algorithms.

A genetic algorithm (GA) is an iterative procedure that maintains a population of "individuals", which are strings of symbols representing a candidate solution for a given problem. At each iteration (or "generation") the current individuals are evaluated by a fitness function, which measures the quality of the candidate solution represented by the individual, and genetic operators are applied to the fittest individuals of the current generation, modifying them and creating a new generation of individuals [Goldberg 89], [Michalewicz 96]. Due to Darwin's principle of natural selection (survival of the fittest), the population tends to converge to highly-fit individuals (high-quality solutions).

In the context of data mining, individuals often represent candidate rules and the fitness function measures the quality of these rules. Note that the fitness function has a role equivalent to the candidate rule-evaluation function in the rule induction paradigm. The main difference between the genetic algorithms and the rule induction paradigms is in the method used for traversing the rule space.

Similarly to rule induction, there are two basic sources of parallelism in genetic algorithms. One can exploit parallelism in the computation of the fitness of individuals and/or in the application of genetic operators. Unless the data being mined is small, the time spent with the computation of individuals' fitness (which involves access to the data being mined) tends to greatly exceed the time spent with the application of genetic operators (which are typically computationally cheap). For this reason, in the remaining of this Section we focus on the parallelization of fitness computation.

There are two basic ways of parallelizing fitness computation. The first consists of exploiting parallelism in the computation of the fitness function of each individual. This is a data-parallel approach, as follows. First of all, the data being mined is partitioned into p mutually exclusive and exhaustive data subsets, where p is the number of processors. Each subset is assigned to a distinct processor. Then each processor computes a partial value for the fitness of an individual by accessing only its local tuples. Next, the partial fitness values are somehow combined to produce the final fitness value for the current individual.

The second way of parallelizing fitness computation is a control-parallel approach. It consists of partitioning the set of individuals of the current population (rather than the data being mined) into p mutually exclusive and exhaustive subsets. Each subset is assigned to a distinct processor. Then each processor computes the fitness function for all its individuals. Note that the computation performed by each processor is independent from the computation performed by other processors. However, the entire data being mined has to be replicated across all processors, which reduces the scalability of the system.

Control-parallel genetic algorithms can be further divided into two broad approaches, namely single-population and distributed-population algorithms. In the former every individual can mate with any other individual, so that all individuals are logically regarded as a single population (even though they are physically distributed across all processors). As a result, in principle (ignoring the non-determinism associated with the application of genetic operators) this kind of control-parallel genetic algorithm discovers the same knowledge as its sequential counterpart.

On the other hand, a distributed-population genetic algorithm treats each of the physically distributed sub-populations as a distinct logical population, so that an individual can mate only with other individuals in the same sub-population. This approach can be further subdivided into coarse-grained and fine-grained approaches [Cantu-Paz 95], [Lin et al. 94]. In the former the population is divided into a small number of sub-populations, each of them with a large number of individuals; whereas in the latter the population is divided into a large number of sub-populations, each of them with a small number of individuals. In both approaches there is some mechanism to allow the exchange of some individuals among the sub-populations from time to time. (Otherwise, the process would be equivalent to run p distinct sequential genetic algorithms in parallel, where p is the number of sub-populations, rather than parallelizing a single run of a genetic algorithm.)

Note that the distributed-population approach represents a significant departure from a single-population approach, once these two approaches lead to the discovery of different pieces of knowledge (different rule sets). Actually, the modification introduced by distributed-population GAs is often considered as advantageous [East & Rowe 96] - one of the reasons being the fact that a distributed population reduces the probability of premature convergence to suboptimal solutions, in comparison with a single population.

An example of a fine-grain, distributed-population GA for data mining is GA-MINER [Flockhart & Radcliffe 95], [Flockhart & Radcliffe 96]. This system is a parallel GA designed to search for several kinds of knowledge, such as "if-then" prediction rules, distribution-comparison patterns within distinct sets of tuples (e.g. the difference in the mean of an attribute's values) and statistical correlation patterns. GA-MINER has been implemented in both shared-memory and distributed-memory parallel machines [Flockhart & Radcliffe 95]. However, in the latter the entire data being mined had to be replicated into the local memory of each processor. This obviously reduces scalability for large databases.

An example of a coarse-grain, distributed-population GA for data mining is REGAL. This is a parallel algorithm in which each individual encodes a candidate rule, so that the whole population corresponds to a rule set [Giordana & Neri 95], [Neri & Giordana 95], [Neri & Saitta 96]. The algorithm was mainly designed to discover classification rules. Experiments have been done with REGAL on a Connection Machine CM-5, with a number of processors varying between 16 and 64. Overall, the authors have reported a superlinear speed up.

A variant of REGAL, called G-NET, has been implemented in a network of workstations via PVM [Anglano et al. 97]. Experiments have been done with the number of workstation varying between 1 and 9. Overall, the speed ups were sublinear. However, in both the above experiments (with REGAL and G-NET) the data being mined was relatively small, so that it is not clear how much speed up can be achieved when mining large databases.

Finally, the exploitation of data parallelism in GA can also be done in a way that integrates data mining and data warehouses, based on the idea of performing generic data mining primitives on a Parallel Database Server (PDS) – see Section 3.2. [Freitas 97b] describes a preliminary work based on this idea, where the time-consuming operations associated with the evaluation of genetic programming individuals are encapsulated in a database query, which can then be executed on a PDS to reduce processing time.

6 Parallel Neural Networks.

In essence, a Neural Network (NN) consists of many Processing Elements (PEs), loosely called "neurons", and weighted interconnections among the PEs. Each PE performs a very simple computation, such as calculating a weighted sum of its input connections, and computes an output signal that is sent to other PEs. The training (mining) phase of a NN consists of adjusting the weights (typically, real-valued numbers) of the interconnections, in order to produce the desired output [Rumelhart & McClelland 86], [Rojas 96].

Note that the "knowledge" of the system is expressed in a low-level representation, distributed across the weights of the interconnections. In passing we note that, in the context of data mining, it is often desirable to convert the learned interconnection weights into a higher-level knowledge representation such as "if-then" rules, to make the discovered knowledge comprehensible for the user [Lu et al. 95], [Vaughn 96]. However, this issue is beyond the scope of this paper. Here we are interested in the influence that the distributed representation of NNs has in the parallelization of NN algorithms.

There are two basic approaches for exploiting parallelism in NNs. The first consists of distributing the data being mined across the processors [Foo et al. 97], [Fathy & Syiam 96]. This is a data-parallel approach, also called training set-parallel, in the terminology of parallel NNs. First of all, the data being mined is partitioned into p mutually exclusive and exhaustive subsets, where p is the number of processors. Each partition is assigned to a distinct processor. Each processor has a complete copy of the NN, with all its neurons (PEs) and all its interconnection weights. Hence, each processor uses its local data subset to compute partial weight updates for its local copy of the entire NN. Then these partial weight updates are somehow combined to produce the final weight updates for the entire NN.

It should be noted that this approach usually involves a form of *batch* weight updating, where the interconnection weights are actually updated only after the combination of the partial weight updates. This is in contrast with the *incremental* nature of conventional weight-update procedures, where weights are usually updated right after the NN processes each training tuple. The batch-updating procedure often leads to a prediction accuracy somewhat different from (either higher or lower than) the one achieved with a conventional incremental-updating procedure.

The second approach for exploiting parallelism in NNs consists of distributing the structure (neurons and interconnections) of the NN among the processors. This approach is based on the inherent control parallelism associated with NNs. The strategy used for distributing the structure of the NN across the processors depends on several factors, such as the number of neurons in each layer of the NN, the number of layers, the number of available processors, the topology of the interconnections, etc.

In one strategy, each processor is assigned a distinct neuron and the set of weights arriving into its neuron. In this case the processing performed by each neuron is entirely independent of the processing performed by other neurons, so that all neurons can operate in parallel. Of course, if the number of neurons is considerably larger than the number of processors, this strategy can be modified so that each processor is assigned a subset of neurons with their corresponding incoming interconnections.

Care must be taken, however, to avoid that many processors get idle at a given time. For instance, assume that the neurons and their incoming interconnections are distributed across the processors in such a way that each processor is assigned only neurons of the same layer of the NN. In this case, while the processors assigned to one layer of the NN are working, the processors assigned to other layers would be idle. This can be avoided by exploiting a form of "temporal parallelism" (or pipelining) across successive layers of the NN. For instance, as soon as a neuron of the layer L_k of the NN finishes its processing of a tuple t_n, that neuron starts to process the next tuple t_{n+1}, while at the same time a neuron of the next layer L_{k+1} starts to process tuple t_n.

Another strategy for exploiting control parallelism in NNs, working in a lower level of granularity than the above strategy, consists of exploiting parallelism within each neuron - see e.g. [Blelloch & Rosenberg 87], [Nordstrom & Svensson 92]. In this strategy, the incoming interconnections of each neuron (with their respective weights) are distributed across a set of processors. Due to its fine-grain nature, this strategy is usually suitable for massively-parallel processing systems, particularly when the number of processors is much larger than the number of neurons.

Although there has been an extensive research on parallel NNs, there has been little research on the area of parallel NNs for data mining applications. An exception is the DAFS project (also mentioned in Section 3.2), which offers parallel implementations of Clementine's Neural Networks on a Parallel Database Server [McLaren et al. 97]. These implementations essentially follow a data-parallel approach.

7 Principles for Designing Efficient Parallel Data Mining Algorithms and Future Research.

We have discussed the parallelization of data mining algorithms of four knowledge discovery paradigms, namely rule induction, instance-based learning, genetic algorithms and neural networks. Despite significant differences between these paradigms, our discussion has focused on an aspect of parallelization whose importance cuts across all these paradigms. Such aspect is the distinction between data parallelism and control parallelism. We believe that this distinction is crucial for the design of parallel data mining algorithms.

Based on the lessons learned from the systems discussed in the previous Sections, as well as on fundamental parallel processing concepts, we can suggest a set of "heuristic principles" for designing efficient parallel data mining algorithms, as shown in Figure 3.

(1) Analyze the proportion of time spent in each part of the algorithm. The most time-consuming parts should be the target of parallel processing.
(2) For each target part of the algorithm, consider whether it is data-intensive or CPU-intensive, in order to opt for data or control parallelism.
(3) When considering data parallelism, try to think of intelligent data partitioning schemes, rather than random or round-robin ones. Data partitioning is a key issue in the exploitation of data parallelism.
(4) When considering control parallelism, try to minimize inter-processor communication. This is a major problem in the exploitation of control parallelism.
(5) When considering control parallelism, recall that it is often necessary to replicate the entire data being mined in every processor. Unless you can avoid this scheme, beware that the scalability of the system will be reduced.
(6) Consider the possibility of hybrid data/control-parallel algorithms, where the type of parallelism being exploited is dynamically switched based on current parameters of the algorithm and search space.

Figure 3: Heuristic principles for designing efficient parallel data mining algorithms.

Principles (1), (3) and (4) are fundamental in the design of any parallel algorithm, while principles (2), (5) and (6) are more related to parallel data mining. Let us elaborate a little more on the latter three principles.

Principle (2) simply says that data-intensive operations should be tackled with data parallelism, while CPU-intensive operations should be tackled with control parallelism. Hence, in the same algorithm, some parts of the algorithm can exploit data parallelism while other parts exploit control parallelism. A good example of the application of this principle was the parallelization of the rule pruning method of C4.5, mentioned in Section 3.1.

Principle (5) is actually a warning about the problem of scalability of control parallelism in data mining. Note that this problem is not very serious in some cases. For instance, control parallelism can be easily exploited in neural networks without replicating the data being mined, since (conventional) neural networks are trained one-tuple-at-a-time. However, the warning associated with this principle is important in many cases where the data mining algorithm is based on the paradigms of rule induction, instance-based learning and genetic algorithms.

Principle (6) suggests the combination of "the best of both worlds". A good example is the parallelization of TDIDT algorithms. In higher levels of the tree (close to the root), the algorithm has to access a large amount of data, which calls for data parallelism. As the tree is expanded, however, the amount of data to be accessed gets smaller and smaller. Hence, in lower levels of the tree, control parallelism tends to be more advantageous than data parallelism.

We emphasize that the above principles are just heuristics – i.e. they are not guaranteed to lead to the best parallel data mining algorithm in all cases, but they work reasonably well in many cases.

We now turn to the topic of future research. With the fast, continuous improvement of interconnection networks, clusters of workstations and/or networks of computers can be easily connected to form larger computer networks. This increases the opportunity for distributed processing. Furthermore, advances in data communications technology blur the distinction between distributed processing and parallel processing. There are already software environments, such as PVM, that allow a cluster of computers to be viewed as a single parallel machine. These factors suggest that the development of parallel data mining algorithms for heterogeneous environments (where the processors have different speeds and memory capacities) will be an important area for future research. Mining the wealth of information available on the Internet in parallel is also an area deserving further research.

In addition, it would be interesting to perform a massive comparison of different parallel data mining algorithms, similarly in spirit to the massive comparison of (sequential) classification algorithms performed in the well-known STATLOG project [Michie et al. 94].

References.

[Agrawal & Shafer 96] R. Agrawal and J.C. Shafer. Parallel mining of association rules: design, implementation and experience. *IBM Research Report RJ 10004*. 1996. To appear in *IEEE Trans. Knowledge & Data Engineering.*

[Akl 89] S.G. Akl. *The Design and Analysis of Parallel Algorithms.* Prentice-Hall, 1989.

[Anand et al. 94a] S.S. Anand, D.A. Bell and J.G. Hughes. Parallelising the discovery of strong rules from databases. *Internal Report*. Facult of Informatics, University of Ulster (Jordanstown). Nov. 1994.

[Anand et al. 94b] S.S. Anand, D.A. Bell and J.G. Hughes. An empirical performance study of the Ingres Search Accelerator for a large property management database system. *Proc. 20th Very Large Databases (VLDB-94) Conf.*, 676-685. Santiago, Chile, 1994.

[Anand et al. 95a] S.S. Anand, C.M. Shapcott, D.A. Bell and J.G. Hughes. Data mining in parallel. *Proc. World Occam and Transputer User Group Conf.* Manchester, April 1995.

[Anand et al. 95b] S.S. Anand, D.A. Bell, J.G. Hughes and C.M. Shapcott. Evidential techniques in parallel database mining. In: *High-Performance Computing and Networking Conf.*, 1995. LNCS-919, 190-195.

[Anglano et al. 97] C. Anglano, A. Giordana, G. Lo Bello and L. Saitta. A network genetic algorithm for concept learning. *Proc. 7th Int. Conf. Genetic Algorithms,* 434-441. 1997.

[Blelloch & Rosenberg 87] G. Blelloch and C.R. Rosenberg. Network learning on the Connection Machine. *Proc. 10th Int. J. Conf. on AI (IJCAI-87)*, 323-326, 1987.

[Cantu-Paz 95] E. Cantu-Paz. A summary of research on parallel genetic algorithms. *Illigal Report No. 95007.* University of Illinois at Urbana-Champaign. July 1995.

[Cohen 95] W.W. Cohen. Fast effective rule induction. *Proc. 12th Int. Conf. Machine Learning (ML-95)*, 115-123. 1995.

[Cook & Holder 90] D.J. Cook and L.B. Holder. Accelerated learning on the connection machine. *Proc. 2nd IEEE Symp. Parallel and Distributed Processing*, 448-454. Dallas, Texas, Dec. 1990.

[Creecy et al. 92] R.H. Creecy, B.M. Masand, S.J. Smith and D.L. Waltz. Trading MIPS and memory for knowledge engineering. *Comm. of the ACM*, 35(8), 48-63, Aug./92.

[East & Rowe 96] I.R. East and J. Rowe. Effects of isolation in a distributed population genetic algorithm. *Proc. 4th Conf. Parallel Problem Solving from Nature, LNCS* 1141, 408-419. Sep. 1996.

[Fathy & Syiam 96] S.K. Fathy and M.M. Syiam. A parallel design and implementation for backpropagation neural network using MIMD architecture. *Proc. 1996 IEEE Int. Conf. Neural Networks*, *Vol. 2*, 1361-1366. Washington, DC, USA. June 1996.

[Flockhart & Radcliffe 95] I.W. Flockhart and N.J. Radcliffe. GA-MINER: parallel data mining with hierarchical genetic algorithms - final report. *EPCC-AIKMS-GA-MINER-Report 1.0*. University of Edinburgh, 1995.

[Flockhart & Radcliffe 96] I.W. Flockhart and N.J. Radcliffe. A genetic algorithm-based approach to data mining. *Proc. 2nd Int. Conf. Knowledge Discovery & Data Mining*, 299-302. 1996.

[Foo et al. 97] S.K. Foo, P. Saratchandran and N. Sundararajan. Parallel implementation of backpropagation neural networks on a heterogeneous array of transputers. *IEEE Trans. Systems, Man, and Cybern. - Part B: Cybern.*, 27(1), 118-126, Feb. 1997.

[Freitas 97] A.A. Freitas. *Generic, Set-Oriented Primitives to Support Data-Parallel Knowledge Discovery in Relational Databases.* Ph.D. thesis. University of Essex, UK, July 1997.

[Freitas 97a] A.A. Freitas. Towards large-scale knowledge discovery in databases (KDD) by exploiting parallelism in generic KDD primitives. *Proc. 3rd Int. Workshop on Next Generation Info. Technologies and Systems (NGITS'97),* 33-43. Neve Ilan, Israel, July 1997.

[Freitas 97b] A.A. Freitas. A genetic programming framework for two data mining tasks: classification and generalized rule induction. *Genetic Programming 1997: Proc. 2nd Annual Conf.,* 96-101. Morgan Kaufmann, 1997.

[Freitas & Lavington 96] A.A. Freitas and S.H. Lavington. Using SQL primitives and parallel DB servers to speed up knowledge discovery in large relational databases. In: R. Trappl. (Ed.) *Cybernetics and Systems'96: Proc. 13th European Meeting on Cybernetics and Systems Research*, 955-960. Vienna, 1996.

[Freitas & Lavington 98] A.A. Freitas & S.H. Lavington. *Mining Very Large Databases with Parallel Processing*. Kluwer Academic Publishers, 1998. ISBN 0-7923-8048-7.

[Giordana & Neri 95] A. Giordana and F. Neri. Search-intensive concept induction. *Evolutionary Computation 3(4)*: 375-416, Winter 1995.

[Goldberg 89]. D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[Han et al. 97] E-H. (S.) Han, G. Karypis and V. Kumar. Scalable parallel data mining for association rules. *Proc. 1997 ACM SIGMOD Int. Conf. Management of Data*, May 1997.

[Hillis & Steele 86] W.D. Hillis and L. Steele Jr. Data parallel algorithms. *Comm. ACM*, 29(12), Dec. 1986, 1170-1183.

[Hofacker et al. 96] I.L. Hofacker, M.A. Huynen, P.F. Stadler and P.E. Stolorz. Knowledge discovery in RNA sequence families of HIV using scalable computers. *Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining*, 20-25. AAAI Press, 1996.

[Holsheimer & Kersten 94] M. Holsheimer and M.L. Kersten. Architectural support for data mining. *Proc. AAAI-94 Workshop on Knowledge Discovery in Databases*, 217-228. AAAI Press, 1994.

[Holsheimer et al. 96] M. Holsheimer, M. Kersten and A. Siebes. Data surveyor: searching the nuggets in parallel. In: U.M. Fayyad et al. (Eds.) *Advances in Knowledge Discovery and Data Mining*, 447-467. AAAI Press, 1996.

[IBC 95] IBC Ltd. *Proc. Conf. Commercial Parallel Processing*. London, Nov./95. (IBC Technical Services Ltd., 57-61 Mortimer St., London, W1N 7TD.)

[Kufrin 95] R. Kufrin. Decision trees on parallel processors. *Proc. IJCAI-95 Workshop on Parallel Processing for Artificial Intelligence*. Aug. 1995.

[Kufrin 97] R. Kufrin. Generating C4.5 production rules in parallel. *Proc. 14th Nat. Conf. on Artif. Intel. (AAAI-97)*. AAAI Press, 1997.

[Lathrop et al. 90] R.H. Lathrop, T.F. Smith, T.A. Webster and P.H. Winston. ARIEL: a massively parallel symbolic learning assistant for protein structure and function. In: *Artificial Intelligence at MIT: Expanding Frontiers*. Vol. 1, 70-103. Cambridge, MA: MIT Press, 1990.

[Lewis 91] T.G. Lewis. Data parallel computing: an alternative for the 1990s. *IEEE Computer*, 24(9), Sep. 1991, 110-111.

[Lin et al. 94] S.-C. Lin, W.F. Punch III and E.D. Goodman. Coarse-grain parallel genetic algorithms: categorization and new approach. *Proc. 6th IEEE Symp. Parallel and Distributed Processing*, 28-37. Oct. 1994.

[Lu et al. 95] H. Lu, R. Setiono and H. Liu. NeuroRule: a connectionist approach to Data Mining. *Proc. 21st Very Large Databases Conf. (VLDB-95)*. Zurich, 1995.

[Mackin 97] N. Mackin. Application of WhiteCross MPP servers to data mining. *Proc. 1st Int. Conf. Practical Applications of Knowledge Discovery and Data Mining (PADD'97)*, 1-8. The Practical Application Company, UK. Apr. 1997.

[McLaren et al. 97] I. McLaren, E. Babb and J. Bocca. DAFS: supporting the knowledge discovery process. *Proc. 1st Int. Conf. Practical Applications of Knowledge Discovery*, 179-190. The Practical Application Company, UK. Apr. 1997.

[Michalewicz 96] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* 3rd Ed. Springer-Verlag, 1996.

[Michalski 83] R.W. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, Vol. 20, 1983, 111-161.

[Michie et al. 94] D. Michie, D.J. Spiegelhalter and C.C. Taylor. *Machine Learning, Neural and Statistical Classification*. New York: Ellis Horwood, 1994.

[Murphy & Pazzani 94] P.M. Murphy and M.J. Pazzani. Exploring the decision forest: an empirical investigation of Occam's razor in decision tree induction. *Journal of Artificial Intelligence Research,* Vol. 1, 1994, 257-275.

[Murthy & Salzberg 95] S. Murthy and S. Salzberg. Lookahead and pathology in decision tree induction. *Proc.*

14th Int. Joint Conf. AI (IJCAI-95), 1025-1031. 1995.

[Neri & Giordana 95] F. Neri and A. Giordana. A parallel genetic algorithm for concept learning. *Proc. 6th Int. Conf. Genetic Algorithms*, 436-443. 1995.

[Neri & Saitta 96] F. Neri and L. Saitta. Exploring the power of genetic search in learning symbolic classifiers. *IEEE Trans. Pattern Analysis and Machine Intel.*, 18(11), 1135-1141, Nov. 1996.

[Nordstron & Svensson 92] R. Nordstron and B. Svensson. Designing and using massively parallel computers for artificial neural networks. *J. of Parallel and Distrib. Computing*, 14(3), Mar. 1992, 260-285.

[O'Leary 95] D.E. O'Leary. Some privacy issues in knowledge discovery: the OECD Personal Privacy Guidelines. *IEEE Expert*, 10(2), Apr./95, 48-52.

[Pass 97] S. Pass. Data mining: the power of integration. *Proc. 1st Int. Conf. Practical Applications of Knowledge Discovery and Data Mining (PADD'97)*, 25-39. The Practical Application Company, UK. Apr. 1997.

[Pearson 94] R.A. Pearson. A coarse-grained parallel induction heuristic. In: H. Kitano, V. Kumar and C.B. Suttner. (Ed.) *Parallel Processing for Artificial Intelligence 2*, 207-226. Elsevier Science, 1994.

[Pfitzner & Salmon 96] D.W. Pfitzner and J.K. Salmon. Parallel halo finding in N-body cosmology simulations. *Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining*, 26-31. AAAI Press, 1996.

[Provost & Aronis 96] F.J. Provost and J.M. Aronis. Scaling up inductive learning with massive parallelism. *Machine Learning* 23(1), Apr./96, 33-46.

[Quinlan & Cameron-Jones 95a] J.R. Quinlan and R.M. Cameron-Jones. Oversearching and layered search in empirical learning. *Proc. 14th Int. Joint Conf. AI (IJCAI-95)*, 1019-1024. 1995.

[Quinn 87] Quinn. *Designing Efficient Algorithms for Parallel Supercomputers*. New York: McGraw-Hill, 1987.

[Rojas 96] R. Rojas. *Neural Networks: a systematic introduction.*. Springer-Verlag, 1996.

[Rumelhart & McClelland 86] D. Rumelhart and McClelland. (Eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Cambridge, MA: MIT Press, 1986.

[Shafer et al. 96] J. Shafer, R. Agrawal and M. Mehta. SPRINT: a scalable parallel classifier for data mining. *Proc. 22nd Int. Conf. Very Large Databases (VLDB-96)*. Bombay, India, 1996.

[Shearer & Khabaza 96] C. Shearer and T. Khabaza. Data mining for data owners: presenting advanced technology to non-technologists through Clementine integrated toolkit. *Proc. Data Mining'96 Unicom Seminar*, 236-242. London: Unicom, 1996.

[Srivastava et al. 97] A. Srivastava, V. Singh, E-H. (S.) Han and Kumar. An efficient, scalable, parallel classifier for data mining. *Technical Report TR-97-010*. University of Minnesota, Dept. of Comp. Sci., 1997.

[Stanfill & Waltz 86] C. Stanfill and D. Waltz. Toward Memory-Based Reasoning. *Comm. ACM,* 29(12), Dec. 1986, 1213-1228.

[Vaughn 96] M.L. Vaughn. Interpretation and knowledge discovery from the multilayer perceptron network: opening the black box. *Neural Comput. & Applic. (1996)4:*72-82.