Large Scale Data Mining: Challenges and Responses

Jaturon Chattratichat John Darlington Moustafa Ghanem
Yike Guo Harald Hüning Martin Köhler
Janjao Sutiwaraphun Hing Wing To Dan Yang

Department of Computing,
Imperial College, London SW7 2BZ, U.K.

Abstract

Data mining over large data-sets is important due to its obvious commercial potential. However, it is also a major challenge due to its computational complexity. Exploiting the inherent parallelism of data mining algorithms provides a direct solution by utilising the large data retrieval and processing power of parallel architectures. In this paper, we present some results of our intensive research on parallelising data mining algorithms. In particular, we also present a methodology for determining the proper parallelisation strategy based on the idea of algorithmic skeletons and performance modelling. This research aims to provide a systematic way to develop parallel data mining algorithms and applications. [1]

Parallelism in Data Mining Algorithms

Numerous algorithms have been previously developed for data mining (Fayyad, Piatetsky-Shapiro, & Smyth 1996). The rich degree of inherent parallelism existing in these algorithms allows some flexibility in choosing the parallelisation scheme that most suited for a particular parallel machine. Two major methods for exploiting parallelism within data mining algorithms can be identified as *task parallelism* and *data parallelism*. A brief summary of how the two approaches can be used to parallelise different data mining algorithms is given in Table 1.

In the task parallelism approach the computation is partitioned amongst the processors of a parallel machine with each processor computing a distinct part of a learning model before co-ordinating with the other processors to form the global model. In the data parallelism approach the training set is partitioned amongst the processors with all processors synchronously constructing the same model; each operating on a different portion of the data-set.

Our practical experience with parallelising data mining algorithms showed an interesting phenomenon. While the parallelisation of certain data mining algorithms shows a consistent performance behaviour when applied to different data-sets, this is not necessarily true across all algorithms. For example, we have implemented a task parallel feed-forward neural network with back-propagation on the Fujitsu AP1000 MPP machine (Ishihata *et al.* 1991). The performance of the implementation illustrated in Figure 1 shows that it is scalable across both processor numbers as well as network size. This performance is also consistent when the implementation is applied to various data-sets.

Algorithm	Task Parallelism	Data Parallelism
Classification Tree	Each branch of the tree is formed into a task.	Training set is partitioned across the processors. Processors evaluate a node of tree in parallel.
Neural Network	Network divided into sub-nets. The sub-nets are allocated across the processors.	Network is duplicated onto each processor. Training set it split across the processors.
Association Rule	Itemset is divided across the processors with selective copying such that each processor can independently count and choose candidates.	Data is distributed across the processors and each node counts and chooses candidates synchronously.

Table 1: Techniques for parallelising data mining algorithms.

However, for other algorithms such as induction based classification algorithms, there appears to be no "ideal" scheme for their parallelisation. The performance of the different parallelisation scheme varies greatly with the characteristics of data-set to which the algorithm is applied. It is these algorithms this paper attempts to investigate.

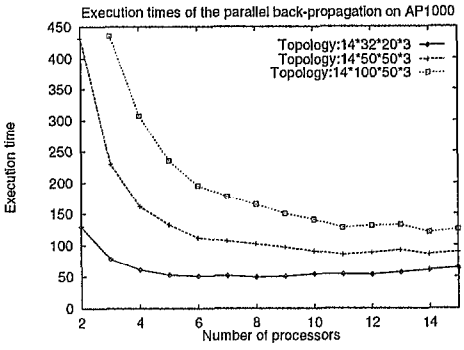


Figure 1: Execution time of PNN on the AP1000.

Parallelising Classification Algorithms

In this section, we describe our experiments in parallelising a well known tree induction algorithm; C4.5 (Quinlan 1993). The aim of these experiments was to gain a deeper understanding of the different parallelisation schemes for classification algorithms.

C4.5 is a classification-rule learning system which uses decision trees as a model representation. The tree models are inferred by using an inductive learning approach applied to a training set. C4.5 attempts to find the simplest decision tree that describes the structure of the data-set by using heuristics based on information theory. These decide which attribute to select for any given node based on its information gain, i.e. its ability to minimise the information needed to further classify the cases in the resulting sub-trees. In our experiments we have implemented both task and data parallel versions of C4.5.

Task Parallel Implementations Several schemes for exploiting task parallelism between the branches of the decision tree exist. We adopted a Dynamic Task Distribution Scheme (DTD) where a master processor maintains a queue of idle slave processors. The subtrees generated at the root of the classification tree are initially dispatched for execution on distinct slave processors. Any slave processor expanding an internal node can also allocate its generated branches to other slaves by initiating a request to the master processor.

Data Parallel Implementations Two variations of data parallel (DP) scheme can be used with C4.5. Given a data-set with size N of k–dimensional data, the entire work load of the algorithm can be characterised by $N \times k$. We have implemented two data parallel C4.5 systems, the first (DP-rec) starts by distributing the data-set N across P processors such that each processor is responsible for N/P records each containing the full k attributes. The second scheme (DP-att) distributes the k attributes across the processors such that each processor is responsible for N records each of k/P attributes.

In both schemes all the processors co-operate in a SPMD (Single Program Multiple Data) fashion in the computation required to expand a particular node. Each processor begins by computing the same function over its local portion of the data-set. The processors then synchronise by exchanging their local information to create a global view of the node computation. the process continues until the full tree has been expanded.

The three schemes have very different execution characteristics. The DTP scheme benefits from the independence between the different tasks executed in parallel but is sensitive to load balancing problems resulting from poor distribution of data elements between branches of a tree. The DP-rec implementation suffers from high communication overheads as the depth of the tree increases, but should perform well with large data sets. The DP-attr scheme has the advantages of being both load-balanced and requiring minimal communications. It is however not generally scalable with the number of processors.

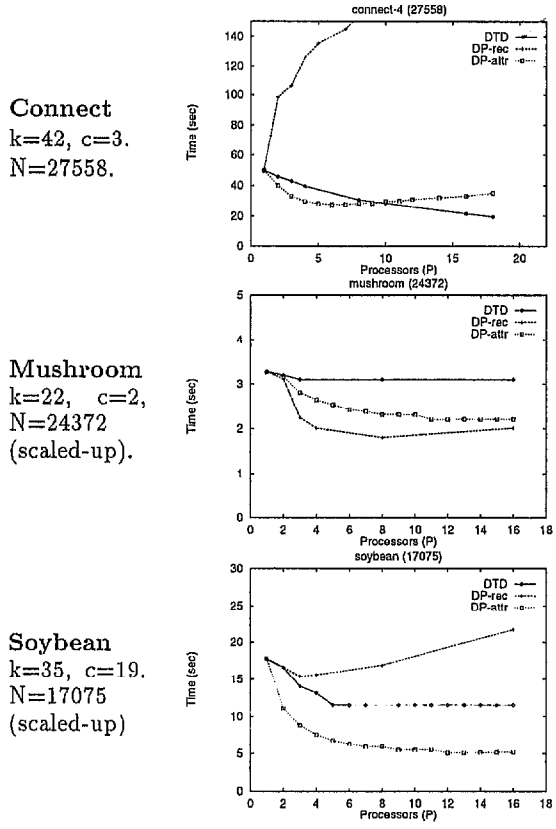
Performance Results The performance of the different implementations were compared using the SOYBEAN, MUSHROOM and CONNECT-4 training sets obtained from the UCI Repository of Machine Learning Database (Merz & Murphy 1996). Both the characteristics of the different data-sets used in the experiments and the measured execution time of the experiments for a varying the number of processors from 2 to 16. are shown in Figure 2. Note that k is the number of attributes in record, c is the number of classes and N is the number of cases used in the training set.

The results mainly indicate that the choice of the best parallelisation strategy highly depends on the data-set being used. For the CONNECT-4 data-set the DTD scheme offered the best performance. This can be attributed to the dense shape of the generated tree where the addition of more processors allows the execution of more nodes in parallel. For the SOYBEAN data-set, the DP-attr version offered the best performance. This version outperforms the DP-rec scheme due to the large number of classes and attributes which govern the amount of data being exchanged in the DP-rec scheme. In contrast using the MUSHROOM data-set, the DP-rec scheme offered the best performance where the data set is characterised by a large training set and low number of classes and attributes.

Structuring Parallel Data Mining

The experiments presented in the previous section highlight some of the difficulties faced when developing efficient implementations of parallel data mining programs where the choice between different parallelisation methods depends on the characteristics of the target machine architecture. This "dynamic" algorithmic behaviour is one of the major challenges for parallelising data mining applications.

At Imperial College we have been developing a structured approach to parallel programming based on the



Connect
k=42, c=3.
N=27558.

Mushroom
k=22, c=2,
N=24372
(scaled-up).

Soybean
k=35, c=19.
N=17075
(scaled-up)

Figure 2: Execution time of different data-sets on the AP1000

use of high-level skeletons (Darlington *et al.* 1995; Darlington, Ghanem, & To 1993). The approach offers a solution to the above challenge by making the choice of the appropriate parallelisation strategy based on the characteristics of the data-set much easier.

Structured Parallel Programming

At the heart of our structured parallel programming approach is the concept of a set of predefined high-level skeletons. A skeleton can be viewed as a software template that specifies a particular pattern of parallel computation and communication. Each skeleton captures a widely used parallel method which is often common to a wide range of applications. A parallel program is first expressed by choosing a skeleton which reflects the parallel structure of the program. The parameters of the skeleton are then instantiated with problem specific sequential function calls.

For example, The D&C skeleton captures the recursive divide and conquer behaviour common to many algorithms such as C4.5. These algorithms work by splitting a large task into several sub-tasks, solving the sub-tasks independently and then combining the results. Since the sub-tasks are independent, they can be executed on separate processors. A functional specification of the D&C skeleton is as follows:

```
DC(simple, solve, divide, combine, P ) =
    if simple (P) then solve (P)
    else
        combine (n, Farm(simple, solve,
                                divide, combine S_i))
        where
            (n, S) = divide (P) S_i ∈ S
```

In this skeletons, the functions simple performs a triviality test on a given task. If the task is trivial it is then solved by the function solve. Otherwise the larger tasks are divided by the function divide into a set of sub-tasks S and a node n. These sub-tasks solved recursively and the sub-results combined by the function combine . The task parallel implementation of the D&C skeleton is specified by the Farm skeleton which captures the behaviour of executing a set of independent tasks in parallel.

By decomposing C4.5 into four basic functions a task parallel tree construction procedure of C4.5 can be structured as a special instance of the D&C skeleton. The functions are: class for computing the set of class values of a data-set, select-attribute for selecting attribute by computing the information gain and test for partitioning the data by logical test on the selected attribute. The algorithm can be specified thus:

```
Parallel-Tree-Construction (S) =
    DC (simple, solve, divide, combine, S)
    where
        simple S = is-singleton (class (S))
        solve S = class (S)
        divide S = test (select-attribute (S))
        combine n T = node( n, T)
```

Navigating using Performance Models

Skeletons are mainly used to provide a precise abstraction of computational behaviour of a parallel implementations while hiding their implementation details. The patterns of parallelism captured by each skeleton are quite general. The exact implementation of a given skeleton may change from one parallel machine to another to make use of a given machine's features. However the use of skeletons to express the control structure of a parallel program enables the capturing of the main features of a parallelisation strategy. This provides a crucial aid in determining the appropriate choice of both a parallelisation scheme and its underlying implementation.

One of the main advantages of the skeleton approach is the ability to associate performance models with each skeleton implementation and machine pair (Darlington *et al.* 1996). This is due to the pre-design of each skeleton implementation which enables the development of a performance formula predicting its runtime performance. For example, the performance of

divide and conquer algorithms can be specified recursively in terms of N, the problem size, B, the branching factor and the total execution times of division and combination routines at a given node, $T_{node}(N, B)$. The performance of the task parallel implementation is

$$T_{dc}(N) = T_{dc}(X/B) + T_{node}(N, B)$$

where

$$T_{node}(N, B) = T_{div}(N, B) + T_{comb}(N, B) + T_{triv}(N, B)$$

This performance model does not give absolute predictive execution times, but is designed to be sufficient for distinguishing between the use of alternative implementations (Darlington *et al.* 1996). Thus they can be used to help the user navigate through the space of all possible implementations, without the need to execute the code. This property is the key to determine the choice of right parallel implementation of a data mining algorithm with respect to a particular application.

Structured Parallelisation of C4.5

As an example of using the skeletons approach the different parallelisation strategies for the C4.5 classification algorithm can be implemented using the skeletons shown in Figure 3. Each path in this option tree specifies a possible implementation of the program in terms of skeletons.

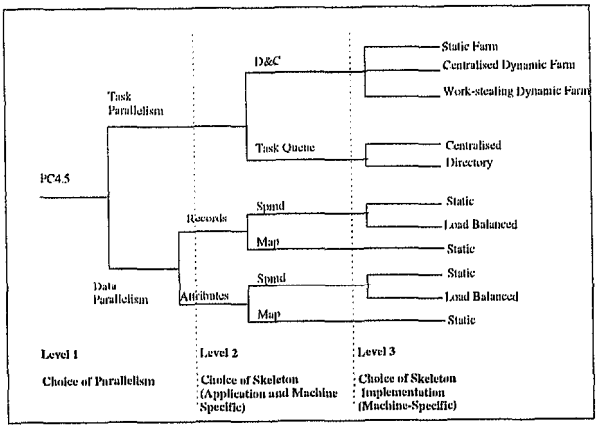


Figure 3: C4.5 parallelisation road map

For algorithms such as C4.5 where the characteristics of the data-set affect the performance of the different implementations, the relationship between the skeleton performance models and the data-set needs to be developed. This can be done by examining the size of the data-set (scale), the number of attributes of the data (complexity), the ratio between the number of continuous attributes and the number of discrete attributes (structure of the data-set) and the characteristics of the problem domain (structure of the tree). The correlation between the *performance models* of parallel data mining systems and the *feature model* of the data-set provides a *decision model* of determining a proper parallel implementation for a particular problem.

Conclusions

In this paper we have investigated the potential for exploiting different parallelisation strategies for a classification algorithm. The results show that the performance of the different implementation schemes was highly susceptible to the properties of the training set used. We have presented a methodology based on skeletons and performance models which provides an aid to the user for navigating the space of parallel implementations of a given algorithm solving a particular problem. We are currently investigating the precise modelling of the implementations and building up the correlation between the performance of these implementations and the data features.

References

Darlington, J.; Guo, Y.; To, H. W.; and Yang, J. 1995. Functional skeletons for parallel coordination. In Haridi, S.; Ali, K.; and Magnussin, P., eds., *Euro-Par'95 Parallel Processing*, 55–69. Springer-Verlag.

Darlington, J.; Ghanem, M.; Guo, Y.; and To, H. W. 1996. Guided resource organisation in heterogeneous parallel computing. *Submitted to the Journal of High Performance Computing.*

Darlington, J.; Ghanem, M.; and To, H. W. 1993. Structured parallel programming. In *Programming Models for Massively Parallel Computers*, 160–169. IEEE Computer Society Press.

Fayyad, U. M.; Piatetsky-Shapiro, G.; and Smyth, P. 1996. From data mining to knowledge discovery: An overview. In Fayyad, U. M.; Piatetesky-Shapiro, G.; Smyth, P.; and Uthurusamy, R., eds., *Advances in Knowledge Discovery and Data Mining.* MIT Press.

Ishihata, H.; Horie, T.; Inano, S.; Shimizu, T.; Kato, S.; and Ikesaka, M. 1991. Third generation message passing computer AP1000. In *International Symposium on Supercomputing*, 46–55.

Merz, C. J., and Murphy, P. M. 1996. UCI repository of machine learning databases. University of California,
Department of Information and Computer Science, http://www.ics.uci.edu/~mlearn/MLRepository.html.

Quinlan, J. R. 1993. *C4.5 Programs for Machine Learning.* Morgan Kaufmann Publishers, Inc.