Towards a Parallel Data Mining Toolbox

Peter Christen* Markus Hegland Ole M. Nielsen Stephen Roberts Peter Strazdins
Tatiana Semenova
Australian National University, Canberra, ACT 0200, Australia

Irfan Altas
Charles Sturt University, Wagga Wagga, NSW 2678, Australia

Timothy Hancock
James Cook University, Townsville, QLD 4811, Australia

Abstract

This paper presents research projects tackling two aspects in data mining. First, a toolbox is discussed that allows flexible and interactive data exploration, analysis and presentation using the scripting language Python. The advantages of this toolbox are that it provides the functionality to process multiple SQL queries in parallel, and enables fast data retrieval using a supervised caching mechanism for commonly used queries. These two facets of the toolbox allow for fast, efficient data access reducing the time spent on data exploration, preparation and analysis.

Secondly, an approach to predictive modelling is presented that leads to scalable parallel algorithms for high dimensional data collections. This is an essential requirement for data mining algorithms as those that do not scale linearly with the data size are infeasible. These algorithms are implemented in parallel and achieve an almost ideal speedup for their respective implementations.

One aim of the presented research is to integrate and combine these two different aspects of data mining into an efficient but flexible data mining toolbox that allows the experienced data miner to attack large scale problems interactively or with batch processing.

1 Introduction

There is much ongoing research in sophisticated algorithms for data mining purposes. Examples include predictive modelling, genetic algorithms, neural networks, decision trees, association rules, and many more. However, it is generally accepted that it is not possible to apply such algorithms without a careful data preparation and data understanding process, which may often dominate the actual data mining activities [8, 20]. It is also rarely feasible to use off-the-shelf data mining software and expect useful results without a substantial amount of data insight. In addition, data miners working as consultants are often presented with datasets from an unfamiliar domain and need to get a good feel for the data and the domain prior to any "real" data mining. The ease of initial data exploration and preprocessing may well hold the key to successful data mining results later in a project. These processes are highly interactive: The data miner investigates the data and extracts subsets of attributes or transactions to be mined and conducts experiments that lead to new ideas and questions requiring further exploration. Fast and flexible data querying and aggregation are therefore mandatory.

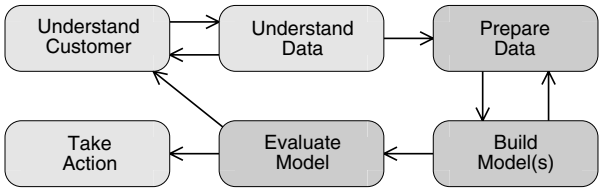

Figure 1. The data mining process

Data mining is an iterative process (Figure 1), as various steps may have to be repeated several times until useful and valuable knowledge is found, e.g. the same mining algorithm is used on different subsets of a data collection to compare different outcomes. A lot of effort in a data mining project is often spent with time consuming routine tasks. A caching of intermediate results can thus shorten response times tremendously and help the data miner to concentrate on knowledge extraction.

Today data collections have the size of Gigabytes and

*Corresponding author, E-Mail: Peter.Christen@anu.edu.au

Terabytes, and the first Petabyte databases are appearing in science [10]. Data mining tools thus have to be able to handle large amounts of data efficiently and they also need to be scalable with the increasing size of data collections. Therefore, algorithms which do not scale linearly with the data size are not feasible. Additionally, the dimensionality of datasets is increasing, which is a major challenge for many algorithms as their complexity grows exponentially with the dimension of a dataset. This has been called the *curse of dimensionality* [13]. Parallel processing can help both to tackle larger problems and to get reasonable response times. It is not only that more computing power becomes available, but equally important is the increased I/O bandwidth and larger main memory provided by most parallel machines.

A further challenge in real world data mining projects is the various data formats that have to be dealt with, like relational databases, flat text files, non-portable binary files or data downloaded from the Web. A flexible *middleware* layer can unify the view of different data collections and facilitate the application of various tools and algorithms.

This paper presents ideas and methods that tackle many of the described challenges. A toolbox approach using the scripting language Python [5] allows flexible data exploration, while parallel predictive modelling algorithms that are scalable both with the number of attributes of a data collection as well as the number of used processing nodes, allow mining of high-dimensional data sets. The toolbox *DMtools* [18] is currently under development and a predecessor has successfully been applied in real-world data mining projects under the ACSys CRC grant[1]. The toolbox assists our research group in all stages of data mining projects, starting from data preprocessing, analysis and simple summaries up to visualisation and report generation. In Section 2 related work in the relevant areas is presented, and Section 3 presents our toolbox approach in more detail. Section 4 discusses our scalable parallel algorithms for predictive modelling, and Section 5 gives an outlook to future work.

2 Related work

There are several projects describing toolbox like approaches to data exploration. The authors of the IDEA (Interactive Data Exploration and Analysis) system [21] identify five general user requirements for data exploration: Querying (the selection of a subset of data according to the values of one or more attributes), segmenting (splitting the data into non-overlapping sub-sets), summary information (like counts or averages), integration of external tools and applications, and history mechanisms. The IDEA framework allows quick data analysis on a sampled sub-set of the data with the possibility to re-run the same analysis later on the complete dataset. IDEA runs on a PC, with the user interacting on a graphical interface.

Another approach used in the Control [14] project is to trade quality and accuracy for interactive response times, in a way that the system quickly returns a rough approximation of a result that is refined continuously. The user can therefore get a glimpse at the final result very quickly and use this information to change the ongoing process. The Control system, among others, includes tools for interactive data aggregation, visualisation and data mining.

Database research and data mining are two related fields and there are many publications dealing with both areas. An overview of database mining is given in [7]. According to the authors the efficient handling of data stored in relational databases is crucial because most available data is in a relational form. Scalable and efficient algorithms are one of the challenges, as is the development of high-level query languages and user interfaces so data mining tasks can be specified by non-experts. One of the identified key requirements is interactivity. The possibilities to interactively analyse data collections, to refine data mining requests, to deepen the analysis and to change the focus should be encouraged, because it is often difficult to predict what exactly could be discovered from a dataset. Interactive data mining is also needed if transformation and manipulation of data are necessary, or if different subsets of a data collection are to be examined. To be able to deal with the huge amounts of data available, parallel and distributed data mining algorithms are important, as is the possibility to mine information from different sources of data.

Parallel data mining is a hot research topic (see [24] for recent research papers), as the need for parallel processing is clearly given by the huge and increasing data collections available. The requirements for parallel KDD systems [17] include not only parallel scalable hardware platforms, parallel I/O and databases, and parallel data mining algorithms, but also frameworks for rapid algorithm development and evaluation. Issues like security, fault tolerance, heterogeneous data access and representation, quality of service, pricing and portability have to be addressed as well. Large-scale parallel KDD systems should support the entire KDD process, including pre- and post-processing.

3 A toolbox for data mining

Using a portable, flexible, and easy to use toolbox can not only facilitate the data exploration phase of a data mining project, it can also help unifying the data access with a middleware library to integrate the access of different data sources to the data mining applications. Thus it forms the

[1] ACSys CRC stands for 'Advanced Computational Systems Collaborative Research Centre' and the data mining consultancies were conducted at the Australian National University (ANU) in collaboration with the Commonwealth Scientific and Industrial Research Organisation (CSIRO)

framework for the application of a suite of more sophisticated data mining algorithms. The command line driven approach of a toolbox – which is also used in packages like Matlab or Mathematica – is maybe not suited for a novice user, but for the experienced data miner it provides a more powerful and flexible tool than a graphical user interface.

The *DMtools* [18] are based on the scripting language *Python* [5], chosen since it has proven to be an excellent tool for rapid code development. It allows for efficient handling of large datasets, it is flexible and easily extensible. Functions and routines can be used as templates which can be changed and extended as needed by the user to do more customised analysis tasks. Having a new data exploration idea in mind, the data miner can implement a rapid prototype very easily by writing a script using the functions provided by our toolbox.

Because many data collections are stored in relational databases, it is important that such data can be accessed efficiently by data mining applications [6]. Furthermore, databases using SQL are a standard tool for storing and accessing transactional data in a safe and well-defined manner. However, both complex queries and transmissions of large data quantities tend to be prohibitively slow. For our toolbox we follow another route: Only simple queries (e.g. no joins) are sent to the database server, and the results are cached and processed within the toolbox. The Python database API [15] allows us to access a relational database by SQL queries. Currently, we are using MySQL [23] for the underlying database engine, but Python modules for other database servers are available as well. Both MySQL and Python are licensed as free software and enjoy very good support from a large user community. In addition, both products are very efficient and robust.

3.1 Toolbox architecture

In our toolbox the ease of SQL queries and the safety of relational databases are combined with the efficiency of binary file access and the flexibility of object-oriented programming languages in an architecture as shown in Figure 2. Based on relational databases, flat files, the Web, or any other data source a *Data Manager* deals with retrieval, caching and storage of data. It provides routines to execute an arbitrary SQL query and to read and write binary and text files. The two important core components of this layer are its transparent caching mechanism and its parallel database interface which intercepts SQL queries and parallelises them on-the-fly. The *Aggregation* module implements a library of Python routines taking care of simple data exploration, statistical computations, and aggregation of raw data. The *Modelling* module contains functions for parallel predictive modelling, clustering, and generation of association rules. The *Report* module provides visualisation

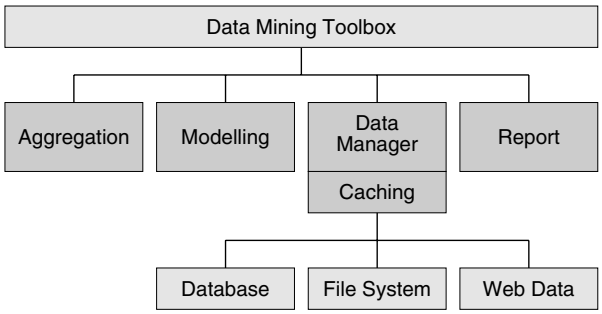

Figure 2. Architecture of DMtools

and allows facilities for simple automatic report generation. Complex domain-specific functions and end-user applications can then be written in terms of the scripting language making use of the available modules.

3.2 Caching and database parallelism

Caching of function results is a core technology used throughout our data mining toolbox. We have developed a methodology for *supervised* caching of function results as opposed to the more common (and also very useful) *automatic* disk caching provided by most operating systems and Web browsers.

Like automatic disk caching, supervised caching trades space for time, but the approach we use is one where time consuming operations such as database queries or complex functions are intercepted, evaluated and the resulting objects are made persistent for rapid retrieval at a later stage. We have observed that most of these time consuming functions tend to be called repetitively with the same arguments. Thus, instead of computing them every time, the cached results are returned when available, leading to substantial time savings. The repetitiveness is even more pronounced when the toolbox cache is shared among many users, a feature we use extensively.

This type of caching is particularly useful for computationally intensive functions with few frequently used combinations of input arguments. Note that if the inputs or outputs are very large, caching might not save time because disk access may dominate the execution time. Supervised caching is invoked in the toolbox by explicitly applying it to chosen functions. For a given Python function of the form `T = func(arg1,...,argn)` caching in its simplest form is invoked by replacing the function call with the following call: `T = cache(func,(arg1,...,argn))`. This structure has been applied in the *Data Manager* module of the toolbox so using the top level toolbox routines will utilise caching transparently. For example, most of the SQL queries that are automatically generated by the toolbox are cached in this fashion. Generating queries automatically in-

creases the chance of cache hits as opposed to queries written by the end user because of their inherent uniformity. In addition to this, caching can be used in code development for quick retrieval of precomputed results. For example, if a result is obtained by automatically crawling the Web and parsing HTML or XML pages, caching will help in retrieving the same information later – even if a Web server is unserviceable.

An example query used in a health data mining project that extracts all patients belonging to a particular group together with a count of their transactions required on the average 489 seconds worth of CPU time on a Sun Enterprise server and the result took up about 200 Kilobytes of memory. After having cached this query, subsequent loading takes 0.22 seconds – more than 2,000 times faster than the computing time. This particular function was hit 168 times in a fortnight saving four users a total of 23 hours of waiting.

If a function definition changes after a result has been cached, or if the result depends on other files, wrong results may occur when using caching in its simplest form. The caching utility therefore supports specification of explicit dependencies in the form of a file list, which, if modified, triggers a recomputation of the cached function result.

If a database server allows parallel execution of independent queries, we use this parallelism within our toolbox by sending a list of queries to the database server and process the returned results sequentially. This is very efficient if the queries take a long time to proceed, but only return a small result list.

Example 3.1 *Caching of XML documents*

Supervised caching is used extensively for database querying but is by no means restricted to this. Caching has proven to be useful in other aspects of the data mining toolbox. An example is a Web application built on top of the toolbox which allows managers to explore and rank branches according to one or more user-defined features such as *Annual revenue*, *Number of customers serviced relative to total population*, or *Average sales per customer*. The underlying data is historical sales transaction data which is updated monthly, so features need only be computed once for new data when it is added to the database. Because the data is static, cached features are never recomputed and the application can therefore make heavy use of the cached database queries. Moreover, no matter how complicated a feature is, it can be retrieved as quickly as any other feature once it has been cached. In addition, the Web application is configured through an XML document defining the data model and describing how to compute the features. The XML document must be read by the toolbox, parsed and converted into appropriate Python structures prior to any computations. Because response time is paramount in an interactive application, parsing and interpretation of XML is prohibitive, but by using the caching module, the resulting Python structures are cached and retrieved quickly enough for the interactive application. The caching function was made dependent on the XML file itself, so that all structures are recomputed whenever the XML file has been edited – for example to modify an existing feature-definition, add a new month, or change the data description. Below is a code snippet from the Web application. The XML configuration file is assumed to reside in `sales.xml`. The parser which builds Python structures from XML is called `parse_config` and it takes the XML filename as input. To cache this function, instead of the call `(feature_list, branch_list) = parse_config(filename)` we write:

```
filename = "sales.xml"
(feature_list, branch_list) =     \
  cache(parse_config, filename,   \
        dependencies = filename)
```

3.3 Integration of parallel applications

One aim of our research is to integrate parallel applications into the toolbox to enable fast and efficient execution of large and complex data mining tasks, so the data miner is able to attack large problems interactively or with batch processing. The toolbox also gives a common interface to various data mining algorithms, whereby the details of the parallel application and architecture (like starting and working with a parallel environment) are hidden from the user.

Using a scripting language like Python to control parallel applications has already been used for steering [4, 19] of scientific applications, where a user can change parameters at run-time to control the behaviour of long-running simulations like molecular dynamics applications.

In our toolbox, we use the standard Python interpreter to start parallel applications with a dynamically generated system call. The Unix command `system` is executed through Python to invoke an MPI [12] program like in the following example:

```
mpi_str = "mpirun -np " + str(num_proc) + \
          predmodel " + arg_str + result_str
os.system(mpi_str)
```

The string `arg_str` contains the input arguments and `result_str` contains the name of the result file. A Python wrapper code converts toolbox objects (like lists or dictionaries) into a format which is processable by the parallel application, and creates configuration and data files as necessary. The parallel application gets its input parameters from the Python script at the command line, and then loads and processes the requested data files. Results are saved to files by the parallel program and read by the toolbox for further processing. Time consuming data mining algorithms

can therefore be run in parallel, while the Python toolbox handles the more high level aspects of data mining like preprocessing and visualisation.

4 Scalable parallel predictive modelling

Algorithms applied in data mining have to deal with two major challenges: Large datasets and high dimensionality. It has also been suggested that the size of databases in an average company doubles every 18 months [3] which is similar to the growth of hardware performance according to Moore's law. Consequently, data mining algorithms have to be able to scale from smaller to larger data sizes when more data becomes available. The complexity of data is also growing as more attributes tend to be logged in each record. Data mining algorithms must, therefore, be able to handle high dimensions in order to process such datasets, and algorithms which do not scale linearly with data size and dimension are not feasible.

An important technique applied in data mining is predictive modelling. A predictive model in some way describes the average behaviour of a data set, and can be used to find data records that lie outside of the expected behaviour. These *outliers* often have simple natural explanations but, in some cases, may be linked to fraudulent behaviour.

A predictive model is described by a function $y = f(x_1, \ldots, x_d)$ from the set, T, of attribute vectors of dimension d in the response set, S. If S is a finite set (often $S = \{0, 1\}$), the determination of f is a *classification problem* and if S is the set of real numbers, one speaks of *regression*. In the following it will mainly be assumed that all the attributes x_i as well as y are real values and we set $\mathbf{x} = (x_1, \ldots, x_d)^T$.

In many applications, the response variable y is known to depend in a smooth way on the values of the attributes, so it is natural to compute f as a least squares approximation to the data with an additional smoothness component imposed. In this paper, we state the problem formally as follows. Given n data records $(\mathbf{x}^{(i)}, y^{(i)}), i = 1, \ldots, n$ where $\mathbf{x}^{(i)} \in \Omega$ with $\Omega = [0, 1]^d$ (the d-dimensional unit cube), we wish to minimise the following functional subject to some constraints:

$$J_\alpha(f) = \sum_{i=1}^{n} (f(\mathbf{x}^{(i)}) - y^{(i)})^2 + \alpha \int_\Omega |\mathcal{L} f(\mathbf{x})|^2 \, d\mathbf{x} \quad (1)$$

where α is the smoothing parameter and $\mathcal{L}$ is a differential operator whose different choices may lead to different approximation techniques. The smoothing parameter α controls the trade-off between smoothness and fit. One can choose different function spaces to approximate the minimiser f of equation (1).

We have developed three different methods [9] to approximate the minimiser of Equation (1). TPSFEM uses piecewise multilinear finite elements and gives the most accurate approximation at the highest computational costs; HISURF is based on interpolatory wavelets which provides good approximations at reasonable costs; and ADDFIT implements additive models which have the lowest costs but give the coarsest approximation. The three methods differ in how well they approximate f and more importantly in their algorithmic complexities, but all three consist of the following two steps:

1. **Assembly**: An $m \times m$ symmetric matrix $\mathbf{A}$ and a corresponding $m \times 1$ vector $\mathbf{b}$ are assembled whose structures depend on the chosen method, but whose dimension m is independent on the number of data records n. For the TPSFEM method, the matrix structure is sparse with 3^d filled diagonals (d the dimensionality of the dataset), and for both HISURF and ADDFIT we store dense matrices. The size m of the assembled linear system depends on the total number of categories for categorical variables and on the resolution of the finite element grid for continuous variables. The assembly step requires access to all n data records once only, and it can be organised such that the amount of computational work is linear in n. As usually $m << n$ this step can be interpreted as a reduction operation on the original data. Note that the assembly of the matrices coming from the smoothing part of Equation (1) and constraints do not require accessing the data at all. These matrices have similar sizes to $\mathbf{A}$.

2. **Solving**: This step assembles the $m \times m$ matrices coming from the smoothing part of Equation (1) and solves the entire linear system. It does not involve the n data records and the computational work depends only on m, typically as $O(m^3)$ for the dense and $O(m)$ for the sparse equations.

Note that for large n step 1 will dominate whereas for large m step 2 will dominate. As the number of data records n is usually very large for data mining applications, the overall complexity is mainly determined by n.

The process of assembling the linear systems has the same structure for all three methods. For each data record, some nonzero elements are added into the matrix and vector. The number of nonzero elements per data record is $O(d)$, forming the normal equations matrix $\mathbf{A}$ is thus of order $O(d^2)$ for each data record. The total complexity of assembling n data records sequentially is therefore:

$$T_{assem}(1) = O(d^2 n) \quad (2)$$

The assembly of data records into the linear system is additive and thus each data record can be assembled independently from all others. By reading a fraction n/p of the

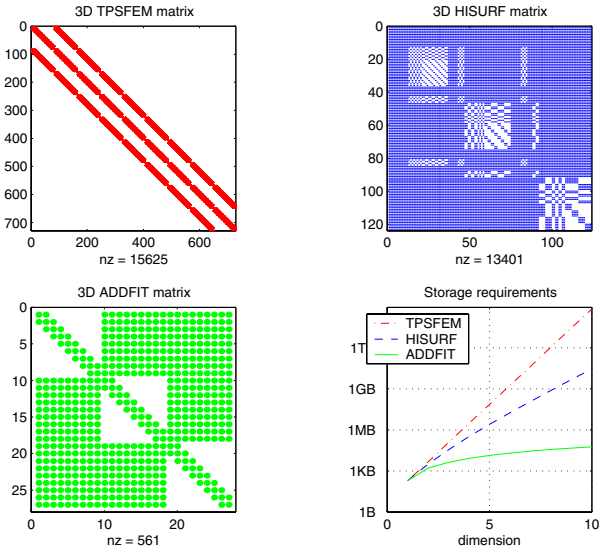

Figure 3. Matrix structure (3D) and complexity

dataset in parallel each processing node can assemble a local linear system without communication, so the parallel complexity of the assembly process on p nodes becomes:

$$T_{assem}(p) = O\left(\frac{d^2 n}{p}\right) \qquad (3)$$

The complete matrix data structure has to be stored on each node, as every data record can contribute nonzero elements anywhere in the matrix. The linear system is therefore distributed, but not replicated, on the nodes and the final linear system is the sum of all local linear systems.

Figure 3 shows the structure of the assembled data matrix for the three methods for a 3D-problem with a grid resolution of nine points in each dimension. The lower right graph gives the amount of memory needed for a 1D to 10D problem (again with a grid resolution of nine points in each dimension). One can clearly see the limitation of the TPSFEM and HISURF methods due to their storage requirements.

The assembly process without communication is limited by the available amount of main memory. For matrices that are too large, a more complex assembly has to be applied (not yet implemented), where the matrix data has to be distributed in a memory-scalable way. A blocking structure of reading and redistributing data will be used.

4.1 Parallel implementation

At the time of writing, the assembly phase has been implemented for ADDFIT in ANSI C and using MPI [12] for communication. As the basic assembly structure is the same

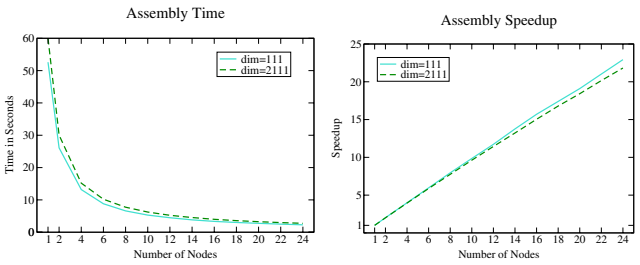

Figure 4. Assembly on Beowulf Linux cluster

for all three methods, it is simple to include analogous routines for TPSFEM and HISURF: The matrix data structures and the assembly of data records are the only parts that have to be changed.

First timing tests with synthetic datasets (consisting of 5 million records with 10 attributes each) show an almost ideal speedup for different matrix dimensions. As an example Figure 4 illustrates the times and speedup achieved on the 196 processor Beowulf Linux cluster *Bunyip* at the Australian National University [1]. This cluster is built with 98 dual 550 MHz Pentium III nodes, each equipped with 384 Megabytes of RAM (total about 36 Gigabytes), 13 Gigabytes of disk space (total 1.3 Terabytes) and 3×100 MBit/s fast Ethernet cards. Logically 96 nodes are connected in four groups of 24 nodes arranged as a tetrahedron with a group of nodes at each vertex. Two nodes are designated as servers. For our tests we only used one group (i.e. up to 24 nodes), whereas all data files have been distributed onto local disks. The results in Figure 4 show the times used for the assembly and redistribution of two linear systems of different size (corresponding to a different number of used attributes). The communication time is almost negligible compared to the assembly time, but yields to a small time increase for the larger linear system.

Solving the assembled linear system can be done with either a sequential or parallel solver, depending on the size of the system and the available parallel architecture. The sparse linear system resulting from TPSFEM can be solved with a Conjugate-Gradient iterative solver approach [9].

The systems currently generated by HISURF and ADDFIT are dense and symmetric, positive definite in the former case, and semi-definite in the latter case. However, in future refinements of these models, the definiteness property may be lost, for example because of the addition of extra constraints or – in the case of additive models – extending it to a second-order model.

For HISURF and ADDFIT a solver is thus required that will be accurate for any symmetric dense system, and also has good parallel and sequential performance. The former requirement argues for a direct solver with good stability properties; the latter argues for one that exploits symme-

try to require only $\frac{m^3}{3} + O(m^2)$ floating point operations, and that has been shown to have an efficient parallelisation. A direct solver for general symmetric (indefinite) systems based on the diagonal pivoting method [2, 11] meets these requirements.

In the diagonal pivoting method, the decomposition $A = LDL^T$ is performed, where L is an $m \times m$ lower triangular matrix with a unit diagonal, and D is a block diagonal matrix with either 1×1 or 2×2 sub-blocks [11]. The factorisation of A proceeds column by column; in the elimination of column j, three cases arise:

1. Eliminate using a 1×1 pivot from $A_{j,j}$. This corresponds to the definite case, and will be used when $A_{j,j}$ is sufficiently large (compared with $\max(A_{j+1:m,j})$).

2. Eliminate using a 1×1 pivot from $A_{i,i}$, where $i > j$. This corresponds to the semi-definite case; a symmetric interchange with row/columns i and j must be performed.

3. Eliminate using a 2×2 pivot using columns i' and i ($i', i \geq j, i' \neq i$). This case produces a 2×2 sub-block at column j of D. This corresponds to the indefinite case; a symmetric interchange with rows/columns i', i and $j, j + 1$ must be performed. However, columns j and $j + 1$ are eliminated in this case.

The tests used to decide between these cases, and the searches used to select column i (and i'), yield several algorithms based on the method, the most well-known being the variants of the *Bunch-Kaufman* algorithm (see [11] and the references cited within).

It has been recently shown for the *Bunch-Kaufman* algorithm that there is no guarantee that the growth of L is bounded [2]. Variants such as the *bounded Bunch-Kaufman* and *fast Bunch-Parlett* algorithms have been devised which overcomes this problem. The extra accuracy of these methods results from more extensive searching for stable pivot columns i (and i') for cases 2 and 3, with a correspondingly more frequent use of these cases.

For linear systems that are close to definite, such as are likely to be generated by our models, the diagonal pivoting methods permit most columns to be eliminated by case 1, requiring no symmetric interchanges. For a parallel implementation, this is a highly useful property, as even for large matrices the communication startup and volume overheads of symmetric interchange, when the rows and columns come from different nodes, is considerable [22].

Instead of suppressing interchanges, which even if done judiciously may result in the loss of some accuracy [22], high parallel performance can also be achieved with a *block-search* algorithm that searches for suitable pivot columns i and i' from the current storage block [16]. If this search was successful, the symmetric interchanges would require no communication, resulting in no parallel overhead. Such a strategy could be based on the *Duff-Reid* algorithm used for sparse matrices [2, 16], which also has strong guarantees of accuracy.

However, if the search was not successful, an equally stable means of eliminating column j must then be used. We chose the *bounded Bunch-Kaufman* algorithm over the *fast Parlett-Reid* algorithm, as the latter requires sorting of the columns by the size of the diagonal, which would give it higher parallel overheads.

The solving of the linear systems described in Figure 4 took less than one second for the small system and between 20 (sequential) and 11 (parallel) seconds for the large system. On the used cluster architecture even a matrix dimension of 2111 is too small to be efficiently solved on more than just a few computing nodes.

5 Outlook

We are currently working both on the toolbox and on the parallel algorithms. Planned extensions will be more domain independent high-level analysis functions and the inclusion of other parallel data mining algorithms besides predictive modelling, like clustering and association rules.

On the parallel predictive modelling side we plan to add support for data types other than continuous and categorical variables. We hope to include in a first instance support for sets, time series and hierarchical data types.

Acknowledgements

This research was partially supported by the Australian Advanced Computational Systems CRC, and Peter Christen was funded by grants from the *Swiss National Science Foundation* (SNF) and the *Novartis Stiftung, vormals Ciba-Geigy Jubiläums-Stiftung*, Switzerland.

References

[1] D. Aberdeen, J. Baxter and R. Edwards, *98 c/MFlop, Ultra-Large-Scale Neural Network Training on a PIII Cluster*, Gordon Bell award price/performance entry. Submitted to the High-Performance Networking and Computing Conference, Dallas, November 2000.

[2] C. Ashcraft, R.G. Grimes, and J.G. Lewis, *Accurate Symmetric Indefinite Linear Equation Solvers*, Simax, 20(2), 1998.

[3] G. Bell and J.N. Gray, *The revolution yet to happen*, Beyond Calculation (P.J. Denning and R.M. Metcalfe, eds.), Springer Verlag, 1997.

[4] D.M. Beazley and P.S. Lomdahl, *Extensible message passing application development and debugging with Python*, Proceedings 11th International Parallel Processing Symposium, April 1–5, 1997, Geneva, Switzerland, IEEE Computer Society Press, 1997.

[5] D.M. Beazly, *Python Essential Reference*, New Riders, October 1999.

[6] S. Chaudhuri, *Data Mining and Database Systems: Where is the Intersection*, Bulletin of the IEEE Technical Committee on Data Engineering, num. 21, March 1998.

[7] M.-S. Chen, J. Han and P.S. Yu, *Data Mining: An Overview from a Database Perspective*, IEEE Transactions on Knowledge Discovery and Data Engineering, Vol. 8, No. 6, December 1996.

[8] P. Chapman, R. Kerber, J. Clinton, T. Khabaza, T. Reinartz and R. Wirth, *The CRISP-DM Process Model*, Discussion paper, March 1999. `www.crisp.org`

[9] P. Christen, M. Hegland, O.M. Nielsen, S. Roberts, P.E. Strazdins and I. Altas, *Scalable Parallel Algorithms for Surface Fitting and Data Mining*, accepted for the Elsevier Journal of Parallel Computing, special issue on Aspects of Parallel Computing for Linear Systems and Associated Problems, September 2000.

[10] D. Düllmann, *Petabyte databases*. Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD-99), ACM Press, July 1999.

[11] G. Golub and C. Van Loan, *Matrix Computations*, John Hopkins University Press, Baltimore, Second edition, 1989.

[12] W. Gropp, E. Lusk and A. Skjellum,*Using MPI – Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, Cambridge, Massachusetts, 1994.

[13] T.J. Hastie and R.J. Tibshirani, *Generalized additive models*, Chapman and Hall, Monographs on statistics and applied probability 43, 1990.

[14] J.M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman,T. Roth and P. Haas, *Interactive Data Analysis: The Control Project*, IEEE Computer, Vol. 32, August 1999.

[15] A.M. Kuchling, *The Python DB-API*, Linux Journal, May 1998.

[16] J.G. Lewis. Private communications, 1999.

[17] W.A. Maniatty and M.J. Zaki, *A Requirements Analysis for Parallel KDD Systems*, IPDPS'2000 Data Mining Workshop, Cancun, Maxico, May 2000.

[18] O.M. Nielsen, P. Christen, M. Hegland and T. Semenova, *A Toolbox Approach to Flexible and Efficient Data Mining*, Accepted for the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'2001), Hong Kong, April 2001.

[19] S.G. Parker, C.R. Johnson and D.M. Beazley, *Computational Steering Software Systems and Strategies*, IEEE Computational Science & Engineering, 1997.

[20] D. Pyle, *Data Preparation for Data Mining*. Morgan Kaufmann Publishers, Inc., 1999.

[21] P.G. Selfridge, D. Srivastava and L.O. Wilson, *IDEA: Interactive Data Exploration and Analysis*, Proceedings of the ACM SIGMOD International Conference on Management of Data, 1996.

[22] P.E. Strazdins, *Accelerated Methods for Performing the LDLT Decomposition*, Proceedings of CTAC-99: The 9th Biennial Computational Techniques and Applications Conference and Workshops, Canberra, September 1999.

[23] R.J. Yarger, G. Reese and T. King, *MySQL & mSQL*, O'Reilly, July 1999.

[24] M.J. Zaki and C-T. Ho, *Large-Scale Parallel Data Mining*, Lecture Notes in Computer Science and Lecture Notes in Artificial Intelligence, Vol. 1759, Springer-Verlag, 2000.