A Knowledge Discovery Methodology for the Performance Evaluation of Scientific Software

Vassilios S. Verykios

Elias N. Houstis
Purdue University, enh@cs.purdue.edu

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
98-042

A KNOWLEDGE DISCOVERY METHODOLOGY FOR THE PERFORMANCE EVALUATION OF SCIENTIFIC SOFTWARE

Vassilios S. Verykios
Elias N. Houstis
John R. Rice

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD-TR #98-042
November 1998

A Knowledge Discovery Methodology for the Performance Evaluation of Scientific Software *

Vassilios S. Verykios, Elias N. Houstis, and John R. Rice
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907-1398, USA.

November 5, 1998

Abstract

In this paper we define a knowledge discovery in databases (KDD) methodology to automatically generate metadata (i.e., knowledge rules) from software/machine pair performance databases. This metadata can be used to characterize the computational behavior of various classes of software or machines. The core and the most computationally intensive part of the KDD methodology is the *data mining* phase which identifies "interesting" patterns from the performance data. The discovery patterns are expressed in a high level representation to be used to summarize and predict the computational behavior of the targeted software/machine. This paper presents an implementation and evaluation of the proposed KDD process for a class of scientific software together with three data mining algorithms (ID3, HOODG, and CN2). For this case study we have selected a set of software that implements the "mesh/grid partitioning" phase of the domain decomposition approach used for the parallel processing of partial differential equation (PDEs) computations. The raw performance database is generated from a population of elliptic PDEs and PELLPACK [HRW+98] solvers by varying the PDE domain, mesh, and domain partitioning (DP) algorithm. The goal of the KDD process here is to evaluate the performance of PELLPACK, CHACO [HL95c], METIS [KK95c], and PARTY [PD96] algorithms/software. This case study shows that (a) the three data mining algorithms used are qualitatively and quantitatively equally effective, (b) the knowledge discovered for the DP algorithms by this KDD process is quantitatively similar to that deduced by purely experimental observations [VH97], and (c) the KDD process is not limited by the size of the performance data and its dimensionality.

1 Introduction

Selecting and tuning software for a specific application to achieve a user's objectives on a target machine is part of every computational endeavor with constraints on time and and cost resources. Performance evaluation of software/machine pairs is currently used for achieving this task for which several measurement and visualization tools are available. They are designed for real or simulated architectures and are capable of playing back or visualizing the computations. The purely visualization approach used by tools such as the one described in [HF91], has limited utility in massively parallel systems and integrated life cycle simulations. This is because the inference, domain parameter estimation, and scalability issues are left to be resolved by the performance evaluator and system designer.

Alternatives to this approach are techniques that attempt to *mine* the knowledge from the performance data stored in a relational database system without significant user interaction. However, making effective and accurate decisions directly from such data is difficult for two reasons: a) these databases are frequently very large, and b) the database is usually described using low level concepts and relations that are cognitively distant from the concepts used by the decision maker.

*This work was supported in part by DARPA contract N66001-97-C8533, NSF grant 9202536, and a Purdue Research Foundation special initiation award.

In this paper we define a knowledge discovery in databases approach to evaluate the performance of scientific software on a class of machine architectures. Moreover, our implementation of this KDD process [Hou98] avoids some of the shortcomings observed above. Its data mining phase explores the contents of the database to automatically identify concepts and rules supported by the data. In addition, the rules discovered can be tested on different databases so that their predictive ability and generality can be evaluated and the rules can be edited before they are incorporated into the performance knowledge base (KB) being developed.

This paper is organized as follows. Section 2 defines a knowledge discovery process for the performance evaluation of scientific software and related work. Moreover, it presents three data mining algorithms in terms of the selected scientific software. In Section 3 we describe the scientific software selected and discuss the generation of the performance data used. The case study for the KDD process uses four major libraries of DP algorithms. Section 4 discusses the implementation of the proposed KDD approach on the performance database of the case study and presents the knowledge structures generated by the three data mining algorithms considered. In Section 5 we present an analysis of the effectiveness of the KDD process and our implementations of its various phases in the context of the DP case study. Finally, Section 6 summarizes our observations and Appendix A provides a description of the particular DP software used in this case study.

2 A Knowledge Discovery Approach for Scientific Software

In general, raw data contain information useful for decision support or exploration and understanding of the phenomena governing the data source [FHS96]. Traditionally, analysis of data has been a strictly manual process. One or more analysts would become intimately familiar with the data and – with the help of statistical techniques – would provide summaries and generate reports. However, such an approach rapidly breaks down as the quantity of data grows and the number of dimensions increases. To automate this process, especially for databases containing performance data of scientific software, various other methodologies have been proposed.

A systematic approach to the performance evaluation of PDE solvers was implemented in [BRH79] based on what is called today data mining techniques. Subsequently, it was used to complete several systematic performance studies of various classes of PDE solvers. The data mining approach used a non-parametric statistical technique to derive the profiles of various solvers selected for evaluation. In turn these profiles are used to rank solvers with respect to various criteria. The use of these profiles to derive knowledge rules (metadata) for applicable solvers on "similar" PDE problems with known solutions was first realized in the ATHENA expert system [HHR+91]. In this system the inference was implemented using an exemplar or collaborative filtering (i.e., leveraging similarities between an a priori defined population of problems and the user defined one) to make recommendations. This system was further advanced in the PYTHIA project [WHR+96] where other data mining techniques were explored such as Bayesian and neural network methods. In addition, the system GOLEM [MF90] was integrated into PYTHIA to implement the *inductive logic programming* (IPL) knowledge discovery approach. GOLEM is applied to derive automatically rules from the rankings of methods. The inference engine of PYTHIA expert system was implemented by CLIPS [Gia91] system. The use of neuro–fuzzy techniques in the context of PYTHIA system was explored in [JWR+96]. Finally, in the series of conferences on *expert systems for scientific computing* [HRV90, HRV92, HRV94], several expert systems are presented for various scientific software.

In this paper, we use the *knowledge discovery* in *databases* (KDD) approach ([FPSS96]) which can be considered as an extension of the PYTHIA methodology by utilizing database technologies at all phases plus using "symbolic" data mining techniques. The implementation of this KDD methodology is described in [Hou98]. In this paper, we define the KDD process for scientific software and implement the data mining phase with a new class of "heuristic" algorithms. The steps or phases that characterize this process are described in Table 1. Figure 1 gives a schematic view of the process defined in Table 1. The third column of Table 1 lists the implementation choices in the PYTHIA II system [Hou98].

Note that the core and the most computational intensive part of KDD is the *data mining* step that identifies interesting patterns from performance databases. The rest can be viewed as pre- or post-processing steps applied by the user or the knowledge engineer. In Table 2, we present a list of various data mining operations applicable in the context of the KDD process along with several techniques for their implementations, see [FPSS96].

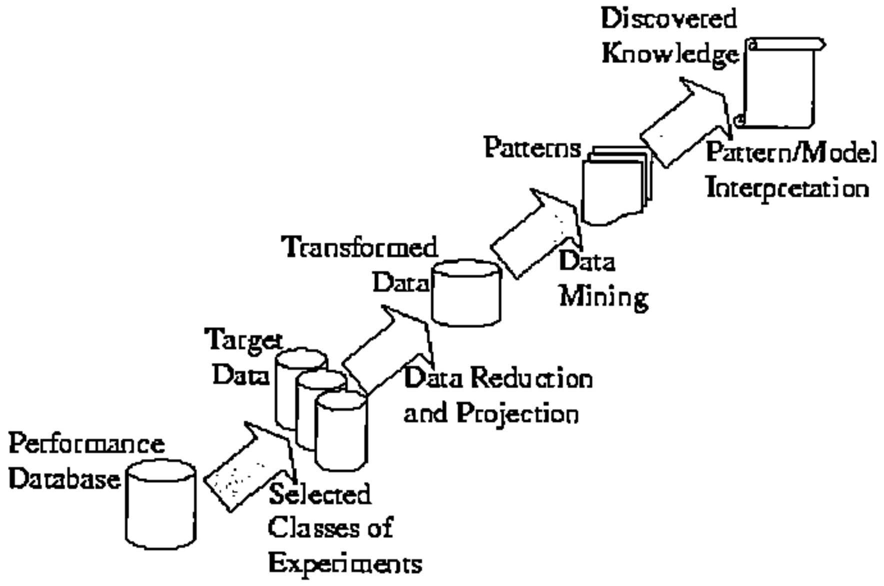
Discovered Knowledge
Pattern/Model Interpretation
Patterns
Data Mining
Transformed Data
Data Reduction and Projection
Target Data
Selected Classes of Experiments
Performance Database

Figure 1: Schematic of the KDD process.

Phases	Description	Implementation
Determine evaluation objectives	Identify the computational objectives with respect to which the performance evaluation of the selected scientific software is carried out.	Manually
Data preparation (1) selection (2) pre-processing	(1) Identify the benchmark applications (i.e., population of scientific problems for the generation of performance data). (2) Identify the performance indicators to be measured. (3) Identify the actual software to be tested together with the numerical values of their parameters. (4) Generate performance data.	POSTGRES95 SQL Tcl/Tk
Data transformation	(1) Transform the data into an analytic or summary form (2) Model the data to suit the intended analysis and the data formats required by the data mining algorithms.	In-house statistical software
Data mining	Mine the transformed data to identify patterns or fit models to the data; this is the heart of the matter and has been mostly automated.	Golem
Analysis of results	This is a post-processing phase done by knowledge engineers and domain experts to ensure correctness of the results.	Manually
Assimilation of knowledge	Create an intelligent interface to utilize the knowledge and to identify the scientific software (with parameters) for user's problems and computational objectives.	CLIPS

Table 1: The steps of the KDD process for scientific software.

Predictive Modeling	Database Segmentation	Link analysis	Deviation Detection
Classification	Clustering	Association discovery	Visualization
Regression		Sequential pattern discovery	Statistics
Summarization			
Value prediction			

Table 2: Data mining operations and their corresponding techniques

In this study we consider three classification techniques for predicting the performance of software with implementations available in the MLC++ [KSD96] software library and the CN2 [CN89] program. The classifiers we select are CN2 [CN89] (using rule-induction) ID3 [Qui86] (using decision trees) and HOODG [Koh94] (using graphs). We describe these algorithms in terms of the case study DP software and its performance database.

CN2 generates an ordered list of simple, propositional like classification rules such as "if-then" statements where the conditional part consists of a propositional expression (i.e., a conjunction of tests on the performance metrics). The consequent part of the rule is the DP algorithm. CN2 constructs the rules iteratively one at a time. A rule is constructed by searching for a conditional part that covers a large number of performance records of a particular DP algorithm and few records of other DP algorithms. Having found a "good" rule, the algorithm removes those performance records from the input covered by the rule and adds the current rule to the end of the list of rules. The best conditional part in an input set is found using a beam search (i.e., the construction of a search graph, where at each iteration the graph is extended the *beam width*, a mostly user specified limit on the leaves). CN2 estimates the quality of a rule from its accuracy and significance. CN2 uses the information theoretic entropy measure (the same used by ID3) to evaluate the accuracy of the rule and the likelihood ratio statistic to evaluate the statistical significance of the rule. This is done by locating a regularity among the data that is unlikely to have occurred by chance [CN89].

ID3 [Qui86] constructs a decision tree from records (each one corresponding to a DP algorithm) whose fields contain measurements of the performance metrics. In the generated decision tree, (a) every internal node is labeled with the name of one of the performance metrics used, see Table 4 in Section 3.2.2 for those of the DP case study, (b) the branches coming out from an internal node are labeled with values (or ranges of values) of the performance metric in that node, and (c) every leaf node is labeled with a DP algorithm. ID3 algorithm uses a top-down irrevocable strategy (i.e., an operation is selected and applied irrevocably without provision for later reconsideration) that searches only part of all applicable to the performance database decision trees, guaranteeing that a simple – but not necessarily the simplest – tree is found. A simple decision tree can be generated by selecting a metric which minimizes the information needed in the resulting subtrees to differentiate subsets of performance data based on their corresponding DP algorithms.

HOODG [Koh94] represents the underlying knowledge as a decision graph. This graph is directed and acyclic with the following properties: a) there are exactly as many nodes in the graph that have out-degree zero as the number of DP algorithms, b) the nodes that do not belong in a) are labeled by some performance metric and their out-going edges are labeled by a distinct value (or range of values) from the domain of values of the performance metric, and c) there is one distinguished node which is called root, that is the only node with in-degree zero. In case each performance metric occurs only once along any path, then the graph is called *read-once*. If all the nodes at a given level are labeled by the same metric, then the graph is called *oblivious*. An oblivious decision graph is *reduced* when there are no two distinct nodes at the same level that branch in exactly the same way on the same value. We denote by OODG a reduced oblivious read-once decision graph. HOODG is a hill climbing algorithm for constructing OODGs bottom-up. The algorithm starts from a set of sets $\{A_1, A_2, \ldots\}$ where each set A_i is the set of all performance records for the i-th DP algorithm. The ten DP algorithms for our case study are listed in Table 3.

For a case study using the KDD process and its implementation, we have selected a set of ten domain decomposition software modules from the PELLPACK system used to decompose geometric data structures (i.e., meshes). These decompositions are used in PDE applications to partition the underlying computations for parallel machines.

3 A Performance Database for Domain Partitioning Software

In this section, we describe briefly the software of the case study of the KDD process and list the benchmark computations and metrics utilized to generate the performance database.

3.1 An Overview of the Domain Decomposition Problem and Software

Mapping computations to parallel machines by partitioning the associated discrete geometric data (i.e., meshes, grids) and the underlying computations is standard for solving PDE problems. The mesh/grid partitioning problem is very often reduced to a graph partitioning problem [Chr92] in which the vertices of the graph are divided into a specified number of subsets, such that the load is almost equally distributed among them, and a minimum number of edges join two vertices in different subsets. A partition of the graph into subgraphs leads to a decomposition of the data and/or tasks associated with a computational problem. The subgraphs can then be mapped to the processing elements of the parallel computer.

More formally, let $G = (N, E, W_N, W_E)$ be an undirected graph without self edges or multiple edges from one node to another. N is the set of nodes, E is the set of edges connecting nodes, W_N is the set of node weights and W_E is the set of edge weights. We use the notation $|N|$ (or $|E|$) to denote the number of elements in the set N (or E). We consider a node n in N as representing an independent task to be executed, and the weight w_n as a measure of the cost of task n. An edge e between two nodes i and j in E means that an amount of data W_e must be transferred from node j to node i to complete the overall task. Partitioning G creates a division of N into P disjoint pieces, $N = N_1 \cup N_2 \cup \ldots \cup N_p$, where the nodes in N_i are assigned to the processing element P_i. This partitioning is subject to the optimality criteria below:

- The sums of the weights W_N of the nodes n in each N_i are approximately equal. Consequently, the load is approximately balanced across the processors.
- The sum of the weights W_E of edges connecting nodes in different N_i and N_j are minimized. In other words, the total size of all messages communicated between different processors is minimized.

There are two classes of partitioning heuristics, depending on whether or not we have coordinate information for each node of G, such as (x, y) or (x, y, z) coordinates. Coordinates are usually available if the graph comes from triangulating, or otherwise discretizing a physical domain. For graphs with coordinate information, the partitioning heuristics are *geometric*, in the following sense: assume, first, that there are two coordinates x and y, that is, the node lies on the plane. Typically, such graphs have the additional property that only nodes which are spatially close together have edges connecting them. Then the algorithms seek either a line or a circle that divides the nodes in half, with (about) half the nodes on one side of the line or circle and the rest on the other side. All these algorithms work by trying to find a geometric "middle" of the nodes (there are several definitions of middle) and then pass a line (or plane) or circle (or sphere) through this middle point. These algorithms completely ignore where the edges are; they implicitly assume that edges connect only nearby nodes. If the graph satisfies some reasonable properties (which often arise in finite element meshes), then one can prove that most of these algorithms find good partitions (which means that they split the nodes nearly in half, while keeping the number of edges connecting different sub-domains close to a minimum).

The second kind of graph we want to partition is coordinate free, i.e., we have no identification of a node with a physical point in space. In addition, edges have no simple interpretation, such as representing physical proximity. These graphs require *combinatorial* rather than geometric algorithms to partition them (although combinatorial algorithms may also be applied to graphs with coordinate information). The *Kernighan-Lin* (KL) algorithm takes an initial partition of G into N_1 and N_2 and iteratively improves it by swapping groups of nodes between N_1 and N_2, greedily picking the group to swap that minimizes the number of edge crossings at each step. In practice, the KL algorithm converges quickly to a (local) optimum, if it is given a good initial partition. Most recently, the *spectral partitioning* algorithm has been introduced, based on the intuition that the second lowest vibrational mode of a vibrating string naturally divides the string in half.

Finally, many of the above algorithms can be accelerated by using a *multilevel technique*. The idea is (1) to replace the graph by a coarser graph with much fewer nodes, (2) to partition the coarser graph, and (3) to use the coarse partition as an initial guess for partitioning the original graph. The coarse graph may, in turn, be partitioned by using the same algorithm recursively: a yet coarser graph is partitioned to get a starting guess, and so on.

3.2 The PELLPACK System and Domain Partitioning Software

PELLPACK [HRW+98] is a problem solving environment for solving PDE problems on high performance computing platforms and a development environment for building new PDE solvers or PDE solver components. PELLPACK allows the user to (symbolically) specify a PDE problem, determine the solution algorithms to be applied, solve the problem, and finally analyze the results produced. The problem and solution algorithm are specified in a custom designed high level language, through a complete graphical editing environment. The user interface and programming environment of PELLPACK is independent of the targeted machine architecture and its native programming environment. PELLPACK adopts the "divide and conquer" parallel computational paradigm and utilizes the discrete geometry non-overlapping domain decomposition approach for problem partitioning and load balancing. A number of tools and libraries exist in the PELLPACK environment to support this approach and estimate (specify) its parameters. These include sequential and parallel finite element mesh generators, automatic (heuristic) domain decomposers, finite element and finite difference modules for discretizing elliptic PDEs, and parallel implementations of the sequential ITPACK [KRG82] linear solver library. The parallel libraries have been implemented in both the hosted and hostless node programming modules for several multiprocessor platforms using different native and portable communication libraries. The implementations utilize standard send/receive, reduction, barrier synchronization and broadcast communication primitives from these message passing communication libraries. No particular machine configuration topology is assumed in the implementation. In our measurements we use the MPI [GLS94] portable communication library of the parallel ITPACK implementation. In addition, PELLPACK supports a *reuse-methodology*, referred to as $\mathcal{M}+$, which assumes that the discretization of the PDE problem is made by an existing sequential "legacy" PDE code. PELLPACK then uses a parallel machine to solve the system of discrete equations. The parallel solution of the discretized PDE equations requires a decomposition of the sequentially produced algebraic system. This decomposition can be made either *implicitly* obtained through a decomposition of the mesh or grid structures, or *explicitly* specified by the user. Then, the partitioned system is down-loaded on the parallel machine. This is the most widely used methodology, since it allows for the preservation of the most knowledge intensive part of the code and for speeding up the most computationally intensive part.

In this case study, we evaluate the computational behavior of DP software used in the context of the PELLPACK parallel processing methodology described above. For this, we take various performance measurements from several modules of the parallel ITPACK linear solver library of a selected population of PDE problems on various parallel architectures following the $\mathcal{M}+$ approach. The ten DP algorithms considered are briefly described in Table 3 and the references cited. For the sake of completeness, we describe the entire suite of algorithms provided through the DP libraries incorporated into the PELLPACK system in Appendix A.

Next we describe the benchmark computations and the performance metrics in the database used for the evaluation of the KDD process.

3.2.1 Benchmark Computations and Metrics for the Evaluation of the DP Software

The computational experiments for this study were performed on an nCUBE/2 and a network of workstations. The nCUBE/2, is a 64-node system with 4 MBs memory per node. The network of workstations used consists of 8 Sparc Station 20s (Model 61) with 32 MBs of memory running Solaris 2.4, connected to both a 10 Mbps Ethernet and an ATM switch, running at speeds up to 155 Mbps.

The benchmark computations selected for the evaluation of the DP software listed in Table 3 are used to solve the following two simple 3D elliptic PDE problems:

$$u_{xx} + u_{yy} + u_{zz} = 0 \qquad (1)$$

$$u_{xx} + u_{yy} + u_{zz} = 3e^{x+y+z} \qquad (2)$$

subject to Dirichlet boundary conditions on five domains. The domains are a cylinder, a union of a cube with a cylinder, a cube with a hole, an airplane, and the U-object depicted in Figure 2 ((a) the mesh and (b) rendering of the solution on the mesh).

For the discretization of Eq. 1 and 2, we apply two different software modules implementing the linear finite element method. The first is the VECFEM module (see [GS91]) and the second is the native FEM PELLPACK's software module. The discretized PDE system generated is decomposed, based on a particular

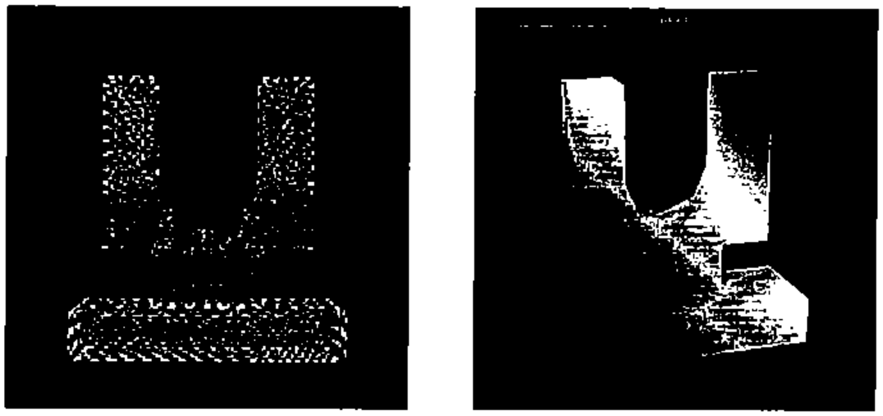
(a) (b)

Figure 2: (a) Visualization for the mesh of the U-Object domain and (b) Rendering of the solution on the mesh

Acronym	Description	Reference
Chaco-Inertial	Inertial Method (from the CHACO library)	[HL95c]
Chaco-Linear	Linear Scheme (from the CHACO library)	[HL95c]
Chaco-Multilevel-KL	Multilevel Kernighan-Lin (from the CHACO library)	[HL95c]
Chaco-Spectral	Spectral partitioning (from the CHACO library)	[HL95c]
Metis-Default	Multilevel graph partitioning (PMETIS) with default values chosen by implementors (from the METIS library)	[KK95c]
Native-Inertial	Partitioning of the mesh along the Inertia axis (from the PELLPACK native domain decomposition library)	[WH96]
Native-PxQxS	Partitioning of the mesh into a specified number of segments along the x-coordinate, y-coordinate, or z-coordinate (from the PELLPACK native domain decomposition library)	[WH96]
Party-Coordinate	The Coordinate Sorting method partitions the mesh along the coordinate that has the widest range (from the PARTY library)	[PD96]
Party-Farhat	The algorithm of Farhat is greedy and utilizes a breadth-first search of the graph, starting from a vertex of minimum degree (from the PARTY library)	[PD96]
Party-Gain	The Gain method is a simple greedy strategy based on adjacency information for constructing a balanced partition (from the PARTY library)	[PD96]

Table 3: The ten domain partitioning algorithms used in the context of $\mathcal{M}+$ parallel reuse methodology and evaluated by the KDD approach

partition of the underlying mesh, and is solved on the selected parallel platform using the $\mathcal{M}+$ approach. The system is solved using the parallel ITPACK Jacobi CG module.

For the mesh generation process, we use the 3D Geompack [MV92] software package, in which the domain is defined as a surface triangulation of a 3D domain, where the user specifies a set of parameters that affect the generated tetrahedral mesh. By varying one of the parameters that relates to the target number of the generated tetrahedrons in the mesh file, we can produce various meshes with an increasing number of vertices that are refinements of an arbitrary initial mesh.

We apply the DP algorithms listed in Table 3 to obtain the mesh partitions (see Figure 3). The performance metrics used in the experiments conducted are also listed in Table 4.

Extensive presentation of the performance evaluation data for this case study can be found in [VH97]. Here we present a subset of these data in order to evaluate the KDD process for scientific software in general and for DP software in particular.

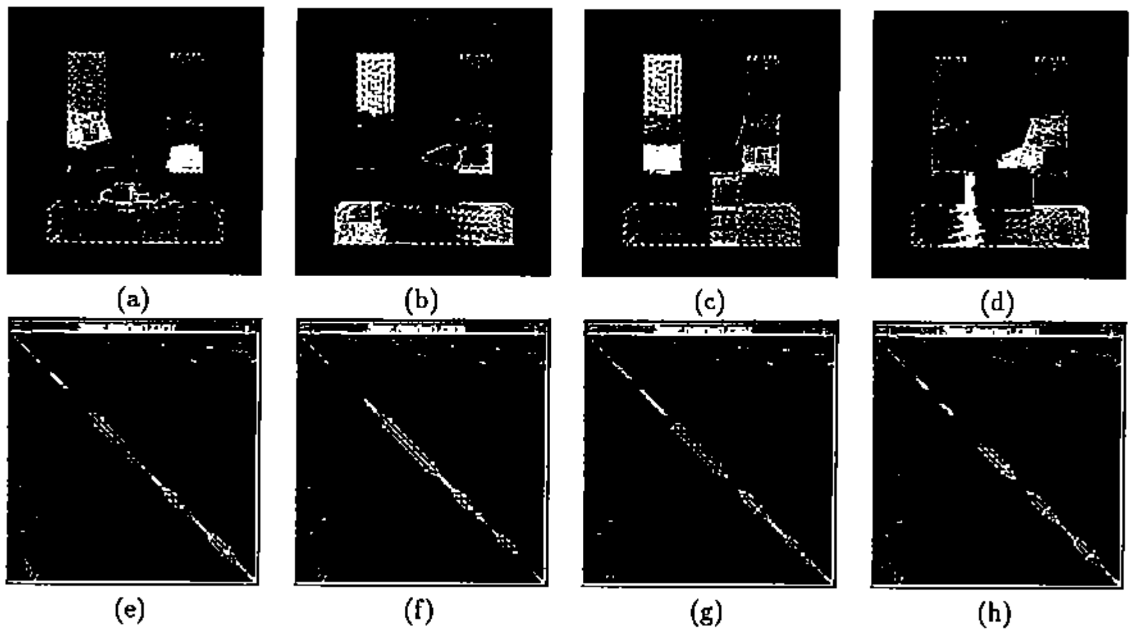

Figure 3: (a), (b), (c), (d) : Domain partitions, and (e), (f), (g), (h) : sparsity patterns of their corresponding linear systems for the Chaco-Inertial, Metis-Default, Native-PxQxS and Party-Coordinate DP algorithms

3.2.2 Performance Metrics for DP software

Several authors [GMT95, KK95b, CHR94] have conducted numerical experiments to compare DP algorithms based on both the quality of the partitions produced (the number of edges cut and the load balancing of the subdomains) and the time taken to compute the partition. Other studies consider the DP problem in the context of the numerical solution of PDEs on multicomputers and use PDE domain dependent metrics to compare the various DP algorithms.

The objectives of all DP algorithms considered in this paper include *load balancing, minimum interface length,* and *minimum subdomain connectivity.* Several formulations of these criteria have been proposed and we adopt the most common one presented in [WH96] where:

1. The *load balancing* is measured by

$$\max(N_i - N_j)/N_n, i = 1, \ldots, N_s, j = 1, \ldots, N_s, i \neq j \tag{3}$$

   where N_i and N_j denote the number of nodes in subdomains i and j, N_s denotes the number of subdomains and N_n the number of nodes in the mesh.

2. The *interface length* is measured by

$$\max \sum_{i=1}^{N_s} C_{i,j}, j = 1, \ldots, N_s, i \neq j \tag{4}$$

$$\text{and} \max C_{i,j}, i = 1, \ldots, N_s, j = 1, \ldots, N_s, i \neq j \tag{5}$$

   where $C_{i,j}$ denotes the number of common edges or nodes between the subdomains i and j.

3. The *subdomain connectivity* is measured by

$$\max \left(N_{b,j} \sum_{i=1}^{N_s} C_{i,j} \right), j = 1, \ldots, N_s, i \neq j \tag{6}$$

   where $N_{b,j}$ denotes the number of neighboring subdomains that subdomain j has.

Acronym	Description	Characterization
Number-of-Subdomains	Number of subdomains generated by the domain partitioning algorithm	Integer, Machine independent
Load-Balancing	The maximum difference in the number of nodes of the subdomains	Integer, Machine independent
Maximum-Interface-Length	The maximum number of outer nodes among all the subdomains	Integer, Machine independent
Subdomain-Connectivity	The maximum number of neighboring subdomains multiplied by the number of outer nodes for each subdomain	Integer, Machine independent
Decomposition-Time	The time in seconds for an algorithm to decompose a mesh	Real, Machine dependent
Maximum-Solving-Time	Maximum time in seconds over all domains solving the linear system	Real, Machine dependent
Speedup	This number shows how many times a parallel configuration is faster than its sequential version (with respect to the solving time)	Real, Machine dependent
Maximum-Decomposition-Solving-Time	The maximum time in seconds over all processors for decomposing and solving the problem	Real, Machine dependent
Maximum-Communication-Time	Maximum time in seconds over all processors spent for communication purposes, during the solution of the linear system	Real, Machine dependent
Maximum-Total-Time	Maximum time in seconds over all processors for loading, solving and saving the solution	Real, Machine dependent

Table 4: Machine independent and machine dependent performance metrics for the evaluation of the DP algorithms.

In addition to the above machine-independent criteria, we use six machine-dependent ones: (1) the time taken to compute the partition, (2) the maximum time taken by a processor, for solving its corresponding part of the linear system, (3) the speedup achieved by the parallel computation, (4) the maximum time taken for both decomposing the entire domain and solving the linear systems that correspond to each one of the subdomains, (5) the maximum communication time over all processors, and (6) the maximum total execution time taken by a processor for initialization, loading of the linear system, solving it and saving the solution. "Maximum time" is considered to be the execution time of the slowest processor needed to run the problem in the parallel computation.

4 Implementation of the KDD Process for the DP Software

The implementation of the entire KDD process defined in Section 2 is described in [Hou98], here we describe only the data mining phase. The performance database for the DP algorithms of Table 3 is organized into records with respect to each geometric domain and DP algorithm considered. Each record consists of measurements of the performance metrics defined in Table 4 for a particular machine configuration (i.e., number of processors). After the generation of the performance database, a selection phase is applied to the database for extracting the relevant information which will be used as input to the data mining algorithms. This phase is implemented in terms of *selection*, *projection*, and *join* queries submitted to the database. The next step of the KDD process is deciding on the data mining operations and their implementation (i.e., algorithms and software).

The applicable data mining operations and techniques are selected based on their ability to satisfy the following requirements: (a) rank the DP algorithms under evaluation with respect to the performance metrics and data, (b) predict the computational behavior of the DP algorithms from the performance database, (c) express the patterns identified in a high level representational form (in order to make it easier for the domain expert to evaluate the generated knowledge), and (d) deal efficiently with both the numerical and the symbolic nature of the metrics involved. These four requirements lead us to select the classification approach for the predictive modeling operation and the so-called *rule induction* [LS95] and decision tree/graph [Qui86, Koh94] classification techniques for its implementation. These classification techniques generate knowledge in a form that is amenable to inspection and interpretation, i.e., knowledge engineers and domain experts can easily understand the underlying decision boundaries in the data and take action based upon them. We have selected the three public domain classifiers CN2, ID3, and HOODG described in Section 2 to implement the data mining phase in this case study.

These data mining algorithms are to identify the patterns that characterize the behavior of the DP algorithms/machines considered with respect to: (1) the performance of these algorithms, (2) the quality of the decomposed mesh, and (3) the performance of parallel PDE solvers that utilize the mesh decompositions generated by the DP algorithms considered. Thus, the data mining algorithm has to search the performance records corresponding to each PDE domain (mesh) and DP algorithm and identify (classify) those DP algorithms for which some of the performance measurements are within certain ranges of values of the performance metrics.

4.1 Application of the Data Mining Algorithms to DP Performance Database

The data mining algorithms ID3, HOODG, and CN2 are given as input 50 records (5 records per DP algorithm, for a total of 10 DP algorithms) that contain measurements of the performance metrics listed in Table 4.

Parts of the decision tree generated by the ID3 algorithm for these data is shown in Figures 4, 5. Each node of the tree contains an integer *condition* vector C of length 10 (since 10 DP algorithms are in our case study). Its i-th component indicates the number of records associated with the i-th DP algorithm for a given domain, for which the values of the performance metrics satisfy all the conditions depicted on the graph edges that belong to the path from the root of the tree leading to every node.

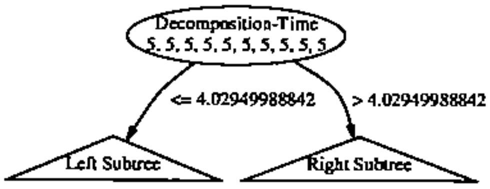

Figure 4: Root node with the left and right subtrees of the decision tree generated by ID3.

A decision tree can be transformed into a rule set by converting each path of the tree into a rule as follows: (a) the internal nodes and their output branches are converted into conditions of the antecedent ("if-part") of the rule; (b) the leaf node is converted into the consequent ("then-part") of the rule. For example, the conversion of the leftmost path of the tree (Figures 4 and 5), from the root to the leaf, reads as follows: *if the decomposition time for an algorithm is less than* 0.46499 *and the maximum interface length is less than or equal to* 308 *nodes (for the particular domain we are considering) then the algorithm that gives these time and length characteristics is the "Native-PxQxS" algorithm.* This transformation is applied to all the paths from the root to the leaves to uncover very "interesting" patterns. Moreover, we observe that ID3 produces decision trees with strong classification. This means that their leaves contain performance records from only one DP algorithm, i.e., only one number from the integer condition vector, which is previously described, is non-zero for all of the leaves. Although this kind of classification is interesting, frequently, it leads to trivial patterns (e.g., leaves that describe only one performance record). It would be of much interest, though, to come up with rules for leaves that cover a maximal number of records, like those that correspond to the two left-most paths, from the root to the leaves (see Figure 5).

A solution to this problem is to use a discretizer as a wrapper to the ID3 decision tree classifier. In essence, a discretization algorithm splits the values of a continuous (integer-valued or real-valued) performance metric into a small list of intervals. This effectively converts continuous metrics into symbolic values (i.e., each resulting interval is regarded as a discrete value of that metric). Figure 6 shows the decision tree produced by the ID3 classifier after applying such a discretization pre-processing step. We observe that the discretization algorithm decomposes the decomposition time variable into five non-overlapping intervals, the maximum interface length into three non-overlapping intervals, and the maximum decomposition plus solving time into three non-overlapping intervals. We also observe that the discretization is consistent for all the nodes that test the values of a particular discretized metric (the out-going edges are consistently labeled with the same ranges of values) such as the maximum interface length in Figure 6. Another important point to notice is that even if the leaves are labeled with only one DP algorithm, they cover performance records

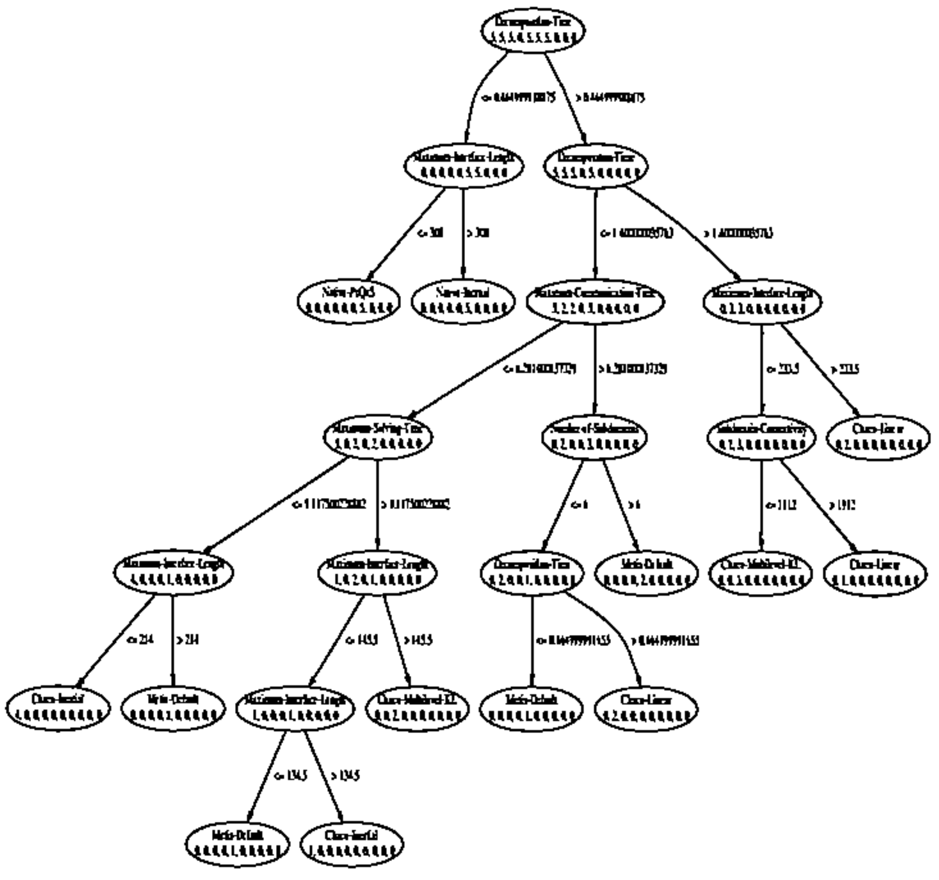

Figure 5: Left subtree of the Decision tree generated by ID3. The condition vector *C* is given at each node.

from certain classes of DP algorithms. This should not be interpreted as a situation where reduced accuracy has occurred, but as an intentional clustering for some of the DP algorithms. For example, the leaf labeled as Chaco-Inertial contains records corresponding to Chaco-Linear, Chaco-Multilevel-KL, and Metis-Default in addition to the Chaco-Inertial records. This is an indication that all of these algorithms behave in a similar manner with respect to the decomposition time and the maximum decomposition plus solving time performance metrics.

The output of the HOODG algorithm applied to the DP performance database is illustrated in Figure 7. In general, during the data collection phase, the analyst collects the maximum possible number of metrics, without knowing in advance whether they are useful for the application domain or not. All the data mining algorithms considered, including the HOODG algorithm, suffer when irrelevant or weakly relevant performance metrics [KSD96] are present. Thus, a selection of a subset of performance metrics should be applied as a pre-processing step to the data mining algorithm. Actually, this step can be applied by the same program that applies the data mining algorithm. This is done primarily for three reasons: (a) to speed up the data mining process, by reducing the total amount of data that needs to be considered, (b) to reduce the number of metrics that the domain expert needs to understand, and (c) to increase the quality of patterns. The patterns induced by applying the HOODG algorithm are similar to those of the decision tree algorithm.

Finally, we analyze the ordered rule list produced by the CN2 induction algorithm which is shown in Figure 8. The patterns in this case are quite understandable, since they are directly expressed as "if-then" statements. The condition vector *C* is also shown next to each one of the rules. One of the disadvantages of this algorithm is that it does not allow discretization. As a result, the splitting of metric values is not consistent throughout the different rules that test the value of the same metric.

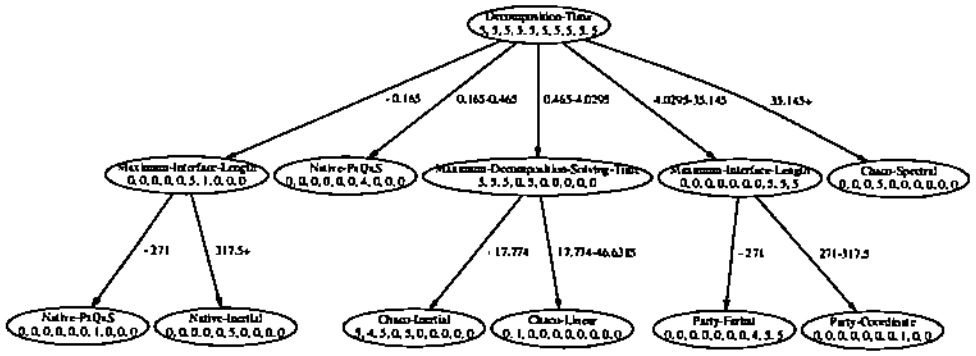

Figure 6: Decision tree generated by ID3 after using a discretization algorithm for the metrics.

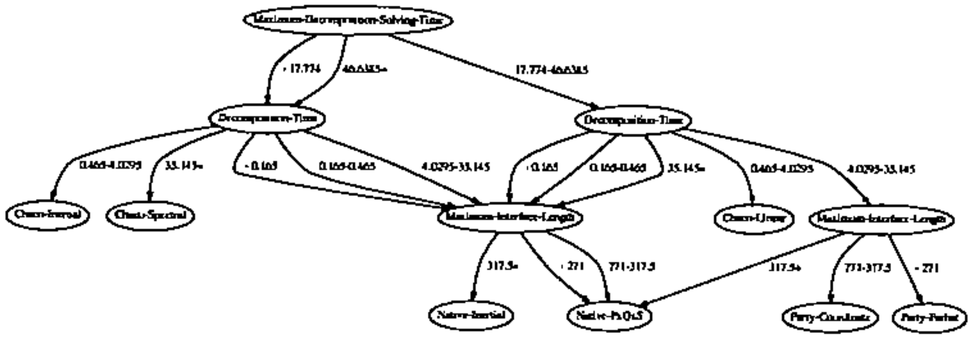

Figure 7: Decision graph generated by HOODG after using a discretization algorithm for the metrics.

```
IF    Maximum-Interface-Length > 317.50
THEN  Algorithm = Native-Inertial  [0 0 0 0 0 5 0 0 0 0]
ELSE IF    Decomposition-Time > 35.15
THEN  Algorithm = Chaco-Spectral  [0 0 0 5 0 0 0 0 0 0]
ELSE IF    Decomposition-Time < 0.46
THEN  Algorithm = Native-PxQxS  [0 0 0 0 0 0 5 0 0 0]
ELSE IF    Subdomain-Connectivity < 1705.00
  AND Decomposition-Time > 6.60
  AND Maximum-Communication-Time < 6.74
THEN  Algorithm = Party-Coordinate  [0 0 0 0 0 0 0 5 1 1]
ELSE IF    Subdomain-Connectivity < 1890.50
  AND Maximum-Decomposition-Solving-Time > 24.79
THEN  Algorithm = Party-Farhat  [0 0 0 0 0 0 0 0 4 0]
ELSE IF    Maximum-Interface-Length < 207.50
  AND Maximum-Decomposition-Solving-Time < 15.23
  AND 0.73 < Maximum-Communication-Time < 6.55
THEN  Algorithm = Chaco-Inertial  [5 0 0 0 0 0 0 0 0 0]
ELSE IF    Maximum-Interface-Length < 234.50
  AND Decomposition-Time < 1.39
  AND Maximum-Solving-Time < 14.51
THEN  Algorithm = Metis-Default  [0 0 1 0 5 0 0 0 0 0]
ELSE IF    Decomposition-Time < 2.15
  AND Maximum-Communication-Time > 5.95
THEN  Algorithm = Chaco-Linear  [0 5 1 0 0 0 0 0 0 0]
ELSE IF    Decomposition-Time > 4.03
THEN  Algorithm = Party-Gain  [0 0 0 0 0 0 0 0 0 4]
ELSE
(DEFAULT) Algorithm = Chaco-Multilevel-KL  [0 0 3 0 0 0 0 0 0 0]
```

Figure 8: Ordered rule list produced by CN2 induction algorithm.

Range of Values	Symbolic Value
0.0000 – 0.1650	VERY SHORT
0.1650 – 0.4650	SHORT
0.4650 – 4.0295	MEDIUM
4.0295 – 35.145	LONG
35.145–	VERY LONG

Table 5: Discretization of the `Decomposition Time` metric and the corresponding symbolic values.

Range of Values	Symbolic Value
0.0 – 271.0	SHORT
271.0 – 317.5	MEDIUM
317.5–	LONG

Table 6: Discretization of the `Maximum Interface Length` metric and the corresponding symbolic values.

5 Analysis of the Metadata Produced by the KDD Process for the DP Case Study

In this section, we analyze the metadata produced by the KDD process and the data mining algorithms for the case of DP software. First, we discretize the performance metrics as specified in Tables 5, 6 and 7. Symbolic values are given for the selected ranges of metrics. Then we interpret the output of the ID3 classifier. Specifically, the metadata represented by the decision tree of Figure 6 can be summarized as follows:

1. The *Native-Inertial* algorithm produces decompositions with long maximum interface length and very short decomposition time (*fast*) for all the meshes considered. This is consistent with the underlying heuristic used.

2. The *Native-PxQxS* algorithm exhibits similar behavior. This makes us to conjecture that the Native PELLPACK library contains the fastest algorithms considered in this study.

3. The *Inertial*, *Linear*, and *Multilevel-KL* algorithms from CHACO, and the METIS default algorithm all fall under the same category of algorithms. They are characterized by a medium decomposition time (reasonably fast algorithms), and a short maximum decomposition and solving time. This characterization implies that they are not too fast, they sacrifice computation time in the decomposition process, in order to produce high quality partitions of the PDE domain meshes. The *Linear method* takes advantage of the ordering of the mesh nodes produced by the mesh generator, while the local refinement algorithm KL (see Section 3.1) increases the quality of the generated mesh.

4. All the algorithms from the PARTY library are characterized by a long decomposition time and short interface decomposition length. The latter happens, primarily, because the Helpful Set local partitioning algorithm (see Appendix A.4) tries to optimize the interface length quality metric. Note that no local refinement algorithm is used in the algorithms from the Native PELLPACK decomposition library.

Range of Values	Symbolic Value
0.0000 – 17.7740	SHORT
17.7740 – 46.6385	MEDIUM
46.6385 -	LONG

Table 7: Discretization of the `Maximum Decomposition` plus `Solving Time` metrics and the corresponding symbolic values.

5. The *Spectral Method* from CHACO is always characterized by a very long decomposition time. This is primarily due to the cost of computing the eigenvalues of the Laplacian matrix of the graph that corresponds to the mesh (see Section 3.1, and Appendix A.1).

The metadata in Figures 5-8 suggest that the patterns uncovered by both ID3 and HOODG (after preprocessing the data by a discretization technique) and by the CN2 program coincide with the conclusions derived by studying the raw performance data with the "naked eye" approach [VH97]. Both ID3 and HOODG algorithms choose: (1) the time taken to decompose the mesh, (2) the maximum interface length, and (3) the total maximum decomposition and solving time as the most important metrics for comparing and characterizing the DP algorithms. The pruning performed by these classifiers uses, as a consequence, only these metrics for building the metadata models.

We also observe that both ID3 and HOODG identify almost the same patterns. Their main difference is that the graph representation of the metadata induced by HOODG is slightly more condensed (fewer nodes for the same patterns) than the ID3 decision tree classifier. Moreover all the non-leaf nodes at a given level of the graph are labeled by the same metric (this is a result of the oblivious property previously defined). This is consistent with the theoretical behavior of these algorithms.

The information from the ordered rule set produced by CN2 indicates that: a) it uses more metrics to exhibit a pattern, b) the values of the variables identified can be considered as lower and/or upper bounds and can be utilized more effectively if they can be correlated with the ranges of values of the corresponding metrics, and c) the most significant rules (based on the ordering in the rule set) are very similar to those induced by the decision tree and graph classifiers.

6 Conclusions

We have presented a knowledge discovery in databases (KDD) methodology to automatically generate metadata (i.e., knowledge rules) from a performance database associated with the domain partitioning (DP) software integrated in PELLPACK [HRW+98]. This metadata are used to characterize the computational behavior of DP software on two parallel platforms. The *data mining* part of the KDD methodology is implemented by three classification algorithms (ID3, HOODG, and CN2). The performance database for the DP algorithms was generated from a population of elliptic PDE problems and PELLPACK solvers by varying the PDE domain, mesh, and DP algorithm. This case study shows that (a) the three data mining algorithms considered are qualitatively and quantitatively equally effective, (b) the knowledge discovered for DP algorithms by the proposed KDD process is quantitatively similar to that deduced by purely experimental observations [VH97], and (c) the KDD process is not limited by the size of the performance data and its dimensionality. We note that the KDD process and the three data mining algorithms considered are independent of the nature of the performance database. Thus, the process can be adopted for the evaluation of any domain of scientific software to predict its performance over a well defined domain of applications and computational environments.

References

[BRH79] Ronald F. Boisvert, John R. Rice, and Elias N. Houstis. A system for performance evaluation of partial differential equations software. *IEEE Transactions on Software Engineering*, SE-5(4):418–425, 1979.

[Chr92] N. P. Chrisochoides. *On The Mapping of Partial Differential Equation Computations onto Distributed Memory MIMD Parallel Machines.* PhD thesis, Department of Computer Sciences,Purdue University, 1992.

[CHR94] N. P. Chrisochoides, E. N. Houstis, and J. R. Rice. Mapping Algorithms and Software Environments for Data Parallel PDE Solvers. *Special Issue of the Journal of Parallel and Distributed Computing on Data-Parallel Algorithms and Programming*, 21(1):75–95, 1994.

[CN89] P. Clark and T. Niblett. The CN2 Induction Algorithm. *Machine Learning*, 3(4):261–283, 1989.

[FHS96] U. Fayyad, D. Haussler, and P. Stolorz. Mining Scientific Data. *Comm. ACM*, 39(11):51–57, 1996.

[FPSS96] U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From Data Mining to Knowledge Discovery: An Overview. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 1–34. AAAI Press/MIT Press, 1996.

[Gia91] J. C. Giarratano. *CLIPS User's Guide, Version 5.1.* NASA Lyndon B. Johnson Space Center, 1991.

[GLS94] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface.* MIT Press, October 1994.

[GMT95] John R. Gilbert, Gary L. Miller, and Shang-Hua Teng. Geometric Mesh Partitioning: Implementation and Experiments. In *Proc. of 9th International Parallel Processing Symposium*, pages 418–427. IEEE Computer Society Press, 1995.

[GS91] L. Gross and P. Sternecker. *The Finite Element Tool Package VECFEM.* University of Karlsruhe, 1991.

[HF91] M. T. Heath and J. E. Finger. Visualizing the Performance of Parallel Programs. *IEEE Software*, 8(5):29–39, 1991.

[HHR+91] C.E. Houstis, E.N. Houstis, J.R. Rice, P. Varadaglou, and T.S. Papatheodorou. Athena: A Knowledge Based System for //ELLPACK. In E. Diday and Y. Lechavallier, editors, *Symbolic–Numeric Data Analysis and Learning*, pages 459–467. Nova Science, 1991.

[HL93] B. Hendrickson and R. Leland. An Improved Spectral Load Balancing Method. In Sinovec et al., editor, *Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing*, pages 953–961. SIAM Pub., Philadelphia, 1993.

[HL95a] B. Hendrickson and R. Leland. An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations. *SIAM J. Sci. Computing*, 16(2):452–469, 1995.

[HL95b] B. Hendrickson and R. Leland. A Multilevel Algorithm for Partitioning Graphs. In *Proc. Supercomputing '95.* ACM, 1995.

[HL95c] B. Hendrickson and R. Leland. *The Chaco User's Guide, Version 2.0.* Sandia National Laboratories, July 1995.

[Hou98] E. N. Houstis. PYTHIA II: A Recommender System for High Performance Software/Hardware Artifacts. Technical Report TR-98-031, Dept. Comp. Sci., Purdue University, 1998.

[HRV90] E. N. Houstis, J. R. Rice, and R. Vichnevetsky, editors. *Intelligent Mathematical Software Systems.* North–Holland, 1990.

[HRV92] E. N. Houstis, J. R. Rice, and R. Vichnevetsky, editors. *Expert Systems for Scientific Computing.* North–Holland, 1992.

[HRV94] E. N. Houstis, J. R. Rice, and R. Vichnevetsky, editors. *Third International Conference on Expert Systems*, volume 36 of Special triple issue of. *Math. Comp. Simulation*, 267-503, 1994.

[HRW+98] E.N. Houstis, J.R. Rice, S. Weerawarana, A.C. Catlin, M. Gaitatzes, P. Papachiou, and K. Wang. Parallel ELLPACK: A Problem Solving Environment for PDE Based Applications on Multicomputer Platforms. *ACM Trans. Math. Soft.*, 24(1):30–73, 1998.

[JWR+96] A. Joshi, S. Weerawarana, N. Ramakrishnan, E.N. Houstis, and J.R. Rice. Neuro–Fuzzy Support for PSEs: A Step Toward the Automated Solution of PDEs. *IEEE Computational Science and Engineering*, vol.3(1):44–56, 1996.

[KK95a] G. Karypis and V. Kumar. Analysis of Multilevel Graph Partitioning. Technical Report TR 95-037, Department of Computer Science, University of Minnesota, 1995.

[KK95b] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. Technical Report TR 95-035, Department of Computer Science, University of Minnesota, 1995.

[KK95c] George Karypis and Vipin Kumar. *METIS Unstructured Graph Partitioning and Sparse Matrix Ordering.* University of Minnesota, Department of Computer Science, August 26, 1995.

[Koh94] Ron Kohavi. Bottom-up Induction of Oblivious Read-Once Decision Graphs: Strengths and Limitations. In *Proc. 12th National Conference on Artificial Intelligence*, pages 613–618, 1994.

[KRG82] D. Kinkaid, J. Respess, and R. Grimes. Algorithm 586: Itpack 2c: A Fortran Package for Solving Large Linear Systems by Adaptive Accelerated Iterative Methods. *ACM Trans. Math. Soft.*, 8(3):302–322, 1982.

[KSD96] Ron Kohavi, Dan Sommerfield, and James Dougherty. Data mining using MLC++: A machine learning library in C++. In *Tools with Artificial Intelligence.* IEEE Computer Society Press, 1996. `http://www.sgi.com/Technology/mlc`.

[LS95] P. Langley and H. A. Simon. Applications of Machine Learning and Rule Induction. *Comm. ACM*, 38(11):55–64, 1995.

[MF90] S. Muggleton and C. Feng. Efficient Induction of Logic Programs. In S. Arikawa, S. Goto, S. Ohsuga, and T. Yokomori, editors, *Proceedings of the First International Conference on Algorithmic Learning Theory*, pages 368–381. Japanese Society for Artificial Intelligence, Tokyo, 1990.

[MV92] S. A. Mitchell and S.A. Vavasis. Quality Mesh Generation in Three Dimensions. In *Proc. ACM Computational Geometry Conference*, pages 212–221. ACM Press, 1992.

[PD96] R. Preis and R. Diekmann. *The PARTY Partitioning-Library, User Guide - Version 1.1.* University of Paderborn, Sep. 1996.

[Qui86] J. R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, 1986.

[VH97] V. S. Verykios and E. N. Houstis. Parallel ELLPACK 3D Problem Solving Environment. Technical Report TR-97-028, Dept. Comp. Sci., Purdue University, 1997.

[WH96] P. Wu and E. N. Houstis. Parallel Adaptive Mesh Generation and Decomposition. *Engineering with Computers*, 1(12):155–167, 1996.

[WHR+96] S. Weerawarana, E.N. Houstis, J.R. Rice, A. Joshi, and C.E. Houstis. PYTHIA: A Knowledge Based System to Select Scientific Algorithms. *ACM Transactions on Mathematical Software*, 22(4):447–468, 1996.

[Wu95] Poting Wu. *Parallel Electronic Prototyping of Physical Objects.* PhD thesis, Department of Computer Sciences, Purdue University, 1995.

A PELLPACK Domain Partitioning Software

A.1 Native PELLPACK Decomposition Software

The native PELLPACK decomposition algorithms are described extensively in [WH96, Wu95]. Here, we present a brief overview. The PxQxS algorithm splits the domain along the main axis (cartesian or polar), after sorting the coordinates of the nodes (FD mode), or the center of the mass of the elements (FEM mode). The Inertia Axis algorithm keeps splitting the domain into subdomains along the symmetry axis defined by the coordinates of the nodes (FD), or the center of mass of the elements (FEM), until the specified number of subdomains is reached. The Recursive Spectral Bisection algorithm uses eigenvector based spectral search. Nodes are visited in order of increasing eigenvector values of the Laplacian matrix of the graph. The Neighborhood Search algorithm splits the initial mesh based on a neighborhood traversal scheme. Thereafter, the subdomains are gathered on the basis of the searching order defined. The objective of a data clustering algorithm is to group the mesh points into clusters, such that the points within a cluster have a high degree of "natural association" among themselves, while the clusters are relatively "distinct" from each other.

A.2 CHACO Graph Partitioning Software

The CHACO 2.0 version of CHACO [HL95c, HL93, HL95a, HL95b] library integrated into PELLPACK addresses three classes of problems. First and foremost, it partitions graphs using a variety of approaches with different properties. Second, it intelligently embeds the partitions it generates into several different topologies. The topologies the code considers are those of the common parallel architectures, namely hypercubes and meshes. Third, it can use spectral methods to sequence graphs in a manner that preserves locality. The five classes of partitioning algorithms currently implemented in CHACO are simple, spectral, inertial, Kernighan-Lin(KL) and multilevel-KL. Each of these algorithms can be used to partition the graph into two, four or eight sets at each stage of a recursive decomposition. KL is considered to be a local refinement technique, while the other methods are global partitioning methods. CHACO allows the output of any of the global methods to be fed into a local method. It also allows a partition to be read from a file and be refined with a local method.

A.3 METIS Unstructured Graph Partitioning

The METIS 2.0 version of METIS software library [KK95c] is a set of programs that implement the various algorithms described in [KK95a, KK95b]. The basic idea is to use the multilevel graph partitioning, that is the graph G is first coarsened down to a few hundred vertices, then a bisection of this much smaller graph is computed, and finally this partition is projected back towards the original graph (finer graph), by periodically refining the partition. The coarsening phase is accomplished by finding a maximal matching and by collapsing together the vertices that are incident on each edge of the matching. The next phase of the multilevel algorithm is to compute a minimum edge-cut bisection of the coarse graph, such that each part contains roughly half of the vertices of the original graph. Since, during coarsening, the weights of the vertices and edges of the coarser graph are set to reflect the weights of the vertices and edges of the finer graph, the graph from the coarsening process contains sufficient information to intelligently enforce the balanced partition and the minimum edge-cut requirements. During the last phase, the partition of the coarsest graph is projected back to the original graph by going through the sequence of graphs created during the coarsening process. Since the projected graph is finer, it has more degrees of freedom that can be used to further improve the partition, and thus, decrease the edge-cut. Hence, it may still be possible to improve the projected partition by local refinement heuristics.

A.4 PARTY Partitioning Library

The PARTY partitioning library [PD96] tries to solve the graph partitioning problem, and its goal is (a) to provide the user with several partitioning methods of different character, (b) efficient implementations to guarantee high performance, (c) a variety of generalization of the methods to reflect the specific constraints of the application more precisely, and (d) a simple interface which guarantees an easy to use as well as a strong control over the methods. Each partitioning algorithm in the PARTY library, as a first step, applies a global heuristic to construct an initial partition π_1. In the second step, a local heuristic can be applied to construct a partition π_2 from the partition π_1. We list the global methods implemented in PARTY. The *Optimal* (OPT) method produces an optimal result, but has an exponential run time behavior. The heuristic methods *Linear* (LIN), *Scattered* (SCA) and *Random* (RAN) are exclusively depending on the vertex list, whereas the *Gain* (GAI) and *Farhat* (FAR) methods take the adjacency information into account. Coordinate information is needed by the *Coordinate Sorting* (COO) method. Although a global partitioning method already produces a balanced partition, local methods try to improve the cut size. Most local partitioning methods were originally developed for improving graph bisections. The two local partitioning methods in PARTY are: the

Kernighan-Lin heuristic and the *Helpful-Set* method which is based on local rearrangements. The PARTY library offers the option of either partitioning the graph into two parts and then applies recursively the same procedure (recursive mode) until the required number of partitions has been reached, or partitionins the graph into the specified number of parts in one step (direct mode).