Scientific Programming 15 (2007) 121–136
IOS Press

Causal analysis for performance modeling of computer programs

Jan Lemeire*, Erik Dirkx and Frederik Verbist
ETRO Department, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium

Abstract. Causal modeling and the accompanying learning algorithms provide useful extensions for in-depth statistical investigation and automation of performance modeling. We enlarged the scope of existing causal structure learning algorithms by using the form-free information-theoretic concept of mutual information and by introducing the complexity criterion for selecting direct relations among equivalent relations. The underlying probability distribution of experimental data is estimated by kernel density estimation. We then reported on the benefits of a dependency analysis and the decompositional capacities of causal models. Useful qualitative models, providing insight into the role of every performance factor, were inferred from experimental data. This paper reports on the results for a LU decomposition algorithm and on the study of the parameter sensitivity of the Kakadu implementation of the JPEG-2000 standard. Next, the analysis was used to search for generic performance characteristics of the applications.

Keywords: Performance modeling, modeling techniques, performance attributes, smoothing, graph algorithms, multivariate statistics, machine learning

1. Introduction

The design and implementation of performance critical computer applications requires knowledge of many interdependent factors. This task can only be done with the support of computer aided performance analysis. Multifunctional performance models result in deeper insight in the dynamic interactions between various hardware and software resources. The models should fulfill many requirements. They should provide information on the expected performance, offer insight in the causes of performance degradation, should be constituted of reusable submodels, tell which variables should be known for predicting others, should make it possible to estimate the effects of optimizations and enable to reason under uncertainty. Causal models offer an elegant formalization of these properties.

Causal models intend to graphically describe the structure of the underlying physical mechanisms governing a system under study. Causal analysis enables a decomposition into independent submodels. Each variable is determined by its direct causes. Causal theory developed by Pearl is based on the property that a causal structure is observed by the conditional independencies it entails. Basically, the theory is based on a dependency analysis, as expressed by the *Markov property*. For model $A \rightarrow B \rightarrow C$, variable A affects C, but B 'screens off' the influence, since A becomes independent from C by *conditioning* on B, written as $A \perp\!\!\!\perp C|B$. If the state of B is known, learning A's state offers no additional information about C. In absence of deeper explanations, one can say that B is the direct cause of C.

Causal structure learning algorithms are able to construct models based on the independencies found in the experimental data and background knowledge provided by an expert. However, real performance data are more complex than data typically encountered in research about causal analysis. They contain a mixture of continuous and discrete variables. The relationships between the variables are not always linear and stochastic as assumed in most research. Our generalized approach eliminates both constraints. All these aspects

*Corresponding author. E-mail: jan.lemeire@vub.ac.be.

ISSN 1058-9244/07/$17.00

are handled by the extensions we developed and integrated into the TETRAD tool, developed by the Dept. of Philosophy of Carnegie Mellon University [31]. We integrated the causal analysis in a tool, called EPDA, also comprising other statistical techniques such as regression analysis, complexity analysis and kernel density estimation.

This paper reports on the benefits of causal analysis in the performance modeling process. We investigated how the dependency analysis, the causal decomposition and the Markov property can help model construction. A causal model presents a qualitative model revealing how variables affect one another. The dependency analysis, on which the model is based, allows the verification of independency assumptions, which are implicitly present in any model. After causal decomposition, a regression analysis can be applied on the submodels $X = f(direct\ causes\ of\ X)$ to turn the model into a quantitative one. The Markov property enables to validate performance characteristics of the application, quantities that aim at fully characterizing the performance behaviour so that they can be used to explain and predict performance. The requested models should be of the form

$$application \rightarrow characteristics \rightarrow performance \quad (1)$$

The *characteristics* should hold all performance information of the application, which is captured by the independence $application \perp\!\!\!\perp performance \mid characteristics$.

The next section gives an overview of related work, causal models are defined in Section 3 and causal performance models in Section 4. Section 5 explains the structural learning algorithms, Section 6 the implemented extensions and Section 7 the statistical techniques used in the modeling process. The experimental results are presented in Section 8.

2. Related work

Many tools exist for automated performance analysis. They are integrated in frameworks for coordinated monitoring and control of computer applications. It is widely recognized that the complexity of deployed systems surpasses the ability of humans to diagnose and respond to problems rapidly and correctly. Research on automated diagnosis and control, beginning with tools to analyze and interpret instrumentation data, should provide the means to guide the developer and user with understandable information.

Our research focuses on dependency analysis, when the model is not a priori known. The most common approach is to incorporate a priori models, which explicitly or implicitly represent how variables relate to each other. Other approaches let the user himself discover the interrelational structure incrementally. Current tools that support multiple experiment analysis plot performance variables (SCALEA [34]) and inefficiencies (Aksum [9]) as a function of application and system parameters. Others additionally provide a regression analysis (AIMS [37]). Several tools allow automatic bottleneck detection. Examples are Kojak [21] or Paradyn [15]. Our approach adds the facility to find a priori unknown relations and to reason with the relational information.

Most performance monitoring tools fall under two categories: one collecting statistical data (concerned with counts and durations), the other event traces (the exact sequence of actions that took place during a run are recorded). Statistical data is more compact then that of event traces, but the predictive power is limited [5]. Our approach is only applicable for the first category. It is a multivariate analysis that tries to characterize the relations among the data. The difference with current work, such as PMaC [28] or PERFORM [12], is that they work with relational structures that are a priori chosen or have to be configured manually.

Causal models are not widely used for performance analysis yet. Cohen and Chase use Tree-Augmented Bayesian Networks (TANs) to identify combinations of system-level metrics and threshold values that correlate with high-level performance states in a three-tier Web service under a variety of conditions [6].

Most research on causal learning does not consider models that contain such a wide variety of variables and relations as encountered in real performance models. They focus on one type of variables, discrete or continuous, where the continuous are most often expected to be quasi-linearly related with Gaussian disturbances [30]. In case of deterministic relationships, one should exclude variables from the dataset that are definable in terms of other variables in the set [27]. We will argue that deterministic variables contain valuable information, worth of being added to the model.

3. Causal models

This chapter will briefly introduce causal models, see [25,30,33] for a complete theoretic elaboration. Causal models intend to graphically describe the struc-

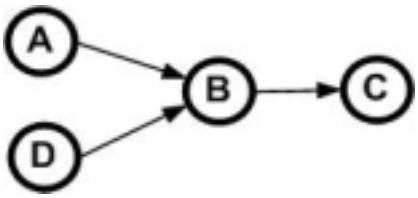

Fig. 1. Example causal model.

ture of the underlying physical mechanisms governing a system under study. Figure 1 shows a causal model with 4 variables and their causal relations.

The model breaks up into the submodels $P(X_i \mid parents(X_i))$, where, for each variable X_i, the conditional probability distribution (CPD) describes the information we have about X_i when knowing the parents of node X_i in the graph. Each submodel represents a stochastic process by which the values of X_i are chosen in response to the values of $parents(X_i)$. The stochastic variation of this assignment is assumed independent of the variations in all other assignments. Modularity results in autonomy: a submodel can be altered without affecting the rest of the model. Causality thus inherently provide autonomous submodels that can be reused when subsystems are used in different contexts.

Each assignment process remains invariant to possible changes in assignment processes that govern other variables in the system. This modularity assumption enables us to predict the effect of *interventions*. Interventions are described as specific modifications of some submodels $P(X_i \mid parents(X_i))$. A causal model enables not only the prediction of unknown variables, but also the effect of local changes made to the system. They are therefore extremely useful in engineering or decision-making, as in economics. In fact, models used in engineering are implicitly based on the properties of causality. Consider the digital circuit of Fig. 2, the scheme makes it possible to predict the output when the input nodes are set to a certain value. But it also allows reasoning of how parts of the component should be changed to attain a certain desired transfer function.

The causal structure, represented by graph G, can be observed by the conditional independencies. From model $A \rightarrow B \rightarrow C$ (a Markov chain), it follows that A is independent from C by conditioning on B. This is called the *Markov property*. A conditional independency is denoted by the ternary operation $.\perp\!\!\!\perp. \mid .$ and is defined as follows

$$A \perp\!\!\!\perp C \mid B \Leftrightarrow P(A \mid B, C) = P(A \mid B) \qquad (2)$$

The conditional independency implies that learning the value of C does not provide additional information about A once we know B. Note that independencies are qualitative properties, despite the fact that the definition is based on an equality of numerical quantities. People can easily and confidently detect dependencies, even though they may not be able to provide precise numerical estimates of the probabilities [24].

The *Causal Markov Condition* describes the independencies that follow from a causal structure. It states that a variable becomes independent from all its non-descendants by conditioning on its direct causes. Pearl developed a graphical criterion, called d-separation, to retrieve all independencies that follow from the Markov condition.

Definition 1. (d-separation) Let p be a path between a node X and a node Y of a DAG G. Path p is called *blocked* given subset $\boldsymbol{Z}$ of nodes in G if there is a node w on p satisfying one of the following conditions:

1. w has converging arrows (along p) and neither w nor any of its descendants are in $\boldsymbol{Z}$, or
2. w does not have converging arrows (along p) and w is in $\boldsymbol{Z}$.

$\boldsymbol{Z}$ is said to *d-separate* X from Y in G, denoted $X \perp Y \mid \boldsymbol{Z}$, iff they block every path from X to Y.

In the model of Fig. 1, A and C get d-separated by B; D and C by B. A and D are d-separated, but are not d-separated if B is given. $A \rightarrow B \leftarrow D$ is called a *v-structure*. Conditioning unblocks a v-structure in a path whereas it blocks non-v-structures.

The DAG G together with the conditional distributions $P(node_i \mid parents(node_i))$ gives a dense description of a *joint probability distribution* over all variables. The Directed Acyclic Graph (DAG) of Fig. 1 corresponds to the following factorization:

$$P(A,B,C,D) = P(A).P(D).P(B \mid A, D) \\ .P(C \mid B). \qquad (3)$$

A factorization can be built starting from any variable ordering. Ordering A, B, C, D results in the Bayesian network depicted in Fig. 3. In general, a variable has incoming edges from all other variables that precede it in the variable ordering. Unless there are conditional independencies that can reduce the number of parents, such as independency $A \perp\!\!\!\perp C \mid B$ in Fig. 3. The graph, however, contains more edges as the true model of Fig. 1 and represents less independencies. It nevertheless describes the same distribution. A Bayesian network is adequate for the prediction of unknown variables, but not for reasoning about changes to the system (interventions).

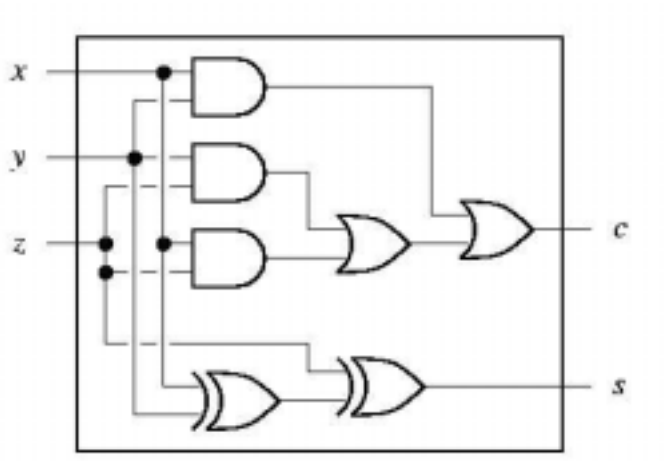

Fig. 2. Digital circuit, a causal model used in engineering.

If all dependencies appearing in the joint distribution can be retrieved from the graph with the d-separation criterion, the graph is called *faithful* to the distribution. This means that

$$X \perp\!\!\!\perp Y \mid \boldsymbol{Z} \;\Leftrightarrow\; X \perp Y \mid \boldsymbol{Z}. \tag{4}$$

Causal models should be faithful to reality. This means that they are able to explain all observable properties of the system. It can be shown that if a faithful graph exists for a distribution, it is the Bayesian network with the minimal number of edges. Since learning is based on searching for the minimal model, this property allows the learning of the causal model based on observational data. This is further discussed in Section 5.

4. Causal models of computer performance

To introduce causal performance models, we will use a simplified performance model of a LU decomposition algorithm. This model is then extended to a more realistic one in Section 5. *LU decomposition* is a matrix decomposition which writes a matrix as the product of a lower and upper triangular matrix [13]. For a 3×3 matrix, this becomes

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{12} & 1 & 0 \\ l_{13} & l_{23} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \tag{5}$$

This decomposition facilitates solving a system of linear equations, $Ax = b$, or finding the inverse of a matrix. The sequential implementation of the algorithm mainly consists of 3 nested *for*-loops with the inner loop having a division, a multiplication and a subtraction of matrix elements.

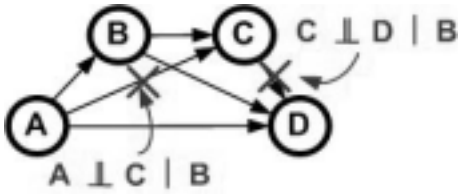

Fig. 3. Factorization based on variable ordering A, B, C, D and independencies $A \perp\!\!\!\perp C \mid B$ and $C \perp\!\!\!\perp D \mid B$.

The performance model shown in Fig. 4 is constructed in an intuitive way, for what the user expects to be a true and useful model. It represents a first-order approximation of the performance for the computation time T_{comp} of a LU decomposition algorithm. $\#op$ is the number of basic operations, defined as the execution of the inner loop. It is determined by the row and column size n. We only consider square $n \times n$ matrices. The *datatype* of the matrix ($float$, $double$, $integer$, ...) influences the number of instructions $\#instr_{op}$ needed to perform 1 basic operation, but also C_{op}, the number of processor cycles for 1 basic operation. Together with the processor's clock frequency f_{clock} and $\#op$ they determine the runtime T_{comp}.

The analysis of the properties attributed to this performance model shows that they correspond to those defining causal models. A variable is expected to be determined by its parents only. This corresponds to the Markov condition. The model is assumed not to contain redundant relations. Hence it is minimal. Furthermore, the paths between the variables show the dependencies among the variables. This corresponds to the faithful-

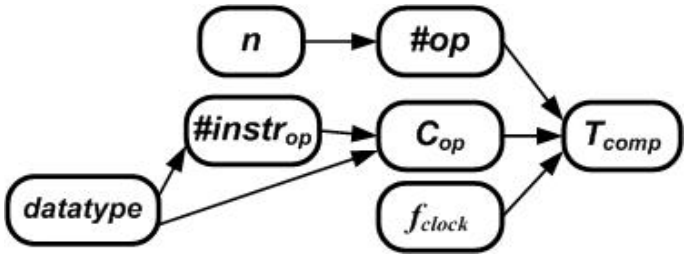

Fig. 4. Simplified causal model of the performance of LU decomposition.

ness condition. Spirtes et al. use these 3 properties – the Markov condition, minimality and faithfulness – as axiomatic foundation of causal models [30]. Causal models explicitly encode relational information. They offer a formalization of a structured representation of the relations among variables. They provide insight into complex situations with many variables and dependencies.

Next, the questions that are supposed to be answered by a performance analysis are of causal nature. Consider the study of network performance: communication delays should be attributed to the different steps of the communication process, such as machine latency, transfer time, network contention, flight time, etc [2]. A correct understanding of the origins of the delays is indispensable. The task of identifying them becomes even more difficult when implementation-specific low level issues come into play, such as specific protocol behavior, window delays or chatter [23], since these are not always fully understood and often cannot be measured directly. Accordingly, the relations between the overall performance quantities observed throughout the application and the computer system have a causal interpretation.

Besides the explanatory facilities, causal models can be exploited to *reason* about the performance and answer questions like: Which part of the application gives space for adequate optimization? What is the most efficient upgrade of the system?

5. Causal structure learning

Besides the formal treatment of causal models by Pearl, another important advance was the design of algorithms for learning causal models from observational data [30,35]. Various tools exist that implement these algorithms, for an overview see [16] or [22]. TETRAD (freely available at [31]) is open source software, written in Java and contains an extensive set of algorithms. There are two basic types of structure learning algorithms: constraint-based and scoring-based. All learning algorithms of TETRAD are of the constraint-based type, from which the *PC algorithm* is the basic one [30]. The algorithm consist of two parts: first it constructs an undirected graph by finding *direct relations*. This part is the same for all constraint-based algorithms. Secondly, the algorithm tries to direct the edges using orientation rules. Besides this, the flexibility of the algorithm allows the insertion of background knowledge. The user can specify edges that are required or forbidden and put constraints on orientations. In performance models, application and system parameters are input nodes that can only have outgoing edges.

The first step of the algorithm, called *adjacency search*, is based on the property that direct relations cannot become probabilistically independent upon conditioning on some other set of vertices (see the Markov condition). Adjacent variables share exclusive information, while indirectly related variables become independent by conditioning on some other variables, which will lie on the path between both variables. The algorithm starts with a full-connected undirected graph and removes all edges for which a conditioning set can be found that renders both variables independent. The algorithm will go through all subsets of variables and check for conditional independencies. If a test is successful, the edge is removed. The algorithm starts by checking unconditional dependencies and then gradually adds nodes to the conditioning set up to a certain maximal number. It selects the nodes in an optimized way to minimize the tests it has to perform. Figure 5 shows this part of the algorithm for learning the model of Fig. 1.

Secondly, the algorithm tries to direct the edges using orientation rules. These rules are based on the detection of *v-structures*. If three variables are connected by two edges, for example $A - B - D$, there are four possibilities to orient both edges. The v-structure is recognized among these, since for $A \rightarrow B \leftarrow D$, A and D are initially independent, but become dependent by conditioning on B. For all three other orientation

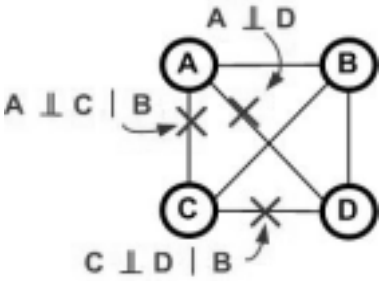

Fig. 5. First part of the learning algorithm: Edges are removed from the full-connected undirected graph by conditional independencies.

possibilities the opposite holds; A and D are initially dependent, but become independent by conditioning on B. Applied on the undirected graph of Fig. 5, v-structure $A \rightarrow B \leftarrow D$ is detected. The $B-C$ relation can be oriented as $B \rightarrow C$, since an opposite orientation leads to v-structure $A \rightarrow B \leftarrow C$ that cannot be confirmed by the independencies found in the data. In absence of enough v-structures, it might be that there is not enough information to direct all edges. Thus, the learning algorithm leads to a set of observationally indistinguishable models, which have the same undirected graph and v-structures. Fortunately, the orientation of the relations in performance models is straight forward in most cases. Knowledge of the input and output variables allows orientation of the edges connected to it. Any further doubt about the orientation of some edges can be resolved by an expert.

The TETRAD manual states that under 6 assumptions the algorithm will find the correct equivalence class of indistinguishable causal models [27]. One assumption is that the experiment should be typical, i.e. random. We will choose the input parameters randomly from a uniform distribution. Another assumption states that the faithfulness condition should hold. This property is violated when there are deterministic relationships among the variables. Performance models contain inevitably deterministic relations. This issue is discussed in Section 6.2. The other 4 assumptions are valid. Moreover, the PC algorithm requires *causal sufficiency*, i.e. that all common causes should be known: variables that are the direct cause of at least two variables. By including all parameters of system and algorithm in the data, all possible common causes are present.

6. Extensions

To capture the complexity of performance data, the existing models and algorithms had to be extended in three ways:

- To handle *a mixture of discrete and continuous data*, we used the information-theoretic concept of mutual information for dependency measurement. It measures the decrease of uncertainty of one variable when observing other variables.
- Mutual information offers a *form-free dependency measure*, whereas the widely used Pearson correlation coefficient measures the closeness of a relation to perfect linearity.
- *Deterministic relations*, by which a variable is a function of a set of other variables, imply additional conditional independencies that cannot be captured by faithful models. We developed extensions to the definition of causal models and to the learning algorithm.

Our approach to handle these types of variables and relations is elaborated in the next subsections.

6.1. Information-theoretic dependency

The learning algorithms are based on the information about the conditional independencies. TETRAD uses Pearson's correlation coefficient for calculating the dependency of continuous variables. It gives a measure of how close a relation approximates linearity. Conditional independencies are measured by partial correlations, which can be calculated directly from the correlation coefficients, but only if linearity holds. Correlations can measure non-linear relations, as long as they are quasi-monotonically increasing or decreasing. Partial correlations, however, fail if the relations diverge too much from linearity. This was confirmed by our experiments. Therefore we had to use a form-free definition of dependency. Moreover, correlations cannot distinguish between the deviances of linearity and the uncertainty of the relation. The *mutual information* measures, independently from the form of the relation, the degree of association between variables. It is a measure defined by information theory, see the excellent introductory book of Cover and Thomas [7]. The core concept is entropy.

Entropy is the amount of uncertainty of a stochastic variable. For a discrete random variable X with alphabet A and probability mass function $p(x)$, its *entropy* is defined as

$$H(X) = -\sum_{x \in A} p(x) \log_2\ p(x) \tag{6}$$

It represents the number of bits for the minimal binary code that can describe x. It is maximal for the uni-

form distribution. The conditional entropy $H(Y \mid X)$ is defined as

$$H(X \mid Y) = \sum_{y \in B} H(X \mid Y = y) \tag{7}$$

where B is the alphabet of random variable Y. Probabilistic dependency can be defined as the *mutual information* $I(X;Y)$ of X and Y, which is the reduction in uncertainty of X when knowing Y:

$$I(X;Y) = H(X) - H(X \mid Y) \tag{8}$$

It is zero when both variables are independent. The mutual information can be rewritten as

$$I(X;Y) = \sum_{x \in A} \sum_{y \in B} p(x,y) \log_2 \frac{p(x,y)}{p(x)p(y)} \tag{9}$$

Conditional independence, $I(X;Y \mid Z)$, is defined in the same way. The G^2 independence test [3], used by the TETRAD tool, is of the same form:

$$G^2 = 2 \sum \ observed. \ln \frac{observed}{expected} \tag{10}$$

For measuring the entropy of continuous variables, their distributions are discretized. For a variable X with density $f(x)$ and division of the range of X in bins of length Δ, the entropy of the quantized distribution is [7]

$$H(X^{\Delta}) = -\int_{-\infty}^{+\infty} f(x_i)\Delta \log_2(f(x_i)\Delta) \tag{11}$$

The discretized definition of mutual information is of the same form as Eq. (9). This makes it possible to handle continuous and discrete variables identically and calculate dependencies for a mixture of continuous and discrete variables. The difference with Pearson's correlation coefficient is that mutual information considers each value of X independently, it sums for every x the decrease in uncertainty of Y. The relation between X and Y can be arbitrary, whereas correlation compares the y values in the whole range of X.

The main disadvantage of our test is that larger sample sizes are needed. The definition suggests that for every value of X multiple data points are needed. This approach would be the same as discretizing the continuous variable and would also induce a quantization error. Kernel density estimation, explained in Section 7.1, overcomes these problems. The reason is that $P(X = x)$ is influenced by the data points in the neighborhood of x. The number of data points needed can be limited, the estimation only assumes a smoothly changing distribution.

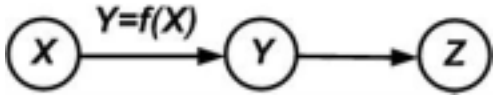

Fig. 6. Causal model with $Y = f(X)$ a function.

For measuring independence, a threshold on the mutual information is used, of 0.4 for continuous and 0.2 for discrete variables.[1] Our independence test replaces the original Tetrad independent tests.

6.2. Deterministic relations

The existence and construction of causal models are based on the property that adjacent nodes share exclusive information, meaning that there is no other node that renders them independent. This assumption is violated if some variables contain the same information about another variable. We call the variables X and Y *information equivalent* with respect to a target variable Z if

$$X \not\perp\!\!\!\perp Z \ \& \ X \perp\!\!\!\perp Z \mid Y \ \& \ Y \perp\!\!\!\perp Z \mid X \tag{12}$$

Either variable becomes conditionally independent from Z by conditioning on the other. Take the model of Fig. 6. X contains all information about Y, thus $Y \perp\!\!\!\perp Z \mid X$ follows, besides $X \perp\!\!\!\perp Z \mid Y$ that follows from the Markov property. The deterministic or functional relation between X and Y imply that both are information equivalent for Z.

Information equivalences cannot be represented by a faithful model. The first condition of Eq. (12) implies that X and Z should be related. The second condition states that X is only indirectly related to Z via Y. Yet the third condition implies the opposite, that Y should be related to Z via X. Consequently, the adjacency search (explained in Section 5) will fail in constructing such a model, since it would remove both edges $X - Z$ and $Y - Z$.

The solution we propose is to augment the models with the information of deterministic relations and information equivalences [9]. This knowledge enables us to draw the right conclusions. The d-separation criterion was already extended by Geiger, Spirtes et al. to retrieve independencies that follow from deterministic relations, to what they called *D-separation* [10,30].

[1] By the estimation of the distributions (Section 7.1), even independent variables generate a small positive value for the mutual information. Calibration by comparing dependent and independent variables lead to the threshold values.

For learning faithful models containing information equivalent variables we propose to connect the target variable with the information equivalent variable which has the simplest relation with the target. From the perspective of information, X and Y are equivalent with respect to Z in the model of Fig. 6. Connecting both variables with Z disrupts minimality, therefore we propose to connect the variable having the simplest relation with Z. Under the assumption that the complexity increases along a causal path, the complexity of $X - Z$ will be higher than that of $Y - Z$. The adjacency search procedure was modified for finding equivalent variables by comparing the conditional independence tests according to Eq. (12). An extra step was added to the algorithm in which the complexity of equivalent relations is calculated using a regression analysis and the one with the least complexity is chosen. The learning algorithm then outputs the minimal model.

Deterministic variables add redundant information to the model. Many authors therefore demand to eliminate them from the data. We take a different viewpoint. Relevant deterministic variables provide insight in the underlying mechanisms. They also often reduce the complexity of the model, as will be shown in the performance modeling experiments.

7. Statistical techniques

Various statistical techniques are combined for the modeling.

7.1. Kernel density estimation

For applying the information-theoretic definitions, it is necessary to obtain an estimation of the underlying probability distributions. The distribution of discrete variables can be estimated by simply counting the number of occurrences of each state and dividing them by the number of data points n. For continuous variables, *kernel density estimation* makes it possible to estimate the distribution from limited sample sizes. The kernel estimate is constructed by centering a scaled kernel at each observation. The value of the estimate at point x is the average of the n kernel ordinates at that point. The idea is to spread a 'probability mass' of size $1/n$ associated with each data point about its neighborhood. See [8] for a gentle introduction, [17] for interactive applets and Fig. 7 for an example.

The estimated distribution is the result of a convolution of the data points with a well-chosen kernel [36]:

$$p(x) = \frac{1}{nb} \sum_{i=1}^{n} K\left(\frac{x - x_i)}{b}\right) \tag{13}$$

with n the sample size and $K(.)$ the multivariate kernel function, which is symmetric and satisfies $\int K(x)\Delta x = 1$. The factor b is the smoothing bandwidth and determines the width of the kernel. Theoretic analysis and simulation have shown that the choice of the kernel is not crucial. The bandwidth is the determining factor for good estimates. A bandwidth too big would flatten the distribution, a too small would generate a peak for every data point. A good trade-off smoothens the distribution nicely, as shown in Fig. 7. In our experiments, we chose the Gaussian kernel function. As a balanced bandwidth, we used 4 times the range divided by the number of data points for the bandwidth. The rationale behind this choice is that kernels of neighboring points should overlap. The bandwidth should therefore be related to the distance to the closest neighbor. The average distance between neighboring points is the range divided by the number of data points. Hence it is a good unit to scale the bandwidth against.

For the estimation of multivariate distributions, multidimensional Gaussians are used. For each dimension a specific bandwidth is calculated according to the method used for the 1-dimensional case.

7.2. Regression and complexity analysis

Regression analysis is used to find the most appropriate function that fits the curve $X_i = f_i(parents(X_i))$. This is based on the fact that any distribution can be described by *functional models*, in which each variable X_i is determined by a function of its parents and the unobserved disturbances U_i: $X_i = f_i(parents(X_i), U_i)$ [25]. The feasibility of the complexity quantification becomes plausible by the modularity assumption of causality (Section 3), by which the overall model is decomposed into local submodel, representing the basic physical mechanisms that generate the states of the variables. The submodels are then of the form $X_i = f_i(parents(X_i)) + U_i$ which makes curve fitting possible.

Curve fitting is a trade-off between hypothesis complexity and goodness-of-fit on the data set. We advocate the Minimum Description Length (MDL) approach, according to which we should pick the hypothesis H_{mdl} from model class M with

$$H_{mdl} = arg\ min_{H \in M}\{L(H) + L(D \mid H)\} \tag{14}$$

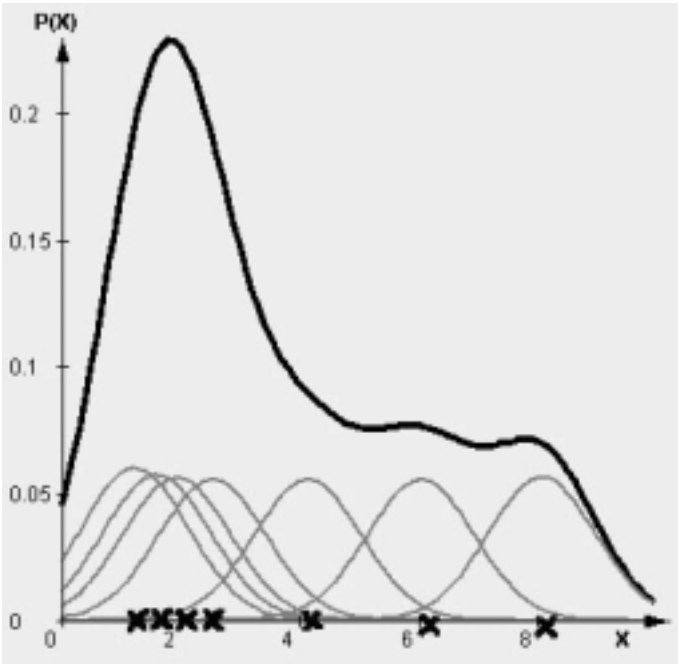

Fig. 7. Data points (the 7 crosses) and kernel estimate (black line) constructed by Gaussians at each point (gray lines).

H_{mdl} is the hypothesis which minimizes the sum of the description length of H and of the data D encoded with the help of H [11].

The model class M is populated with functions appropriate for the system under study. For a performance modeling, we added the polynomials, the inverse, power, square root and step function. The degree of the polynomials is increased until the point where the description length starts increasing. The description of the hypothesis then contains the values of the functions parameters, each needing d bits, and the function type, for which we count 1 byte for each operation (addition, subtraction, multiplication, division, power, square root and logarithm) in the function.[2] A floating-point value is encoded with d bits, whereas an integer value i requires $\log(i)$ bits.

It is shown that the optimal precision d for each parameter is given by $d = 1/2 \log_2 n + c$, with n the sample size and c some constant [26]. Thus

$$L(H) = \#parameters.\frac{\log_2(n)}{2} + 8.\#operations + K \quad (15)$$

with K a constant term that does not depend on H. Therefore it does not play any role in finding the minimal description. The second part of the description, $L(D \mid H)$, reflects the goodness-of-fit of the curve $Y = f(\mathbf{X})$. By choosing the normal distribution as probability distribution of the errors (the deviances of the data with respect to the curve), $L(D \mid H)$ equals the sum of squared errors:

$$L(D \mid H) = \sum_{i=1}^{n} (y_i - f(x_i))^2 \quad (16)$$

The regression analysis has to minimize the sum of Eqs (15) and (16). The Java library, written by Dr Michael Thomas Flanagan (http://www.ee.ucl.ac.uk/∼mflanaga), was used for this purpose. The routines return the closest fit of the given function according to the minimization of the sum of squared errors.

If some of the parents of a node in a causal model are discrete variables, several distinct curves are considered: one for each value combination of the discrete variables. The total complexity is the summation over all individual functions, except that equal parameter values or function types are only counted once. For discrete variables, the conditional distribu-

[2]This choice of description method attributes shorter description lengths for simpler function, but nevertheless is somewhat arbitrary. The correctness of the MDL approach is based on the *Invariance Theorem*. The shortest programs that outputs a given string written in different universal computer languages are of equal length *up to a certain constant* [20]. A complete objective measure is thus not possible.

tions $P(X_i \mid parents(X_i))$ are described by discrete distributions. The number of probabilities in the probability table determine the complexity.

We will define that if one of two information equivalent sets has fewer elements, the relation with the target variable is 'simpler'.

7.3. Modeling process

Figure 8 shows the different steps during the modeling process. Our EPDA tool offers all statistical facilities in a semi-automatic way: the user guides the process by choosing the sequence of tasks to be executed. Experiments are profiled and the measured variables are stored in a database. *Derived variables*, defined by the user, are calculated. The model can, for example, often be simplified by dividing variables with the most influential variable. Performance metrics divided by the work size give work-independent values, characterizing the performance load per work unit. Other interesting values to analyze are the *partial derivatives* of metrics f as functions of the program parameters x, y, z. The partial derivative is the derivative of f with respect to one of those parameters, say x, with the others held constant, written as $\left(\frac{\partial f}{\partial x}\right)_{y,z}$. It quantifies the change of f by a changing x. Modeling is an iterative process. At any time, the user can specify derived variables he wants to be added to the model.

The modeling starts with a qualitative analysis, the main focus of our work. The causal structure learning algorithm applied on the data returns the graph of the causal model. This model offers a reduction in complexity, since the model is decomposed into simpler submodels that can be studied independently. Curve fitting applied on the submodels allows the construction of analytical models of the form $f(X_i \mid parents(X_i))$ in case the parents of (X_i) are continuous variables. The quantification of submodels with a discrete variable produce multiple curves, one for each discrete value of the variable. If those curves have similar shapes, the parameters of the shapes can be analyzed by adding them to the data and the model. This allows characterization of the influence of the discrete variable.

8. Experiments

This section reports on the modeling results of the LU decomposition algorithm and the Kakadu image compression algorithm.

8.1. Experimental setup

During each experiment a record is written to a database containing the measured values for each variable. The number of processor cycles, number of instructions, level 1 and level 2 cache misses are read from the processor hardware counters using the PAPI library [4]. An off-line tool enables to view the experimental data, select variables, calculate derived variables and write them to TAB-separated file [18]. The data can then be loaded as mixed data into TETRAD. The PC algorithm with default options is then applied to it. As background knowledge the user indicates which are the input (parameters) and output variables (overall performance metrics and partial derivatives).

8.2. Performance modeling

The parameters of the the LU decomposition algorithm are:

- n from 5 to 300
- *datatype*: short (2 bytes), int (4b), float (4b), double(8b), long (8b) and longdouble (12b)
- *optimization*: the inner loop of the calculation can be optimized. The result of a division that is performed twice is stored in a temporary variable so that it does not have to be recalculated the second time.

Figure 9 shows the performance model of second-order approximation. We see that only the level 2 cache misses L_2M_{op} have a non-negligible influence on the performance. But even with these additional variables, *datatype* is still directly related to C_{op}. In other words, L_2M_{op} and $instr_{op}$ are insufficient to explain all processor cycles.

We also introduced the derived variable *elementsize*, the size in bytes of the *datatype*. The model then shows that the cache misses are determined only by the size of the data, not by its type. This knowledge enables the prediction of the cache misses for new types. Moreover, *elementsize* is a continuous variable. The relation with the cache misses is a function, which makes it possible to predict the cache misses for yet unknown sizes. The strength of our approach is illustrated by the fact that the relation of the discrete variable *datatype* and the *cache misses* would be a table without predictive capacities. This illustrates the benefits of redundant, deterministic variables.

The model shows that *optimization* only influences the number of instructions, not the cache misses. But

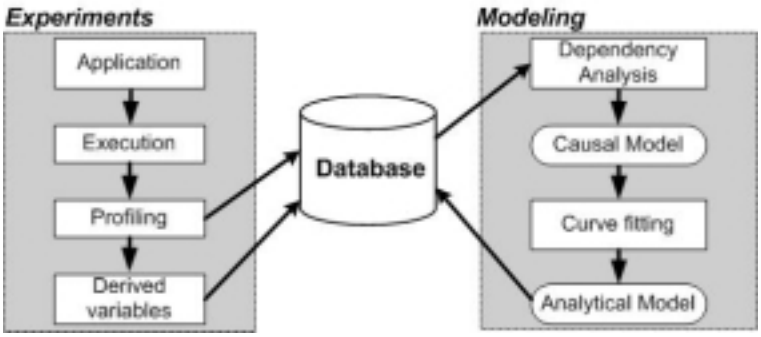

Fig. 8. Scheme of the modeling process.

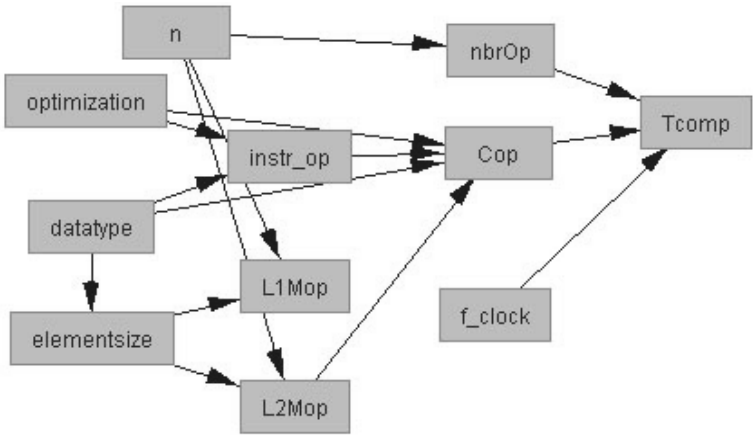

Fig. 9. Detailed performance model of LU decomposition.

the number of instructions cannot completely characterize the effect on the performance. *Optimization* is also related to the number of cycles C_{op}. This confirms that the number of instructions is not a good measure for explaining the overall performance after all. It is therefore the number of cycles we have to look at. To investigate deeper the effect of the optimization on the performance, the partial derivative $\partial C_{op}/\partial\ optimization$ is calculated. It gives the decrease in number of cycles by the optimization. By adding this variable, we can learn on which variables the effect of the optimization depends. Figure 10(a) shows that the decrease in cycles depends on the datatype. Quantitative results show it is about 50 cycles for integers, only 18 cycles for floats or doubles, but 110 for longs.

Dependency analysis also allows the validation of models presumed by the user. The influence from *datatype* on C_{op} goes via L_2M_{op} and $instr_{op}$, as shown in Fig. 9. Omitting the link $datatype \rightarrow C_{op}$, however, would be incorrect. This can be verified by applying the Markov condition (see Section 3) on node C_{op}: without the link, *datatype* would be independent of C_{op} given L_2M_{op} and $instr_{op}$. Yet, the independent test applied on the experimental data gives a mutual information $I(datatype; C_{op} \mid L_2M_{op}, instr_{op}) = 0.45$. This is above the threshold of 0.2, so Markov is violated and both nodes must be connected in the model.

8.3. Datatype characterization

We want to characterize the influence of the discrete variable *datatype* on the performance. It turns out that the cache misses can be predicted with the size of the *datatype*, called *elementsize*, but that the penalty cycles caused by a miss cannot be predicted by an application-independent feature.

Figure 11 presents the experimental data of the level 2 cache misses in function of matrix size. It shows clearly how the misses jump to another level when the cache memory is filled completely. Parametrization of L_2M_{op} in function of its parents is done by a regression analysis performed on the curves for each *datatype*

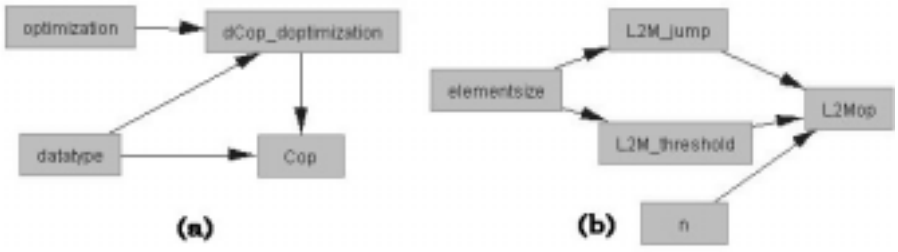

Fig. 10. Parts of the performance model of LU decomposition. Effect of the optimization (a) and modeling of the cache misses (b).

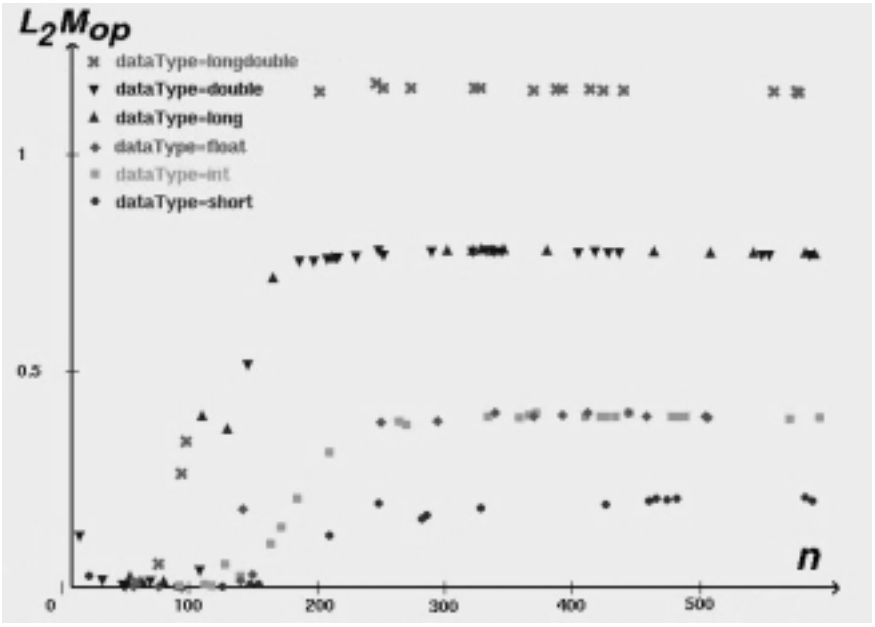

Fig. 11. Experimental results of LU decomposition: Level 2 cache misses versus matrix size n in function of the data type.

separately, as *datatype* is a discrete variable that does not allow to define continuous functions over it. The curve fitting thus seeks for $L_2M_{op} = f_i(n)$ with i corresponding to the datatype. It results in a step function:

$$\begin{aligned} L_2M_{op} &= \pm 0 \quad if \quad n < \ threshold_{L_2M} \\ &= jump_{L_2M} \quad if \quad n > \ threshold_{L_2M} \end{aligned} \tag{17}$$

The cache misses before the jump can be neglected, since it is smaller than the resolution of the estimation. Then, the parameters $jump_{L_2M}$ and $threshold_{L_2M}$ of the functions are added to the model. They both depend on *elementsize*, as shown in Fig. 10(b). A regression analysis reveals that both are linearly related to *elementsize*. This can be expected. The cache data is badly reused since only the first element of the cache line (of 64 bytes) fetched from the lower memory level is effectively used. The cache line is overwritten during the following instructions, when data further in the matrix is needed.

Next, C_{op} is analyzed quantitatively, in order to reveal how many cycles are spent to computation and how many cycles the processor is just waiting for memory accesses. The curve C_{op} is fitted for each datatype separately and gives

$$C_{op} = CPI.\#instr_{op} + C_{L_2M}.L_2M_{op} \tag{18}$$

It results in an equation with 2 unknowns, CPI, the cycles per instructions, and C_{L_2M}, the penalty cycles due to a cache miss. Dependency analysis reveals that CPI still depends on the *datatype*, but unexpectedly also does C_{L_2M}. The penalty cycles increase from 166 per miss for the *integer* datatype, 295 for *double*, to 418 for *longdouble*. These values, however, depend on the application. Which as was shown by a dependency

analysis performed on data retrieved from experiments with other applications.[3]

8.4. Application factors

A goal of performance modeling is to predict the runtime of an application on an arbitrary system. This requires the detection and definition of independent application and system performance characteristics – called application signature and system profile – and a simple functional relation to calculate the performance of the application running on the system. This is called the *convolution method* [29]. The independence of the characteristics signifies that application characteristics are independent of the system, and thus valid for all systems. Vice versa, system characteristics are application independent, adequate for predicting the performance of other applications. Figure 12 shows the performance model developed in the previous sections. We added C_{instr} and C_{mem} to differentiate between the cycles spent on executing instructions and those spent idling due to memory accesses. The model supposes that $threshold_{L_2M}$ can be calculated by the $memory\ usage$ of the application and the $memory\ size$ of the machine.

The figure shows which variables *could* constitute the application signature and which the system profile, respectively the top and bottom variables. This characterization is, however, too simple. C_{L_2M} (the penalty cycles due to a level 2 cache miss) is not a system constant, since depending on the datatype and application. Also CPI, the cycles per instructions, is application dependent. Moreover, $threshold_{L_2M}$ cannot be determined by the memory requirements and memory size only. The modeling thus failed in finding independent system characteristics.

8.5. Parameter sensitivity of image compression

Next, we applied our approach for modeling the parameter sensitivity of the Kakadu algorithm for the compression of still images. Kakadu is a $C{+}{+}$ implementation of the JPEG-2000 standard [14], which has been developed to address a number of weaknesses in the existing JPEG standard and to provide a number of new features. It supports lossless and lossy compression, progressive recovery of images by resolution, region of interest coding, random access to particular regions of an image, etc. It therefore is extremely suitable for internet applications. The coding algorithm mainly consists of two stages. The first stage splits the image into frequency bands through the iterative application of a wavelet transform. Each transform yields four subbands: horizontally and vertically lowpass (LL), horizontally lowpass and vertically highpass (LH), horizontally highpass and vertically lowpass (HL) and horizontally and vertically highpass (HH). The wavelet decomposition is associated with R resolution levels, where at each level the LL band is further decomposed, as shown in Fig. 13. Due to the statistical properties of these subband signals, the transformed data can usually be coded more efficiently than the original untransformed data.

During the second stage, the subbands are partitioned into code blocks, typically 64 $\times$ 64, which are independently coded using a bit-plane coder (all first bits of the pixels of the block are coded together, followed by the second bits, etc.). It generates a sequence of symbols that is compressed by a entropy coder: the MQ coder. Rate scalability is achieved through quality layers. The coding passes containing the most important data (based on the lowpass subbands) are included in the lower layers, while the coding passes associated with finer details (based on the highpass subbands) are included in higher layers. During decoding, the reconstructed image quality improves incrementally with each successive layer processed. Consult [1] for an easy accessible introduction, and [32] for an elaborate discussion.

The Kakadu implementation, written in C, has its key focus on memory efficiency and execution speed. We investigated the parameter sensitivity of the compression execution time – the cost in performance of changing the algorithm's configuration. Due to Kakadu's high number of parameters, a task well suited for being automated. Experiments were performed with 12 different images, scaled to different widths and heights (ranging both from 500 to 10000 pixels), and various parameter settings that overspan the entire configuration space. After some try-outs we selected the most interesting parameters for running the final experiments: the *precision* of the data representation (16-bit or 32-bit), the *kernel* used for the wavelet transform (the $5/3$ or $9/7$ kernel), the *blockSize* used in the second stage (ranging from 4 $\times$ 4 up to 64 $\times$ 64), the number of quality *layers* (varied from 2 to 12) and the bitrate of the highest layer (set between 0.5 to 10 bits/pixel). This last parameter is only set when lossy compression is employed. At each run, the following variables are

[3] A matrix multiplication and a merge sort, which can be regarded as quite similar to the LU decomposition. They exhibit the same cache miss pattern as depicted in Fig. 11.

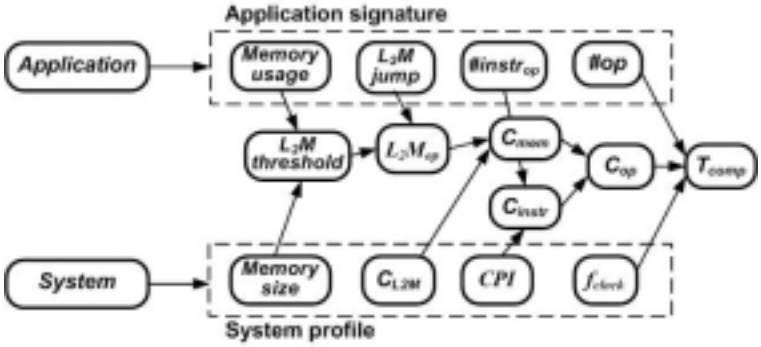

Fig. 12. Fictive performance model of LU decomposition with independent application and system characteristics.

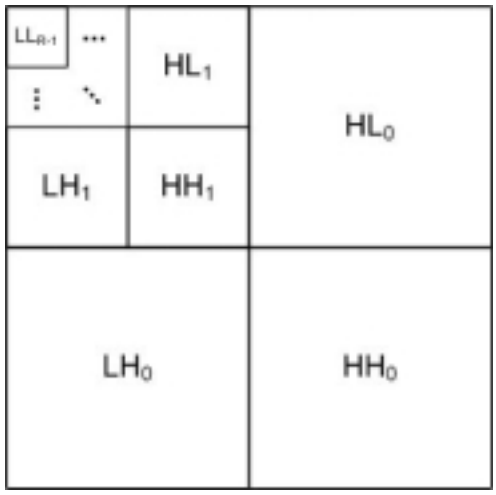

Fig. 13. Wavelet decomposition of an image into subbands by the JPEG-2000 compression.

measured: the image *size*, the runtime T, the number of processor instructions *instr*, the resulted number of bits of the compressed image *bits*, the level 2 caches misses $L2M$ and the number of processor cycles per instructions (CPI). All these quantities, except *size* and CPI, are divided by the image size to get per pixel values. They are denoted by the suffix '*ppel*'. The results approve that most of these values become size independent and hence simplify the model. Furthermore, some interesting partial derivatives are calculated, explained in Section 7.3. They are named dX_dY signifying the derivative of the function of X with respect to parameter Y while the other parameters are kept constant.

Figure 14 shows the learned model for lossless and lossy compression. Lossless compression results in a bitrate, variable *bitsppel*, measured in bits per pixel, which reflects the information necessary to describe the image. It is determined by the compressibility of the image. Lossy compression allows the bitrate to be chosen by the user. The data for lossless and lossy compression were analyzed independently, after which both resulting models were merged into one. Arrows with dashed lines indicate relations that only apply for lossless compression.

The results for lossless compression comprise the following interesting observations:

- All performance characteristics increase linearly with the image size, the values per pixel are therefore size independent.
- The number of desired quality *layers* has no significant impact on the bitrate or performance.
- The resulting bitrate, *bitsppel*, influences the number of instructions, *instrppel*, and the the cycles per instruction, CPI. The bitrate is, however, an outcome of the compression and can thus not be a cause. The correct interpretation is that it is the compressibility of the image, determining *bitsppel*, that also affects the performance.
- The cache misses do not affect the runtime. This confirms the memory efficiency of the implementation. The model reveals that the number of instructions are a good indicator for the number of cache misses.
- The increase of the runtime $Tppel$ by choosing a higher *precision* is not considered as significant by the independency test. The derivative $dTppel\_dprecision$ is, however, positive, and depends on the *kernel* and the image *width*.

The model simplifies for lossy compression, when *bitsppel* is set as a parameter. In that case, the number of instructions, *instrppel*, is only affected by *bitsppel* and *blockSize*. Furthermore, CPI is not influenced by the compressibility of the image.

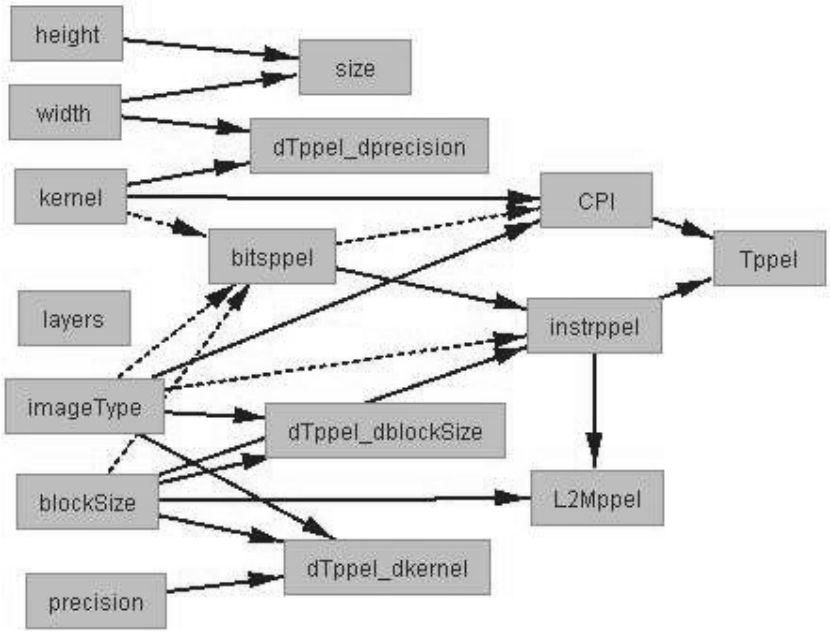

Fig. 14. Performance model of JPEG-2000 Image Compression with Kakadu. Arrows with dashed lines only apply for lossless compression, for which *bitsppel* is not set as a parameter.

8.5.1. Image characterization

Ideally, every image can be characterized by one compression performance factor, which determines the number of instructions as well as the runtime, and is valid for all possible parameter configurations. This is certainly not the case as can be seen on Fig. 14. For lossless compression, *imageType* influences the compressibility *bitsppel*, *instrppel* and CPI independently. This means that an image has at least three distinct features characterizing the compression performance. None of the three variables contains the information of *imageType* so that it is able to predict the values of the 2 others without needing the value of *imageType*. Next, the influence of *imageType* on *dTppel_dkernel* indicates that the compression behaviour of an image with one kernel says little about its behaviour with another kernel. Indeed, inspection of the data reveals that some images are compressed faster with kernel 5/3 and others with 9/7. Images also behave differently under different blocksizes, expressed by the relation between *imageType* and *dTppel_dblockSize*. Only the computational cost of using higher precision data representations, *dTppel_dprecision*, is image independent.

9. Conclusions

A causal model is a qualitative model representing the direct causal relations among the variables. It furthermore allows the retrieval of the dependency qualification between any 2 variables. We showed that causal structure learning algorithms are able to construct causal performance models from experimental data. Statistical techniques are used to extend existing algorithms in order to incorporate the variety of variable types and relations encountered in performance data. The general independence test based on mutual information was used to handle combinations of continuous and discrete variables and non-linear relations. The underlying probability distribution of experimental data is estimated by kernel density estimation. An extension, based on the complexity of relations, was necessary for modeling data containing deterministic relations. The learning module was integrated in the semi-automatic modeling process of performance, which also included parametrization using regression analysis.

The benefit of causal models lies in their intrinsic support for the Markov condition and the decomposability of the models into independent submodels. Experiments show that accurate qualitative models are inferred. The models provide insight in which and how variables affect the overall performance, as demonstrated by the performance sensitivity analysis of the Kakadu image compression algorithm. Modeling consists, besides decomposition, of defining generic system and application properties that characterize the aspects of their behavior that influence the performance.

Causal analysis can reveal such properties. The results showed that for LU decomposition a simple performance model is inadequate in capturing independent application and system characteristics.

References

[1] M.D. Adams, *Coding of still pictures*, the jpeg2000 still image compression standard, iso/iec jtc1/sc29/wg1 n2412. http://www.ece.uvic.ca/~mdadams, 2001.

[2] R.M.e.a. Badia, Dimemas: Predicting mpi applications behavior in grid environments, in: *Workshop on Grid Applications and Programming Tools* (*GGF8*), 2003.

[3] Y.M.M. Bishop, S.E. Fienberg and Holland, *Discrete multivariate analysis. Theory and practice*, Cambridge: MIT Press, 1975.

[4] S. Browne, J. Dongarra, N. Garner, G. Ho and P. Mucci, A portable programming interface for performance evaluation on modern processors, *International Journal of High Performance Computing Applications* **14**(3) (2000), 189–200.

[5] L.C. Carrington, M. Laurenzano, A. Snavely, R.L. Campbell and L.P. Davis, *How well can simple metrics represent the performance of hpc applications*? In Proc. of the 2005 ACM/IEEE conference on Supercomputing, Washington, DC, USA, 2005. IEEE Computer Society, 48.

[6] I. Cohen, J.S. Chase, M. Goldszmidt, T. Kelly and J. Symons, *Correlating instrumentation data to system states: A building block for automated diagnosis and control*, In OSDI, 2004, 231–244.

[7] T.M. Cover and J.A. Thomas, *Elements of Information Theory*, John Wiley & Sons, Inc., 1991.

[8] T. Duong, *An introduction to kernel density estimation*, http://www.maths.uwa.edu.au/~duongt/seminars/intro2kde/, 2001.

[9] T. Fahringer and C. Seragiotto, Automatic search for performance problems in parallel and distributed programs by using multi-experiment analysis, in: *HiPC, volume 2552 of Lecture Notes in Computer Science*, S. Sahni, V.K. Prasanna and U. Shukla, eds, Springer, 2002, pp. 151–162.

[10] D. Geiger, *Graphoids: A Qualitative Framework for Probabilistic Inference*, PhD thesis, University of California, Los Angeles, 1990.

[11] P. Grünwald, *The Minimum Description Length Principle and Reasoning under Uncertainty*, ILLC Dissertation series 1998-03. PhD thesis, University of Amsterdam, 1998.

[12] A.J.G. Hey, A.N. Dunlop and E. Hernández, Realistic parallel performance estimation, *Parallel Computing* **23**(1–2) (1997), 5–21.

[13] R.A. Horn and C.R. Johnson, *Matrix Analysis*, Cambridge University Press, 1985.

[14] ISO/IEC. Iso/iec fcd15444-1, information technology - jpeg 2000 image coding system. http://www.jpeg.org, 2002.

[15] K.L. Karavanic, J. Myllymaki, M. Livny and B.P. Miller, Integrated visualization of parallel program performance data, *Parallel Computing* **23**(1–2) (1997), 181–198.

[16] K.B. Korb and A.E. Nicholson, *Bayesian Artificial Intelligence*, CRC Press, 2003.

[17] J. Lemeire, *Documentation and applets on causal performance models*, http://parallel.vub.ac.be/causalperformance models.

[18] J. Lemeire, *Documentation of epda tool* (*experimental performance data analysis*), http://parallel.vub.ac.be/epda.

[19] J. Lemeire, S. Maes, S. Meganck and E. Dirkx, *The representation and learning of equivalent information in causal models*, Technical Report IRIS-TR-0099, Vrije Universiteit Brussel, 2006.

[20] M. Li and P. Vitanyi, *An Introduction to Kolmogorov Complexity and Its Applications*, Springer Verlag, 1997.

[21] B. Mohr and F. Wolf, Kojak – a tool set for automatic performance analysis of parallel programs, in: *Euro-Par, volume 2790 of Lecture Notes in Computer Science*, H. Kosch, L. Böszörményi and H. Hellwagner, eds, Springer, 2003, pp. 1301–1304.

[22] K. Murphy, *Bayesian network software overview*, http://www.cs.ubc.ca/~murphyk/Software/BNT/bnsoft.html.

[23] I. NetPredict, *Common mistakes in performance analysis*, white paper. NetPredict Inc, 2003.

[24] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, San Mateo, CA, Morgan Kaufman Publishers, 1988.

[25] J. Pearl, *Causality. Models, Reasoning, and Inference*, Cambridge University Press, 2000.

[26] J. Rissanen, *Stochastic Complexity in Statistical Enquiry*, World Scientific, Singapore., 1989.

[27] R. Scheines, P. Spirtes, C. Glymour, C. Meek and T. Richardson, *TETRAD 3: Tools for Causal Modeling – User's mannul*, http://www.phil.cmu.edu/projects/tetrad/tet3/master.htm, 1996.

[28] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R.M. Badia and A. Purkayastha, *A framework for performance modeling and prediction*, In SC, 2002, 1–11.

[29] A. Snavely, N. Wolter and L. Carrington, Modeling application performance by convolving machine signatures with application profiles, in: *WWC '01: Proceedings of the Workload Characterization*, 2001 IEEE International Workshop on Washington, DC, USA, 2001. IEEE Computer Society, pp. 149–156.

[30] P. Spirtes, C. Glymour and R. Scheines, *Causation, Prediction, and Search*, Springer Verlag, 2nd edition, 1993.

[31] P. Spirtes, C. Glymour, R. Scheines and J. Ramsey, *The tetrad project*, http://www.phil.cmu.edu/projects/tetrad/.

[32] D.S. Taubman and M.W. Marcellin, *JPEG 2000: Image Compression Fundamentals, Standards and Practices*, Kluwer Academic, Boston, USA, 2002.

[33] J. Tian and J. Pearl, *A general identification condition for causal effects*, In AAAI/IAAI, pages 567¤C573, 200

[34] H.L. Truong and T. Fahringer, Scalea: A performance analysis tool for distributed and parallel programs, in: *Euro-Par, volume 2400 of Lecture Notes in Computer Science*, B. Monien and R. Feldmann, eds, Springer, 2002, pp. 75–85.

[35] T. Verma and J. Pearl, *Equivalence and synthesis of causal models*, in In Proc. of the 6th workshop on uncertainty in Artifical Intelligence Cambridge, 1991.

[36] M. Wand and M. Jones, *Kernel Smoothing*, Chapman & Hall, London, UK., 1995.

[37] J. Yan, S. Sarukkai and P. Mehra, Performance measurement, visualization and modeling of parallel and distributed programs using the aims toolkit, *Software – Practice and Experience* **25**(4) (19995), 429–461.