

Hierarchical Fuzzy Configuration of Implementation Strategies

Angela Sodan
GMD FIRST
Rudower Chaussee 5
12489 Berlin, Germany
E-mail: acsodan@gmd.de

Vicenç Torra
Universitat Rovira i Virgili
Carretera de Salou s/n
43006 Tarragona, Spain
E-mail: vtorra@etse.urv.es

Keywords: Multistage fuzzy inference, configuration, parallelization, implementation strategies

ABSTRACT

With complexity currently growing and broader ranges of applications having to be dealt with, increasing numbers of configuration problems are arising in compilers. Already many software systems offer multiple specialized implementation strategies and substrategies, differing in terms of applicability and/or cost, depending on the application context. Configurations then have to be created from the different strategies available in accordance with the application characteristics, the global optimization objective, and potential constraints on the strategies' combinability—resulting in many cases in a combinatorial, i.e. discrete, optimization problem. Proper solutions for automating the configuration while limiting the complexity of the solution search are still being sought. We address here the field of parallel/distributed processing and the configuration of dynamic implementation strategies such as for communication or dynamic load balancing. We present a rule-based approach, integrating fuzziness for the classification of application characteristics and for gradual selection preference in rules. The approach extends standard fuzzy inference by a multistage organization and—with proper organization of rules, characteristics and strategies—performs hierarchical fuzzy inference. The approach is demonstrated on concrete configuration examples in parallel compilers.

INTRODUCTION

There is currently a general trend toward synthesis, integrating multiple implementation strategies in a single system. This is based on the observation that there are often a variety of reasonable implementation strategies basically performing the same semantic task. In some cases,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC '99, San Antonio, Texas
©1998 ACM 1-58113-086-4/99/0001 \$5.00

the various strategies have different advantages and disadvantages, the best choice depending on the characteristics of the respective application. In other cases, specializations with better performance may be used for certain parameter settings. Here, we consider dynamic implementation strategies at the runtime-system or operating-system level in parallel and distributed systems, which are currently growing in importance, with even PCs starting to support parallelism via 2 or 4 processors and with improved networks supporting distributed computing in local clusters or via the Internet. The level considered here basically comprises the main functional classes of communication, synchronization, execution model (such as SPMD or multithreading), dynamic memory management, dynamic load balancing, and scheduling. An example of different strategies performing differently depending on the context are in the case of communication: message passing and direct remote-memory access on distributed-memory machines. Message passing generally involves copying data to buffers, thus entailing significant overhead, but it makes synchronization on data easy, whereas direct memory access is faster but requires more careful handling of the data accesses. As an example of specialization, the communication mechanisms need to be more general for distributed computing in dynamically assembled and heterogeneous configurations with unreliable interconnections, whereas they can be simplified in homogeneous, static, and reliably interconnected parallel machines. Discussions of further strategies and their trade-offs can be found e.g. in [3], [1], or [17]. Examples of concrete systems offering a variety of strategies are: the operating system Nexus [6], supporting a variety of message-passing communication mechanisms; the Ace system [16], offering both message passing and virtual shared memory; and the Globus metacomputing system [7] providing a rich ensemble of different policies for different strategy classes. This trend toward multiple-strategy systems is encouraged by the fact that systems are designed for use with a broader range of application behaviors; that the applications themselves are growing in complexity, often consisting of heterogeneous components; that metacomputing applications are evolving which run on highly heterogeneous distributed systems; and that efficiency is more important for complex applications or simple computing devices like pagers (with limited processing and space capacity), which call for specialized solutions.

Multiple-strategy systems raise the problem of configuration and its automation. The configuration of discrete heterogeneous components such as dynamic implementation strategies usually results in a combinatorial, i.e. discrete, optimization problem. Then, neither direct equational solutions are possible, nor do straightforward decision paths normally exist. The optimization objective, i.e. the cost function to be optimized, depends on the system or user requirements, and may, for example, be minimum space, minimum turn-around time, or maximum throughput. Here, we refer to turn-around time (or, correspondingly, speedup), i.e. the parallel runtime of a single application. Our basic approach does not, however, depend on any specific cost function.

Combinatorial discrete optimization is NP hard in the general case, because there are 2^m ways of creating combinations from m strategies and thus full search can be applied only to small problems. For more complex problems, only suboptimal solutions can be sought to make configuration practically tractable, but they are almost always sufficient. Most frequently often applied are heuristics, but more systematic and tunable approaches are desirable and currently under development to allow quantification and selection of the quality of the solution obtained. We apply an approach based on fuzzy rules and fuzzy inference, which potentially falls into either of the latter categories depending on the concrete problem setting. The approach extends conventional fuzzy systems as used in fuzzy control by delivering multiple elements of the reference set at potentially every abstraction level, by incorporating constraint handling, and by properly chaining logically dependent decisions (for details of the basic approach, see [21] and [19]). Here, we extend the approach to hierarchical fuzzy configuration based on proper rules, characteristics and strategy organization. We demonstrate the resulting system, realized as a prototype, by membership-function definition for strategies and by the configuration of strategies for two nontrivial applications. The proposed hierarchical fuzzy configuration offers the following main benefits:

- The use of rules based on knowledge about correlations between parameter ranges and performance allows us to select promising strategies and try combinations for them only, thus reducing the search space. The incorporation of fuzziness enables us to deal with strategies' differing degrees of appropriateness and with transitions between classifications, and thus helps us to find globally good configurations.
- The approach allows the incorporation of uncertain and heuristical knowledge, then being heuristical itself, or an approximative approach with tunable, systematic classification and rule definition if cost functions are available.
- The hierarchical organization helps reduce the search space if predecisions can already be made on the basis of partial configurations, and it also supports the definition of rules and membership functions by reducing complexity.

RELATION TO OTHER WORK

Combinatorial optimization in parallel compilers—the field most-related to ours—often occurs in the static assignment of tasks or data to machine nodes and becomes relevant especially if data or tasks are heterogeneous (intra- or inter-computation phases). Though in some cases of limited problem complexity, optimal solutions are calculated using relatively efficient exponential solutions [11], usually only suboptimal solutions can be obtained within a reasonable time. The approach most frequently applied are heuristics, which incorporate domain knowledge, e.g. for (in subconsiderations) determining the order in which direct-solution-path decisions are made, and which have in many cases proved capable yielding near-optimal results in a significantly shorter time. Heterogeneous Optimal Selection Theory [4], for example, proposes a hierarchical but heuristical approach. It is an approximative approach using the same basic model as for the optimal case, e.g. 0-1 integer solution approaches may be stopped when a sufficiently good solution is found [11]. Such approaches do not, however, apply knowledge about the structure of the solution space. Best-first search with the same ratings applied would deliver similar results to our approach (provided that not only the best, but several good solutions are delivered). The best-first-search variant applied in [10], however, assumes monotonically increasing ratings, and thus—like branch&bound—would not work with our ratings. Some approaches use hierarchical cost functions to reduce calculation complexity such as [14], which, however, uses neither rules nor fuzziness to reduce complexity per level. In the field of classification, cross-over point analysis [20] calculates in which case one or the other communication strategy performs better (if scaling problem sizes). This approach is, however, applicable only to individual decisions.

With respect to fuzzy inference, other approaches in the direction of fuzzy multistep reasoning either do not use forward chaining (such as [2]), or multistage defuzzification (such as [9, 13]). One of the problems that multistage systems have to cope with, is propagation of the uncertainty. When fuzzy rules are chained, the *degree* of uncertainty increases. Thus systems with several stages may deliver as output a completely uncertain solution. The propagation of uncertainty with several inference models was studied in [15]. An alternative that allows us to avoid increasing uncertainty is to defuzzify after each stage. We have used a hybrid solution: defuzzification is applied after each stage, but chaining is allowed within a stage.

THE APPROACH TAKEN

The Configuration Problem

Our approach exploits knowledge about the system's application domain to reduce search space, this knowledge concerning existing correlations between specific parameter ranges and specific performance values or concerning dominance of main strategies over their substrategies with

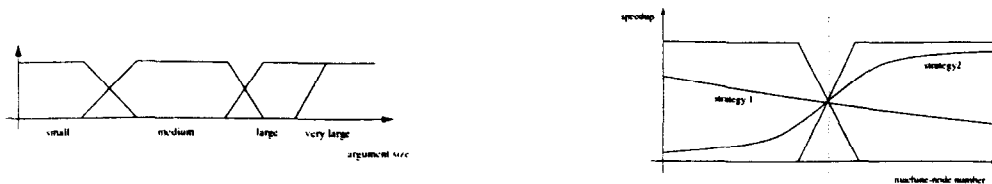


Figure 1: Some examples of fuzzy sets and their potential membership functions (left), and classification of speedup for two different single-strategy configurations (right).

respect to performance. This results in a mixture of pre-selection by static (frozen) know-how and the need for arbitrary dynamic combinatorial trials.

Furthermore, we incorporate fuzziness by using fuzzy classifications for application and system characteristics in the rules' antecedents and fuzzy preference values for the selections in the rules' consequences. Figure 1 gives examples of the fuzzy classification of application characteristics, Figure 1b showing this in relation to a concrete performance curve. Readers unfamiliar with fuzzy systems, are referred to the excellent textbook of Klir and Yuan [12]. Basically, the classifications and rules perform a decomposition into parameter and performance subspaces in order to select those strategies that are relevant for the final configuration.

The gradual transitions offered by the fuzzy classifications capture the fuzzy transitions between the different suitability of different strategies (see Figure 1), and the different fuzzy preference values can express the different performance of the strategies. The strategies are discrete (crisp) entities like a specific load-balancing strategy. Compatibility constraints—mainly expressed by compatibility rules—represent knowledge about legal and illegal combinations (i.e. compatible and incompatible strategies) and are crisp as well. If evaluated during construction of the search tree, they also help to limit the search tree by pruning meaningless subtrees early on. The nodes in the discrete-optimization search tree represent partial configurations.

Characteristics refer to the application and the system, and are measured by certain metrics such as task runtimes, argument sizes, or dynamic space consumption. The characteristics constitute the input to the fuzzy-inference system. The characteristics of the application may be static or dynamic. Static characteristics can be derived by the compiler from either language features, pragmas, or analysis. Dynamic characteristics—such as dynamically determined runtimes per task or degrees of irregularity in a task structure—may be obtained either from weight analysis (where possible) or by monitoring program runs and evaluating the collected data [24]. System characteristics are assumed to be static. Fuzzy classification can, in principle, apply to all of them. If only one specific system context is envisioned, the system characteristics may be implicitly contained in the relations defined by the rules. Otherwise, they appear in antecedents of rules, as the application characteristics always do.

Preference values refer to the performance (runtime or speedup) of the final configuration. Rules usually select individual strategies. Preference values then rate the absolute or relative influence a specific strategy has on overall performance. This means assuming strategies to be independent from each other with respect to performance—but extensions for incorporating dependencies are possible, e.g. selecting strategies together (as a group) that influence each other in terms of performance. During defuzzification, the global context is considered and the individual ratings of strategies are combined to give a global rating of the (full or partial) configuration. How this is done and how preference values are defined depends on the availability of cost functions.

Complete global cost formulas—integrating all variants of strategies and important aspects of behavior—are not yet available, though they probably will be in the near future. Partial metrics relating to some of the available strategies already exist in several subdomains, and integrated studies, examining the suitability of one strategy in combination with another are emerging [1]. At present, however, for complex systems, rules may have to be based on incomplete knowledge and heuristics, this being expressible via the fuzzy characteristics' classifications and the fuzzy preference values. Even relative orderings may be sufficient. Then, of course, the system delivers a heuristic solution only. If complete global cost formulas are available, preference values refer to them. Rule definition still helps reduce the number of strategies to be considered for configuration by performing a pre-selection—which is useful if the overall number is large. Moreover, the fuzzy classifiers may be easier to compute than the full cost functions if these are complex. Fuzzy classifications and preference values then perform a linear approximation of the real function, and this approximation is tunable by the way the rules are defined and how fine-grained they are.

As regards the rating of the configuration, if full metrics are available they can be used for rating the relevant configurations, especially if abstract metrics are defined allowing cost estimation for higher-level partial metrics. In case full metrics are not available, we provide in our approach a weighting-based fuzzy rating (described below). This can be considered a linear combination for approximation of the (unknown) cost function. Depending on whether full cost functions are available or not, verification of the results may be either by full cost metrics or by

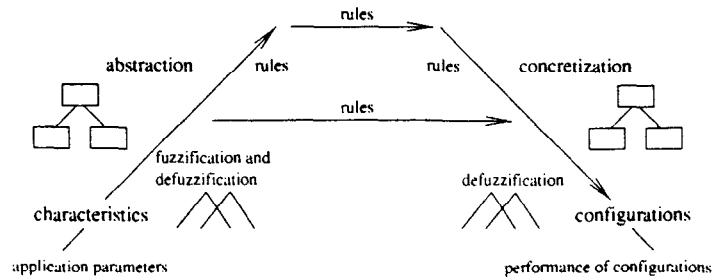


Figure 2: Hierarchy of characteristics and strategies.

human plausibility judgement (potentially accompanied by concrete tests of the configurations).

Solving the Dependency Problem in Fuzzy Decisions

A typical feature of fuzzy inference as used in fuzzy control is that, first, all rules that are partially satisfied are fired in parallel, then the output variable is composed, and finally, a potentially new inference step is applied using the results as input terms. However, in our application context, a more complex chaining of inference is required. In some cases, rules express alternatives that are incompatible, but—since fuzzy ranges may overlap—may both fire with a value in $(0,1)$. Any further conclusions drawn from the alternatives would then be intermixed at the next inference step and flattened into a set. When defuzzification is eventually carried out, this intermixing may lead to incorrect combinations with respect to the dependencies expressed in the individual rules. The causal dependencies that, at some points, need to be reflected for logically disjunct/incompatible alternatives, may not only refer to basic selection of strategies, but to preference values, too. Figures 3 and 5 show concrete examples of dependencies (in Figure 3, the conclusions taken from either static or dynamic behavior should not be intermixed, and in Figure 5, the conclusions taken from either message passing or fine-grained threaded-calls should not be intermixed). To solve dependency problems, the system is organized in multiple stages. At each stage, partial defuzzification and compatibility checking are applied. This may be viewed as knowledge-based defuzzification (because of the application of the compatibility rules) following Yager [23]. As we will show later, the stages can be applied to chaining either hierarchical or same-level sequential decision steps.

Hierarchical Organization

To master complexity in performance estimation and rule construction, a hierarchical organization of characteristics and strategies, accompanied by hierarchical rule and performance-estimation organization, is helpful. It is applicable if higher-level strategies have a dominant influence on overall performance—such as the basic decision

to apply dynamic load balancing if the application exhibits dynamic irregular behavior. Assuming the dominance precondition holds, the system applies the following approach (see Figure 2):

- characteristics are evaluated bottom-up, more complex ones being composed from several lower-level ones
- strategies divide into different classes at different levels of significance (such as main and substrategies) or applicability (such as general and specialized strategies); and configuration basically follows a top-down approach

The same inference mechanisms and multistage defuzzification apply for both parts. The internally resulting decision tree of the system does not distinguish between sequentially and hierarchically dependent decisions, nor between characteristics and strategy selection. We do not impose any restrictions on rules and their relations between levels in characteristics and strategies, although it is likely that in properly structured systems rules most often relate to characteristics and strategies at the same level (see Figure 2).

Figure 3 gives an example of bottom-up characteristics composition. A number of other approaches are available for multilevel evaluation of application characteristics [5, 24], but these are not truly hierarchical (using abstractions), applying crisp-evaluation steps only. Thus, our basic approach is more general.

In the rest of the paper, we mainly consider the top-down strategy selection. Here, the possibility of hierarchical structuring is threefold, allowing a differentiation between 1) main and substrategies, 2) more general and more specific strategies, and 3) different system environments and their potentially different strategies and/or performance ranges. An organization via main and substrategies appears quite frequently, dynamic load balancing, for example, being a main strategy that has different substrategies such as central or decentral control. Specialized solutions are usually more efficient than general solutions, using as they do simpler data types, less conditionals or basically simpler approaches, but they are applicable only under certain circumstances. Specialized solutions, however, may also express (if only referring to performance) more sophisticated (and expensive) strategies needed to

```

IF (communication = asynchronous) OR (task-creation = unpredictable-unbalanced)
  THEN (dynamic-irregular-behavior = true)
IF (communication = synchronous) AND (task-creation = static-or-predictable) AND (data-distribution=static)
  THEN (static-or-predictable-behavior = true)

IF dynamic-irregular-behavior AND (task-granularities = small) AND (communication-rates = frequent)
  THEN (fine-grained-threaded-calls = true)

```

Figure 3: Examples of hierarchical characteristics' classification (first two rules), and a selection rule based on them (last rule).

yield good performance in certain application contexts. Rules may then also set basic defaults, which should be overwritten by more specific rules if an application suggests that a specialized solution will perform better. Different basic system environments (i.e. different machines) may lead to different decision subtrees, either by different sets of substrategies being relevant or rules / membership functions differing.

Assuming a dominant influence of higher-level strategies on performance, early selection of the most relevant branches in the tree is possible. In other words, some promising partial configurations of high-level strategies can be selected at a higher level, the detailing of configurations for substrategies being tried for them only. The same applies to sequentially dependent decisions. This can limit the complexity of the search significantly, while still leaving sufficient flexibility to keep the search broad enough to find a globally good solution. Considering multiple branches, in the general case, is necessary for different reasons. For examples, if only heuristical estimations for ratings of strategies can be given, considering several branches helps not to eliminate potentially good solutions early on. Furthermore, we need to assume that substrategies can at least have a partially relevant influence, in that a substrategy well suited to the application characteristics may improve the overall configuration's performance, while an ill-suited one may lower it. For example, dynamic load balancing with central control may be unsuitable for an application with small task granularities running on a large number of machine nodes (bottleneck), while a strategy with distributed work stealing may perform well [18]. Furthermore, assuming the best substrategies are selected, the application characteristics determine what sort of performance they can achieve. Thus, we assume strategy ratings to include a substrategy rating in the form of a mean value for the potentially selected substrategies, thus abstracting from the substrategies. Complete configurations including substrategies may then deviate to some extent positively or negatively from it. Thus, an overlapping of concrete rating ranges may occur for each strategy with respect to the potential substrategies. Assuming that this overlapping of performance ranges is limited and only applies to strategies with similar ratings, selecting some of the best rated configurations for further consideration offers a solution to this problem. What is more, if dominance of higher-level strategies is ensured and overlapping is limited in the above sense, the optimum solution can be

found. Whether this is true or not depends on the domain in which the approach is applied. Similar arguments apply to the amount of pruning achieved, depending on the amount of overlapping with respect to substrategy consideration—i.e. how narrow or broad the ranges are. Note that because of the overlapping described above, ratings of partial configurations need not be strictly monotonic, but can either increase or decrease at the next level.

THE FUZZY INFERENCE SYSTEM

As already mentioned above, knowledge about the system is expressed in terms of a set of fuzzy rules and a set of compatibility constraints. The former are used in the inference system to select the strategies that are adequate for a certain application; they refer either to the context of the application behavior or to other decisions already taken (in a previous step of the inference system). After each inference step, compatibility constraints are used to maintain logical dependencies across stages of inference (they are used after the defuzzification process, see below). First, we describe these two ways of expressing knowledge.

Selection rules are based on the typical structure of rules in expert systems, and fuzzy control systems: an antecedent, consisting of a logical expression, and a conclusion:

$$\begin{aligned} &\text{if LOG-EXPR}(a_1 = c_1, a_2 = c_2, \dots, a_l = c_l) \\ &\text{then } s_1 = v_1, s_2 = v_2, \dots, s_k = v_k \end{aligned}$$

where the conclusion is a set of strategies s_i ($s_i \in S$) with a fuzzy value (a value in $[0,1]$) attached to each one v_i ; and where the antecedent is a logical expression, LOG-EXPR, built with operators AND, OR, NOT and defined on a set of terms a_i that may correspond to strategies (the ones already selected in previous stages of the inference process) or to characteristics referring to the application or the system, and a fuzzy set for each term a_i . Note that conclusions in rules correspond to an overall conclusion of the form $S = p$, where p is the fuzzy set $p = v_i/s_i$, v_j is in $[0, 1]$, and s_i is in S . The values v_1, \dots, v_k correspond to the fuzzy preference values described above when presenting the solution approach, thus v_j is the fuzzy preference value of strategy s_j where the antecedent of the rule is fulfilled.

Compatibility constraints are defined by means of logical expressions on strategies, i.e. LOG-EXPR(s_1, \dots, s_r) where s_i are strategies. In these rules, all terms are crisp,

a strategy is either present or absent in a final configuration, and the strategies are either compatible or incompatible. And we can say that a particular configuration is legal if all compatibility constraints deliver true.

The selection of the strategies at each stage is performed by the following steps:

1. *Apply selection rules:* Rules whose antecedent evaluations deliver a non-zero value are fired, and the selected strategies are added to the set of strategies from which configurations can be formed
2. *Defuzzification:* Following the previous ordering, each strategy with a value in $[0,1]$, is fixed at either 0 or 1, trying the closest one first
3. *Partial compatibility checking* (as far as decidable): Checking is performed by means of compatibility constraints on the 0/1 settings of those strategies already known at this stage. Most checking is possible—as was true for our examples—if rules refer mainly to the same group of strategies and constraints mainly to strategies selectable at the same stage. When a decidable incompatibility is found, the system leads to a failure and the search subtree is cut.

Once the crisp set is obtained, the system can either apply the next stage or backtrack.

The basic mechanism provided for ranking partial or full configurations is a weighting, combining the individual fuzzy values of the strategies resulting from the inference such that a global fuzzy value for the overall configuration is obtained. This establishes an aggregation, several aggregation operators being defined in fuzzy theory. We apply the so-called WOWA operator (defined by [22]). It combines the advantages of simple weighting (potentially giving different significance to different strategies such as main and substrategies) and of ordered weighted averaging (potentially giving significance to the fact that some strategies suit the application characteristics very poorly and may impair overall performance).

Our prototype inference kernel is implemented in LISP and demonstrated on different application (sub-)domains and rules/membership-functions below.

EXPERIMENTAL RESULTS

Rule Definition and Configuration of Communication

This example demonstrates, on a small set of strategies, how rules can be established and applied to a concrete application (neural networks). Only communication strategies are considered. The machine environment in all cases is the MANNA distributed memory machine [8]. Two main strategies are selectable, namely message passing (via the operating system PEACE [8]) and fine-grained threaded calls (via the multithreaded runtime system EARTH [18]). EARTH is based on direct memory access, but we use a similar communication variant in PEACE

(transferring the bulk of data directly) to make the systems competitive at this point. EARTH can be considered a specialized implementation of communication for parallel machines. In both systems, different communication topologies such as a tree or a sequential loop may be used to spawn parallelism from a central master to a number of workers. Furthermore, with the EARTH system, one or more function-internal threads (at code-block level) may be used to overlap communication latency with computation. We show rules and performance results for this decision subspace.

We measured performance with microbenchmarks, obtaining a concrete performance curve (see Figure 4) that performs the task of a cost function. Complete configurations are measured, and differences in the choice of substrategies relate to this overall performance. Setting the number of machine nodes to 20, speedup depends then on task granularity and data size.

The hierarchical rule organization distinguishes first between message passing and fine-grained threaded calls, because this is by far the most dominant factor. Since threaded calls are a specialization, their performance is always superior. However, there are some performance ranges where the difference becomes critical, and others where message passing still performs sufficiently well. Besides, performance ranges in substrategies do not overlap. At the next stage, a decision is made between sequential or tree distribution. Ultimately, a decision is made between using a single or multiple threads. This is the only point at which performance ranges overlap, because multiple threads are beneficial in broad parameter ranges, but disadvantageous in others. The decisions are interdependent, and the resulting inference system thus needs two stages. No additional configuration rating is necessary in this simple tree. Note that the simple rule definitions used here can only properly distinguish rectangular areas in the solution space (areas here do in fact have nonlinear boundaries), but they make a sufficiently good approximation. Maximum uncertainty introduced by the simplification may be quantified and, for closer approximation, rules may be more fine-grained or superimposing each other (though then they are unlikely to be definable by hand and require automatic support).

Figure 4 also shows results from the application of the simple given rule subset (the speedup curve also shows numbers of machines nodes other than 20) to the configuration of unit parallelism in feedforward neural networks. This application has quite small granularities and large data sizes in communications, is static and regular in its behavior, and has $n:1$ and $1:n$ communications. Here, only fine-grained threaded calls are well-performing (otherwise the application is not parallelizable with reasonable speedup), tree organization is beneficial, and a single thread performs better.

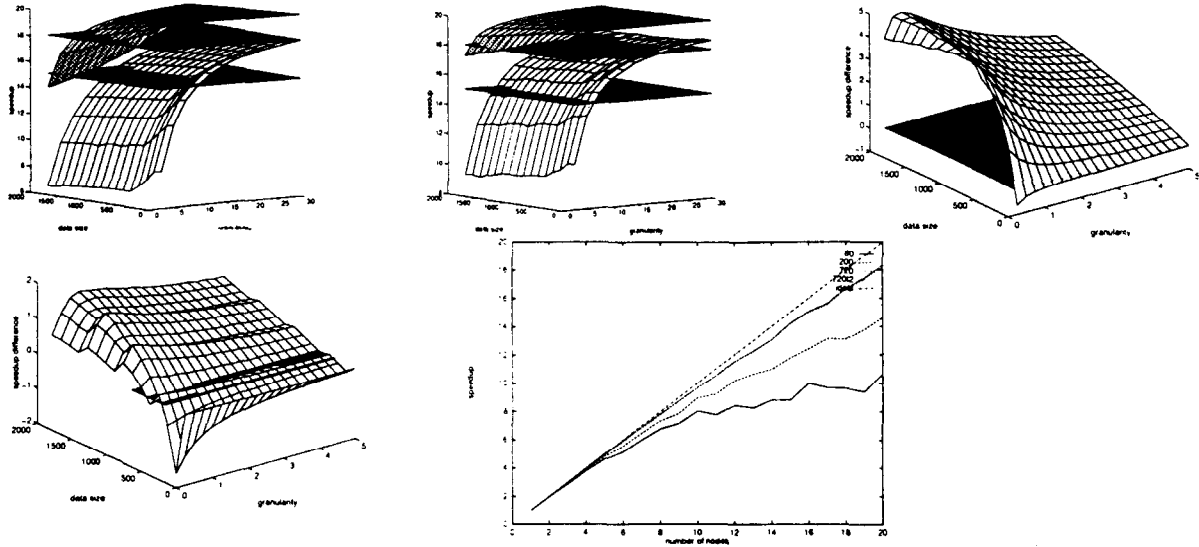


Figure 4: Speedups of loop and tree distribution (top row), each measured for EARTH (top curve) and PEACE (bottom curve); and speedup differences between loop and tree distribution in EARTH (top row, right) and speedup differences when using or not using multiple threads in sequential distribution in EARTH (bottom row, left). Data size is given in bytes, and granularity in ms. In addition, speedups for the concrete example of 3-layer feedforward neural networks (communications are 1:n and n:1) are shown for 80, 200, and 720 units per layer (bottom row, right). Granularity on 20 nodes for 200 units is 0.8 ms and data size 600 bytes.

```

IF (granularity = large) THEN (message-passing = applicable)
IF true THEN (threaded-calls = best-choice)
IF NOT (granularity = large) THEN (threaded-calls = true)

IF threaded-calls AND (comp-to-comm-ratio = large)
  THEN (sequential-distribution = applicable)
IF T THEN (tree-distribution = best-choice)
IF message-passing AND (granularity = very-large) THEN (sequential-distribution = applicable)

IF message-passing OR tree-distribution THEN (single-thread = true)
IF threaded-calls AND sequential-distribution
  AND (granularity = small) AND (argument-size = small)
  THEN (single-thread = true)
IF threaded-calls AND sequential-distribution
  AND (argument-size = medium-to-large) AND NOT (granularity = very-small)
  THEN (multiple-threads = best-choice)
IF threaded-calls AND sequential-distribution
  AND ((argument-size = small) OR (granularity = very-small)) THEN (single-thread = applicable)

```

```

granularity-large = [5, 25, ∞, ∞]
comp-to-comm-ratio-large = [3, 50, ∞, ∞]
granularity-very-large = [10, 30, ∞, ∞]
granularity-small = {0, 0, 1, 2}
granularity-very-small = {0, 0, 0.15, 0.25}
argument-size-small = [0, 0, 500, 750]
argument-size-medium-to-large = {500, 750, ∞, ∞}

```

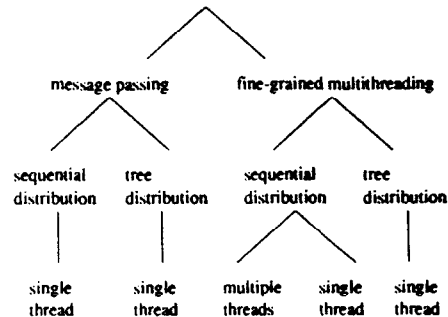


Figure 5: Rules for configuration, definition of trapezoidal membership functions (0 and 1 edges from left to right, time in ms, size in bytes), and the corresponding potential decision tree.

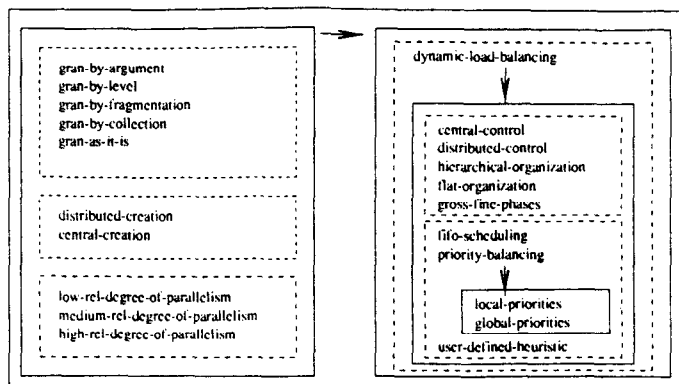


Figure 6: Strategy organization showing only a subset of strategies being shown. Arrows indicate sequentially or hierarchically dependent decisions.

Configuration of Dynamic Load Distribution

In [17], a set of rules was defined for the application domain of dynamic load balancing, scheduling, and dynamic determination of task granularities. 52 strategies were defined and 72 rules formulated, partially expressing hierarchical or sequential dependence (see Figure 6). They refer to top-down strategy selection, while for the characteristics part no rules were defined (only functional abstractions were used that compose several parameters). Compatibility constraints are described in 32 rules. The application characteristics considered are, e.g., computation times, argument sizes or dynamic tree irregularity (all dynamic parameters are obtained by monitoring and statistical evaluation). The original system has three stages, and two hierarchical layers corresponding to main and substrategies. Membership functions were defined manually and as having a trapezoidal shape, for the sake of simplicity. (Generally speaking, any membership function (a function in $[0,1]$) may be suitable.) The rules assumed a specific machine (MANNA) and communication system (PEACE message passing) and were largely heuristic. We are currently extending them to define proper metrics and integrate other communication systems like fine-grained (multi-)threading (EARTH)—for which we here give additional rules and an example of the system's use for the concrete Gröbner Basis application.

Gröbner Basis is a computer-algebra problem—roughly speaking a symbolic form of Gaussian elimination. The parallelization used here results in an inherently dynamic behavior, with asynchronous task creation and communication. Runtimes of tasks vary considerably and are unpredictable, i.e. there is no correlation to the argument size, the degrees of polynomials, etc., but the basic task granularities are cost-effective (being in the microseconds range), and no aggregation is therefore necessary. Moreover, tasks are created dynamically and their number is unpredictable, too. Only fine-grained threaded calls using multiple threads perform efficiently here. Dy-

amic load balancing is necessary as well. Using up to 20 machine nodes, a flat organization for the balancer is sufficient. Control may be either central or distributed, central control either dedicating a specific node for the balancer or running it together with normal work (i.e. letting all nodes contribute to the work). In addition, the application has central data structures, which means that a general decision has to be made with respect to all central activities. Application work depends to a significant degree on the order of execution. Priorities can be assigned, and priority balancing is necessary. Central balancing is favored (priority handling is then global, otherwise local). Central balancing is further reinforced by the fact that heuristic distributed assignment does not achieve good balancing because the number of tasks is fairly low. Load-info-based distributed balancing performs poorly with this application because the task attraction mechanism provided is not suitable. Figure 7 lists the rules that are essential for taking these decisions. The system rates a configuration with central control and global priority handling higher (0.825) than a configuration with distributed control and local priority handling (0.7). Letting all nodes contribute to the real work is advantageous in both cases (viewed on average over all node numbers)—rating is 0.025 lower otherwise. Figure 8 shows the different performances obtained with the various configurations. As can be seen, the system successfully obtained the best configuration.

SUMMARY AND DISCUSSION

We have presented a solution for the hierarchical and fuzzy configuration of systems using multiple strategies selected in accordance with the characteristics of the respective application. A rule-based approach exploiting knowledge about relations between characteristics and the appropriateness of strategies is applied in order to limit the search effort involved in the combinatorial problem. Thus, the approach—through its preselection of configurations per level—lies somewhere between pure combina-

```

IF (granularities = predictable) THEN (use-one-of-granularity-prediction-strategies = true)
IF (granularities = cost-effective) THEN (granularity-as-it-is = true)
IF (task-creation-rates = irregular) OR (task-granularities = irregular)
  THEN (dynamic-load-balancing = true)
IF (number-of-nodes = small-to-medium) THEN (flat-organization = true)
IF (number-of-nodes = large) THEN (hierarchical-organization = true)

IF dynamic-load-balancing THEN (distributed-control = most-likely), (central-control=maybe), (fifo-scheduling=most-likely)
IF (priority-influence-on-work = significant) THEN (priority-balancing = true)
IF priority-balancing AND (priorities = very-critical) AND (communication-overhead = acceptable) THEN (central-control = true)
IF distributed-control AND (number-of-tasks = low) THEN (heuristic-assignment=poor), (real-balancing=desirable)
IF distributed-control AND (number-of-tasks = medium-to-high) THEN (heuristic-assignment=maybe), (real-balancing=maybe)
IF (central-balancing OR (central-data = true)) AND (communication-reaction = fast) AND (service-time-relative-to-query = low)
  THEN (use-all-nodes-for-processing = true)
IF (central-balancing OR (central-data=true)) AND NOT ((communication-reaction=fast) AND (service-time-relative-to-query=low))
  THEN (dedicate-one-node-for-balancing = true)

use-one-of-granularity-control-strategies XOR granularity-as-it-is
NOT dynamic-load-balancing OR (flat-organization XOR hierarchical-organization)
NOT dynamic-load-balancing OR (central-control XOR distributed-control)
NOT dynamic-load-balancing OR (priority-balancing XOR fifo-scheduling)
NOT distributed-control OR (heuristic-assignment XOR real-balancing)
NOT dynamic-load-balancing OR (dedicate-one-node-for-balancing XOR use-all-nodes-for-processing)

```

Figure 7: Subset of selection and compatibility rules applying to the Gröbner Basis application.

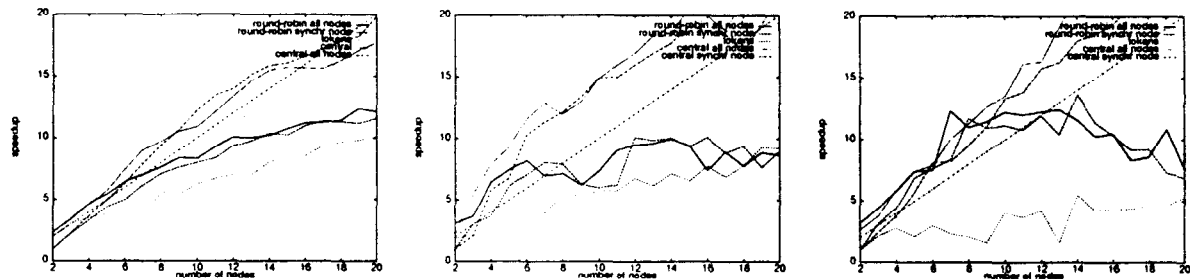


Figure 8: Speedups obtained with different configurations for the Gröbner Basis application with the different inputs Lazard, Katusura-4, and Katsura-5 (from left to right). The configurations shown are load-info-based distributed balancing (by EARTH's token mechanism), heuristical distributed balancing (round-robin), and central balancing using either all nodes for application work or dedicating one node for central control and maintenance tasks.

torial trials and direct derivation from knowledge. If the solution space is—as expected—structured and if near-optimal solutions cluster sparsely, a significant reduction in search effort can be achieved. The integration of fuzziness helps support abstract reasoning on coarse classifications of application behavior, leaving sufficient flexibility for finding near-optimal global solutions. Nevertheless, the approach is—even if full cost functions are available—only approximate, though it is systematic and based on sound theory, i.e. the maximum degree of fuzziness can be calculated if precise information is available.

A special multistage approach had to be developed as an extension to conventional fuzzy inference systems. This can be applied for sequential decisions per level as well as between hierarchical layers. The approach was developed for an inference kernel. demonstrated by the configuration of implementation strategies such as different dynamic load balancing strategies, but it can, in principle,—by defining domain-specific rules and membership functions—be applied to other configuration

problems as well.

References

- [1] Bianchini, R., Carrera E.V., and Kontothanassis, L.: The Interaction of Parallel Programming Constructs and Coherence Protocols, *Conf. on Principles and Practice of Parallel Progr. (PPoPP97)*, Las Vegas / USA, pp. 69-79, June 1997.
- [2] Bellman, R.E., and Zadeh, L.A.: Decision-making in a Fuzzy Environment, *Management Science*, pp. 141-164, 17(4) (1970).
- [3] Culler, D.E. and Singh, J.P., *Parallel Computer Architecture—A Hardware/Software Approach*. Morgan Kaufmann, 1998.
- [4] Ekmeçic, I., Tartalja, I., and Milutinovic, V.: A Survey of Heterogeneous Computing: Concepts and Systems, *Proceedings of the IEEE*, pp. 1127-1144, 84 (8), August 1996.

-
- [5] Di Martino, B., and Chapman, B.: Program Comprehension Techniques to Improve Automatic Parallelization, *Workshop on Automatic Data Layout and Performance Prediction*, Rice University, Houston (USA), Apr. 1995.
- [6] Foster, I., Kesselman, C., and Tuecke, S.: The Nexus Approach to Integrating Multithreading and Communication, *Journal of Parallel and Distributed Computing*, pp. 70-82, **37**, 1996.
- [7] Foster, I., and Kesselman, C.: Globus: A Meta-computing Infrastructure Toolkit, *Internat. Journal of Supercomputer Applications*, pp. 115-128, **21**(2), 1997.
- [8] Giloi, W.K., Brüning, U., and Schröder-Preikschat, W.: MANNA—A Prototype of a Distributed Memory Architecture With Maximized Sustained Perf., *Euromicro PDP Workshop*, 1996.
- [9] Kacprzyk, J., *Multistage Fuzzy Control*. Chichester, England: John Wiley and Sons, 1997.
- [10] Kafil, M., and Ahmad, I.: Optimal Task Assignment in Heterogenous Distributed Computing Systems, *IEEE Concurrency*, July-September 1998.
- [11] Kremer, U.: Optimal and Near-Optimal Solutions for Hard Compilation Problems, *Parallel Processing Letters*, **7** (4), 1997.
- [12] Klir, G.J., and Yuan, B., *Fuzzy Sets and Fuzzy Logic—Theory and Applications*. Upper Saddle River, New Jersey: Prentice Hall, 1995.
- [13] Maeda, H.: An Invest. on the Spread of Fuzz. in Multi-fold Multi-stage Approx. Reasoning by Pictorial Repr., *Fuzzy Sets and Systems*, pp. 133-148, **80**, 1996.
- [14] Mitchell, N., Högstedt, K., Carter, L., and Ferrante, J.: Quantifying the Multi-Level Nature of Tiling Interactions. *Internat. Journal of Parallel Programming*, to appear 1998.
- [15] Roychowdhury, S., and Wang, B.-H.: Cooperative neighbors in defuzzification, *Fuzzy Sets and Systems*, pp. 37-49, **78**, 1996.
- [16] Raghavachari, M., and Rogers, A.: Ace: Linguistic Mechanisms for Customizable Protocols, *Conf. on Principles and Practice of Parallel Progr. (PPoPP97)*, Las Vegas / USA, pp. 80-89, June 1997.
- [17] Sodan, A.C.: A Semi-Automatic Multiple-Strategy Approach to Mapping Tree-Structured Symbolic Processing Programs, *Journal of Symbolic Computation*. pp. 615-634, **21**, 1996.
- [18] Sodan, A.C., Gao, G.R., Maquelin, O., Schultz, J.-U., and Tian, X.-M.: Experiences with Non-Numeric Applications on Multithreaded Architectures. *Conf. on Principles and Practice of Parallel Progr. (PPoPP97)*, Las Vegas / USA, pp. 80-89, June 1997.
- [19] Sodan, A.C., and Torra, V.: Mapping Decisions by Fuzzy Inference. *ICA3PP*, Melbourne/Australia, pp. 405-418. Dec. 1997.
- [20] Sun, X.-H.: Performance Range Comparison via Crossing Point Analysis, in *Lecture Notes in Computer Science*, **1388** (J. Rolim, ed.), Springer, March 1998.
- [21] Torra, V., and Sodan, A.C.: A Multi-Stage System in Compilation Environments, *Fuzzy Sets and Systems*, to appear.
- [22] Torra, V.: The Weighted OWA Operator, *Int. J. of Intelligent Systems*, pp. 153-166, **12**, 1997.
- [23] Yager, R.R.: Knowledge-based defuzzification, *Fuzzy sets and systems*, pp. 177-185, **80**, 1996.
- [24] Xu, Z., Larus, J.R., and Miller, B.P.: Shared-Memory Performance Profiling, *Conf. on Principles and Practice of Parallel Progr. (PPoPP97)*, Las Vegas / USA, pp. 240-251, June 1997.

Angela Sodan obtained her Masters degree and Ph.D. from the Technical University of Berlin and currently holds a permanent research position at GMD. Her research interests include parallel processing, multithreaded systems, classification, and configuration.

Vicenç Torra obtained his Masters degree and Ph.D. from the Universitat Politècnica de Catalunya and is now professor at Universitat Rovira i Virgili. His research interests include fuzzy sets, aggregation operators, and multi-agent systems.