1988

Machine Knowledge Maniputation Issues in Parallel Compilers

Ko-Yang Wang

Dennis Gannon

Piyush Mehrotra

Report Number:
88-842

MACHINE KNOWLEDGE MANIPULATION ISSUES FOR PARALLEL COMPILERS

Ko-Yang Wang
Dennis Gannon
Piyush Mehrotra

CSD-TR-842
December 1988
(Revised December 1990)

MACHINE KNOWLEDGE MANIPULATION ISSUES FOR PARALLEL COMPILERS

Ko-Yang Wang

Department of Computer Sciences, Purdue University, West Lafayette, IN 47907.

Dennis Gannon

Department of Computer Science, Indiana University, Bloomington, IN 47405.

Piyush Mehrotra

ICASE, NASA Langley Research Center, Hampton, VA 23665.

ABSTRACT

Parallel compilers need detail architectural knowledge about the target machines to optimize user programs. The knowledge is needed in order to realize the potential parallelism provided by the hardware and to match the program parallelism with the machine parallelism. In most parallel compilers, the architectural information of the target machine are blurred into the control structures of the compilers. Consequently, these parallel compilers are inflexible and substantial efforts are needed to modify them or to port them to different machines. A solution to this problem is to separate the hardware features from the knowledge for the program optimization and describe the later based on these features. In this way, the control of parallel compilers relies explicitly on the computational model that is described by machine features rather than the hard-coded heuristics. High degree of flexibility, portability and knowledge sharing can be achieved among different target machines.

In this paper, a parallel machine knowledge representation scheme that features hierarchical structure and object oriented approach is presented. Under the scheme, machine features are represented as objects and are organized into hierarchical structures based on relationship between the feature objects. An abstraction process which translates basic machine features into different levels of abstraction is presented. Optimizing compilers can select the levels that suit its objectives and tasks most to work with. The parallel machine knowledge manipulation system is implemented in Prolog and includes mechanism for interactive feature specification, feature classification, and support for reasoning based on the features.

MACHINE KNOWLEDGE MANIPULATION ISSUES FOR PARALLEL COMPILERS

Ko-Yang Wang

Department of Computer Sciences, Purdue University, West Lafayette, IN 47907.

Dennis Gannon

Department of Computer Science, Indiana University, Bloomington, IN 47405.

Piyush Mehrotra

ICASE, NASA Langley Research Center, Hampton, VA 23665.

1. Introduction

Despite the decade long of effort that have gone into studying the utilization of parallelism, there is still a wide discrepancy between the available parallelism and the utilized parallelism. Software development continues to be the major problem in the realization of massive parallelism which was promised by the hardware advances. The difficulty lies with two problems: first, our lack of experiences in handling massive parallelism and secondly, the difficulty in integrating the huge amount of knowledge that is needed to optimize the parallelism. The problem is complicated by the fact that most parallel architectures use different tricks to speed up the computations. Programs need to be twisted extensively in order to match the underlying hardware and thus hurts the portability of the programs.

One trend that is emerging and has high potential to success is to build parallel programming environments that can optimize programs for different parallel architectures. Under this environment, the users can concentrate on improving algorithms to utilize higher fraction of potential parallelism of the system. The programming environments are left with the responsibility of transforming the programs to suit different target machines. This approach relieves the users from the ever complicated program optimization problem for parallel computers but its success relies on the richness of the expertise that the parallel

---

† Revised Sept. 1990.

programming environment possesses. Despite the widely accepted success of the vector compilers, the promises of parallel compilers and parallel programming environments have yet to be realized.

Like other software systems where many context dependent decisions need to be made, parallel compilers and programming environments can take advantages of the expert system technology where situation recognition with conflict resolution strategies rather than "algorithmic" control is used. Knowledge based compilers control the optimization of the programs based on the context of program and target machine. It employs many program optimization heuristics to restructure the program to explore the machine parallelism. The knowledge based approach has the following advantages: adding or modifying knowledge is easy, incomplete knowledge can be incorporated, and knowledge can be built up incrementally. The reasoning ability and learning potential of the knowledge based approach makes it possible to build intelligent parallel compilers that traditional compiler technologies are unable to reach.

One important step that is often overlooked in building parallel compilers is to isolate machine dependent features from the optimization heuristics. Successful implementation of the knowledge based compilers and multiple target machines parallel compilers relies on a suitable knowledge representation and processing scheme for both the program optimization and machine parallelism knowledge. Without the machine knowledge separating process, the implementation of multiple target machine programming environment would be practically impossible.

One problem that makes optimizing programs for parallel architectures hard is that the decision trees of the transformations are normally quite large for even small program modules. Exhausting all paths of possible transformations is usually too costly to be realistic. Applying transformations blindly may actually decrease the degree of parallelism instead of increasing it. Most program transformation systems "solve" this problem by using pre-determined paths to avoid exhausted searches or allowing user interaction, assertions or directions to guide the compiler. In other words, these program restructuring systems either force the programmers to settle with non-optimal solutions or ask them to make the hard decisions.

It is apparent that a real solution is to have intelligence built into the parallel compiler so that it can make these hard decisions intelligently with minimal intervening from the users. One key issue is the balance between the available resources (the computing power that the user is willing to pay for optimization) and the degree of optimization obtained. In other words, it does not only need to make good decisions, it also has to derive the decisions within a reasonable time and cost. To achieve this goal, the parallel compilers and programming environments need to have a good machine knowledge representation scheme to support its "intelligent operations."

1.1. Feature-Directed Program Optimization

Methodologies for optimizing parallelism on a particular parallel machine can often be linked to features of the architecture. Under the *feature-directed program optimization model [Wang90d]* optimizing parallel compilers use the features of the program and machine *explicitly* to control the restructuring of the programs. Unlike other parallel compilers where decisions for program optimization are based on the implicit heuristics that are hardwired and scattered in the programs, this approach allows the compiler to base its decisions on features of both the target machine and the program. In this way, the compiler is actually *"programmed"* by the features of the chosen target machine and the program to be optimized. For example, figure 1 shows the flow graph of a simple heuristic for loop blocking that is based on machine features.

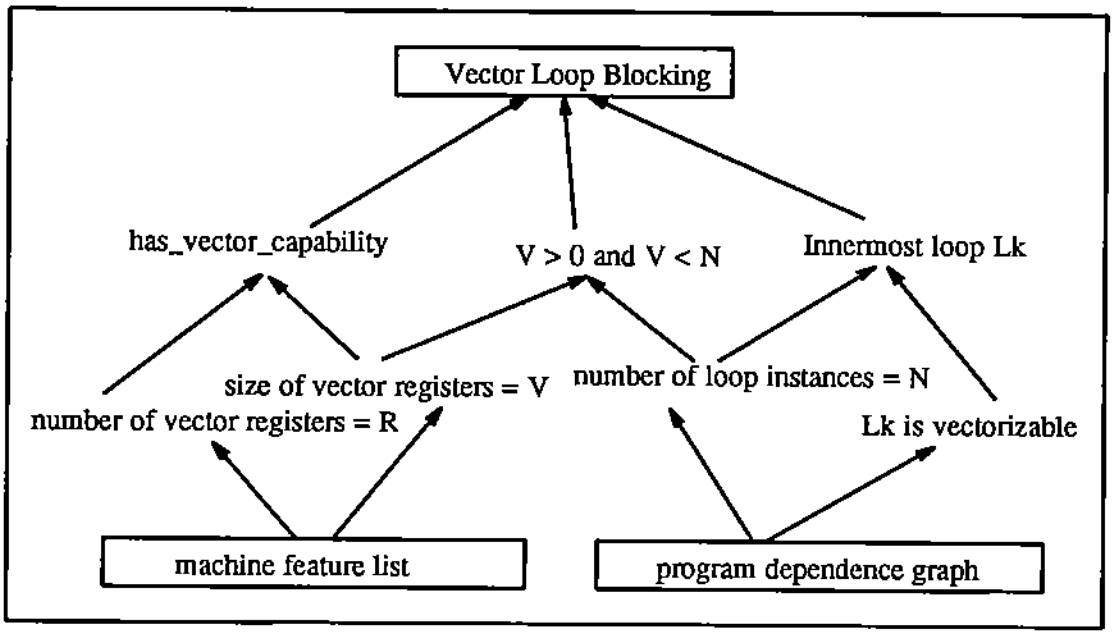


Figure 1. A simple example to illustrate feature-directed program optimization.

The feature-directed program optimization model allows one to build generic parallel program optimization tools and retargetable parallel compilers that can be applied to (and perform well on) a wide variety of parallel machines.

The effectiveness of knowledge-based compilers and multiple target parallel compilers relies on a rich set of program transformation techniques and knowledge to utilize these techniques to improve the parallelism of the program for the target machine. Successful implementation of such compilers requires a suitable knowledge representation and processing schemes for representation and manipulation of the machine parallelism and program optimization knowledge.

The machine knowledge representation scheme needs to provide a foundation for the integration and organization of the program optimization knowledge and supports performance evaluation and reasoning. In section 2, machine features and their uses in parallel compilers are discussed. In section 3, we present an object-oriented machine knowledge representation scheme which features modular knowledge representation, various degrees of abstraction, and hierarchical reasoning.

Recognizing and collecting useful heuristics and analyzing and separating machine features from the program optimization heuristics are very involved jobs. A good

knowledge manipulation system can help knowledge engineers comb through the complex and ill-organized knowledge and identify the essential elements of the knowledge and help them transform fragmented heuristics into well-defined programs. Automatic tools to help knowledge engineers to program parallel compilers are highly desired and are long overdue. In section 4 we introduce a machine knowledge manipulation system that is based on the knowledge representation scheme discussed in section 3. The machine knowledge manipulation system can be used interactively to analyze heuristics for optimizing parallelism, comparing machine features, abstracting new machine knowledge, and installing new target machines.

2. Machine Features and Parallel Compilers

2.1. Machine Features

Properties of the target machine that affect the concurrent execution of the machine are called *machine features*. Definition of the machine features records the distinct properties of the architecture that need to be considered in utilizing the parallelism on the architecture. The manipulation of machine knowledge includes representation and organization of the machine features, inference and modification of the machine features, and supports for reasoning based on the machine features.

A detailed discussion of the machine features and their effects on program transformation was presented in [WaGa89].

2.2. Building up Understanding of Parallelism From Machine Features

A collection of machine features of a parallel computer is usually fragmented and lack a complete understanding of the whole architecture that the features describe. In order for the system to build up an understanding of the target machine, the knowledge needs to be connected by relationships between the features and the knowledge of how these features affect the parallelism of the target machine. This implies that the knowledge representation will have to support the composition of the knowledge and mold pieces of the knowledge together to obtain a whole view. To avoid a tedious specification process, the task of finding relationships between features should be done only once for each pair

of features and not repeated for other machines.

In section 3, an object-oriented knowledge representation scheme which allows one to build up the structure among the features is described.

2.3. The Uses of Machine Features in Parallel Compilers

The machine features can be used in almost every aspect of parallel compilers. In [WaGa89] we described an expert-system based parallel compiler that encodes expert knowledge based on the program and machine features. The heuristics are analyzed and the features of the machine are explicitly represented in the rules that guide the optimization process. Optimizing the program on a target machine requires a cheap but accurate performance prediction mechanism to estimate the performance of a program or effects of a program transformation on the execution. A performance prediction model for parallel compilers that uses machine features to choose performance characterization factors and compute evaluation functions is discussed in [Wang90a].

An intelligent parallel compiler, that implements the feature-directed program optimization, uses machine features and the performance prediction model to guide the systematic state-space search and planning or the heuristic-guided decision making [Wang90d]. The selection and application of transformations (such as array reshaping [Wang90b], message consolidation [Wang90c], etc) are also relied heavily on the features of the target machine.

As opposite to the stepwise refinement of program transformations, the pre-optimized algorithm substitution approach [Wang90e], replaces fragment of a program with an algorithm that are pre-optimized for the target machine. Machine features are used to choose the best applicable pre-optimized algorithms or to fine-tune the selected pre-optimized algorithm to match the target architecture that the algorithm may not be optimized for.

3. Design Considerations for Machine Knowledge Representation Schemes

Few design decisions need to be made to support the reasoning capability of the feature-directed program restructuring model. Several questions need to be considered. Should the machine knowledge base be organized as a flat-structured feature list or a

hierarchical-structured tree? Should the representation allow implicit knowledge implication or should all relationships be spelled out explicitly? Does the compiler need to have knowledge for multiple parallel architectures or, like traditional parallel compilers, is only information about a particular target machine needed? These decisions affect both the power and elegance of the representation scheme. We chose to incorporate knowledge of multiple parallel machines into one system, which allows hierarchical organization of features and implicit and explicit knowledge implication.

Flat structure vs. hierarchical structure

Representing the machine features in a flat structure has the advantage of being simple and explicit. In [WaGa87], a flat-structured, uniform machine feature representation scheme is used to represent and model parallel computers. All basic features are represented as facts in the database. When a target machine is specified, the features of the machine are loaded into the kernel of the expert system. The features are abstracted by applying a set of rules to the facts. This approach is simple and yet powerful enough to model the parallel architectures. Specification of the machine can be done feature by feature and is straightforward. However, the flat-structure representation does not have a mechanism to show the relation and interaction between machine features. The relations between the features must be encoded by using other constructs such as the rules and preconditions used in the above system. To find the relationship between particular features, one must exhaust the rules to find the rules that define the relation between them. This makes manipulation and maintenance of the system an involved task.

A hierarchical knowledge representation scheme incorporates the relationship between knowledge into the representation of the knowledge. The relationships between the machine features are mostly architecture independent and can be inherited from the previous known structures. Therefore, they do not need to be redefined when a new target architecture is defined. On the other hand, it is important to have the ability to define new relationships so that similar architectures can share most of the knowledge but are allowed to specify the differences explicitly.

To effectively support the hierarchical knowledge representation, the knowledge representation scheme should support abstraction and classification. Classification closely

groups relative features into distinct classes that possess special features, whereas abstraction defines relationships between features of different levels. To visualize it, one can imagine that features are in a two-dimensional space; classification is a horizontal operation that groups similar features together and abstraction is a vertical operation that defines the relationship between different levels. For example, figure 3 (b) shows an organization of the machine features; the vertical dimension shows the abstraction level of the architecture and the horizontal dimension shows that related features are grouped together to form the base of the abstraction. Together, classification and abstraction provide a powerful mechanism for the integration of feature knowledge. Our representation scheme defines the hierarchical structure by using the relationship between the subclass and superclass plus relationship functions. Hierarchical and flat structured object modeling can done or even converted by modification of relations.

Explicit vs implicit knowledge representation

Knowledge of the machine can be explicitly expressed or implied by other knowledge. Explicit knowledge has the advantage of being simple and explicit, but may contain redundant knowledge and inflate the size of the knowledge base. On the other hand, allowing implicit knowledge representation leads to more concise representation but increases the difficulty of maintaining the knowledge base. To balance the tradeoff, it is a good idea to make all knowledge implication rules explicit.

For example, in the following example, the feature *cost ratio of global memory access and computation* is defined to be the ratio of the cost of a global memory fetch and the cost of a floating point multiply. This property of the machine can be defined explicitly or implicitly by the features *cost of a global memory fetch, cost of a floating point multiply,* and rule 1 which explicitly defines the relation.

Rule 1 feature_value('cost ratio of global memory access and computation', R) :-
feature_value('cost of a global memory fetch', F),
feature_value('cost of a floating point multiply', M),
R is F / M.

The advantage of spelling out the relationship instead of specifying the value for the information is that when the feature is to be modified (for example, when the memory is upgraded into fast memory), the feature *cost ratio of global memory access and computation* does not need to be redefined. On the other hand, when doing the program optimization on a particular machine, it is the relative cost ratio that will affect the decisions. So when the actual costs of memory access and computation are not required, the value of the ratio can be specified explicitly.

Multiple target vs single target parallel compilers

Multiple target parallel compilers have many advantages over single target parallel compilers: knowledge sharing, procedure sharing, organization of knowledge based on machine features involved, comparison of performance on different architectures under the same platform, etc. One advantage that single target parallel compilers have over multiple target compilers lies in the efficiency: they are less expensive in feature access and knowledge reasoning since all knowledge that has no relation with the target machine does not need to be considered. We discuss in section 5.4 a methodology that achieves the advantages of multiple target software systems but suffers no performance penalties.

4. An Object-Oriented Knowledge Representation Scheme For Parallel Computers

What kind of a machine knowledge representation scheme does an intelligent parallel compiler require? From our point of view, an intelligent parallel compiler needs a ''simple'' machine knowledge representation scheme that can support reasoning, knowledge abstraction, organization, and heuristic comparison. The scheme we describe here is based on the object-oriented knowledge representation paradigm, in which features about machines are treated as objects and the understanding of a machine can be composed from feature objects.

The object-oriented paradigm is a sound knowledge representation methodology. Its modularity and hierarchical abstraction capability make it particularly appealing for the representation of complicated real world knowledge, such as parallelism. The inheritance and chaining of the paradigm allow compact and elegant representation of the relationships between objects; this also makes porting the system to new machines easy. On the other

hand, just like any other knowledge representation paradigms, there are many tradeoff in the object-oriented paradigm and design decisions in implementation that must be made based on the application domain. In this paper, we concentrate on just one application domain -- multiple-target parallel compilers -- although most of the principles and methodologies we have discussed here can be applied or extended to other problem domains as well.

Our knowledge representation scheme consists of three elements: the feature objects, relationship between objects, and operators on the objects. Under this scheme, the knowledge of the parallel computers can be decomposed into features and reassembled into hierarchical models based on feature organization and abstraction techniques. This knowledge representation scheme provides a vehicle for heuristic manipulation and intelligent compiler construction.

4.1. Feature Objects

We followed the convention of the object-oriented paradigm by which objects representing machine features are represented by a set of basic objects that we call *feature objects*. Feature objects are classes of objects that are the basic units for defining properties of parallel computers. A feature object represents a particular property of the target architecture and has slots to store information about the property. These slots are called the attributes of the feature object. The structure of the feature object varies based on the type of the property it represents. Some attributes of the features are common to all features and some apply only to certain features. The template of feature objects is defined by a meta-class that we call a *feature class*.

Each entry in the feature class defines possible values for feature objects. An instance of the feature class identifies the properties of a machine feature. These properties include the name of the feature, type of the feature (used for type checking), conditions for this feature to be meaningful, relation of the feature with other features, and the attributes of this feature. The feature object, which defines what the instances of the feature should be, is the foundation of the representation.

class 'vector registers' subclass_of 'processor' instance_of 'register' with

pre-conditions: has_vector_capabilities,

static_dynamic: static,

type: [boolean, true],

number: [integer, 64],

size: [integer, 32].

feature name:	vector registers	
pre-conditions:	has_vector_capabilities	
static/dynamic:	static	
relations:	parent(vector_capabilities)	
feature type:	boolean	
feature value:	true	
attributes:	type	value
number of vector registers	integer	64
size of vector registers	integer	32
other attributes:	omitted	

Figure 2. The definition of the feature object "vector registers."

The example shown in figure 2 defines a feature named *vector registers*. In the example, the feature object *vector registers* is active only when the target machine has vector capabilities. This object is an instance of the object *register*. Slots in the first part of the template are common for all feature objects (inherited from the class *feature object).* Slots in the second part are the attributes of the feature.

An individual property described by an object is called the instance of the object. An instance of a feature object is defined when a value is associated with the object by an

feature assignment statement: *featureDefine*. For example, the following Prolog function causes a message to be sent to the object "vector registers" and sets the value of attributes of the object to be 64, 32, respectively.

featureDefine('vector registers', 'number', 64)
featureDefine('vector registers', 'size', 32)

When defining the machine features, the following alternative form is also accepted:

the 'number' of 'vector registers' is 64.
the 'size' of 'vector registers' is 32.

The statement *featureValue* provides a uniform way of accessing the feature instances. For example, *featureValue('vector registers', 'number', N)* returns the value of the current instance of the 'number' of the feature 'vector registers' in variable N.

4.2. Attributes Associated with Objects

New attributes can be associated with a feature object through the feature-attribute assignment statements:

featureAttribute(ObjectName, AttributeName)
featureAttribute(ObjectName, AttributeName, Type, DefaultValue).

The first statement defines the relationship between the attribute and the feature, and in the second statement both the relationship and the value are defined. A feature attribute assignment statement explicitly and dynamically defines the binding between the feature object and its properties.

A object defined to be dynamic may have more than one instance but only one instance can be *current*. In some cases, allowing more than one instance of the same object provides a degree of "non-determinism." This is useful when the target machine contains multiple characteristics. For example, on hypercube computers, the communication network can simulate different kinds of networks such as rings, meshes, shuffle-exchange networks, and trees. Different algorithms can utilize any one of the communication patterns. On the other hand, the current instance mechanism provides a way to obtain the deterministic effects.

4.3. Feature Organization

Feature objects form the basis of the representation for parallel computers. In the real world, knowledge of objects is not isolated. Instead, relationships exist between knowledge on different levels of abstraction. Therefore, representation of the relationships between objects using a higher level of abstraction and organization is needed so that the relationships can be recorded and manipulated. From the bottom-up approach, feature objects of similar properties can be grouped to form a feature object of higher level. From the top-down approach, one may decompose the feature objects into more detailed descriptions and continue expanding until the feature becomes a basic fact. With either approach, the feature classes are organized into hierarchical structures based on the relationship abstraction of the features that we call *relation functions*. The relation functions explicitly define the relation between feature objects of different levels. This includes predefined relations such as parent, children, exclude, complement, associate, compose, or user-defined relation functions. The relation functions serve two purposes. First they define the relationship between objects; second, they define the flow of control and messages. The collection of the relation functions organizes the machine knowledge into a hierarchy of features. Some of the relationships, such as parent and children, are inherited from the organization of the hierarchy and are defined by the subclass_of or instance_of relations; others need to be explicitly defined.

The organization of the features can simplify the representation of the features. For example, the pre-condition 'has_vector_capabilities' listed in the example in figure 2 is redundant because the feature *vector registers* is the descendant of the feature *vector capabilities,* and this pre-condition is actually implicitly encoded in the hierarchy of the features.

4.4. Operations On The Objects

4.4.1. Inheritance, Specification and Qualification

The notion of classes and meta-classes provides a mechanism for sharing information between different objects via inheritance. *Inheritance* means that the properties of a class are shared by instances of the class. On the other hand, different properties of the class

can be applied to different instances of the class by *specification.*

Inheritance and specification are used to group closely related knowledge into the same class so that the information can be accessed locally in the system. Specification helps to distinguish objects in the same class, while inheritance keeps the size of the system manageable.

Inheritance and specification are usually associated with *qualification.* That is, an inheritance or specification for an attribute is applied to an instance of a class when certain conditions are satisfied. For example, in distributed computing, achieving a balance between computation and communication is important. And communication with processors that are far away is normally discriminated against. These heuristics can be inherited by knowledge of all distributed computers.

class 'distributed memory' subclass_of 'memory hierarchy'
instance_of 'memory' with
.....
heuristics: [heuristic('balance computation and communication ratio'),
heuristic('avoid far access'), ...],
.....

However, when the cost ratio of far-access and near-access is close to 1 (as in the second generation of hypercube machines such as iPSC/2 or NCUBE/2), sending messages to far away processes does not incur a significant penalty. In this case, the restriction can be lifted by changing the attribute 'avoid far access' to 'far access OK' as below: In the following example, the heuristic 'far access OK' is a specification that overwrites the inherited heuristic 'avoid far access'; and the conditions in the qualification statement *in_case* validate the specification statement.

```
'distributedMemory' instance_of 'distributed memory' with
    ..............
    heuristics: [
      heuristic('far access OK',
        in_case (featureValue(distributedMemory, 'far/near access ratio', A),
              isSmall(A-1.0)))
        ..... ],
    ..............
```

4.4.2. Feature Modification

As we discussed in section 3, the dynamic update of machine features is desirable to provide flexibility to match algorithm decomposition and reasoning. Possible modifications to a feature object include changing the feature value, changing the feature attributes, and changing the heuristics associated with the feature. An instance of a feature object can be modified by the featureDefine statement that we described above. For example, suppose the target machine is a hypercube computer. Since the hypercube can simulate other network topology (like mesh, trees, shuffle exchange networks, etc), at certain stages of the algorithm, a particular type of communication topology may match the algorithm better than others. The communication pattern can be easily changed by two featureDefine statements as below:

```
featureDefine(networkTopology, network, mesh, OldTopology).
  ..... communication using mesh .....
featureDefine(networkTopology, network, OldTopology, _).
  ..... communication using OldTopology .....
```

The feature modification operation can only be applied to the tunable properties of the architecture and attempts to modify static features will fail.

4.4.3. State Adjustment with Dependences

One problem that arises from object modification involves maintaining integrity and consistency. In our object-oriented scheme, this problem is addressed by providing a dependence-mechanism to notify an object of changes in related objects. The relation functions explicitly define the dependence relations between the objects involved. When the source of the dependence is changed, the target object is notified. And the target object may examine the change to decide whether to change its own state or not. For example, the heuristic object *utilizing vector registers* depends on the feature objects *available vector registers* and *vector operations*. If the object *available vector registers* is changed and no vector register is available, then the object *utilizing vector registers* needs either to move certain data in vector registers to memory to reclaim the vector register for the next operand or get the next operand from the memory. What action to take depends on the heuristics in the heuristic object.

4.5. A Simple Example

The specification of a feature involves defining the template (class) and the value (instance) of the object. The formal is normally the task of the system implementor and the latter is the task of system maintainer who installs a new machine. For example, the following is a fragment of the program that specifies the template for the global memory in a shared-memory architecture.

```
class memory_hierarchy with
        type: one_of [shared,distributed,hybrid],
        structure: list_of [global,cluster,local,cache].

class memory with
        type: one_of [shared,distributed,hybrid],
        size: integer,    % 1 unit = 1K bytes
        ratio_of_fetch_and_multiply_op: real,
        ratio_of_fetch_and_register_fetch: real,
        interleave: [integer, 4],
        read_cost: real,
        write_cost: real,
        prefetch: [boolean, false],
        prefetch_size: integer in case prefetch == true,
        connection: one_of [bus, network], % to processor
        network_topology: one_of [hypercube, omega, crossbar, ...]
                in_case connection == network.

class local_memory instance_of memory subclass_of memory_hierarchy with
        type: local,

size: [integer,4000],      % specify default value
        connection: bus.

class global_memory instance_of memory subclass_of memory_hierarchy with
        type: global,
        size: [integer, 16000], % default=16Mbytes
        connection: network.
```

Under the above definition, the local memory on a NCUBE/2 can be defined as:

memoryHierarchy instance_of memory_hierarchy with

type: distributed,

structure: [local].

localMemory instance_of local_memory with

size: 1000,

size: 4000 in case pid < 16, % has uneven memory distribution

read_cost:

write_cost:

prefetch: true,

connection: bus.

The entries can be accessed or updated in the following way:

featureValue(localMemory, size, K).

the size of localMemory is K.

the connection of localMemory is bus.

An interactive feature-specification scheme is described in section 6.2 as an alternative way to interface with the machine feature manipulation system. Different people involved with the system can choose different interface scheme to use. NH 2 Features of the Parallel Machine Knowledge Representation Scheme

The most important feature of our knowledge manipulation scheme is in the flexibility and the modularity of the scheme. The representation scheme can be used to achieve the following features.

1. Allows dynamic modification of machine knowledge.
2. Has high flexibility in knowledge characterization and organization.
3. Has various abstraction levels of the knowledge.
4. Supports knowledge hiding and global visibility.
5. Supports inference and knowledge encapsulation.

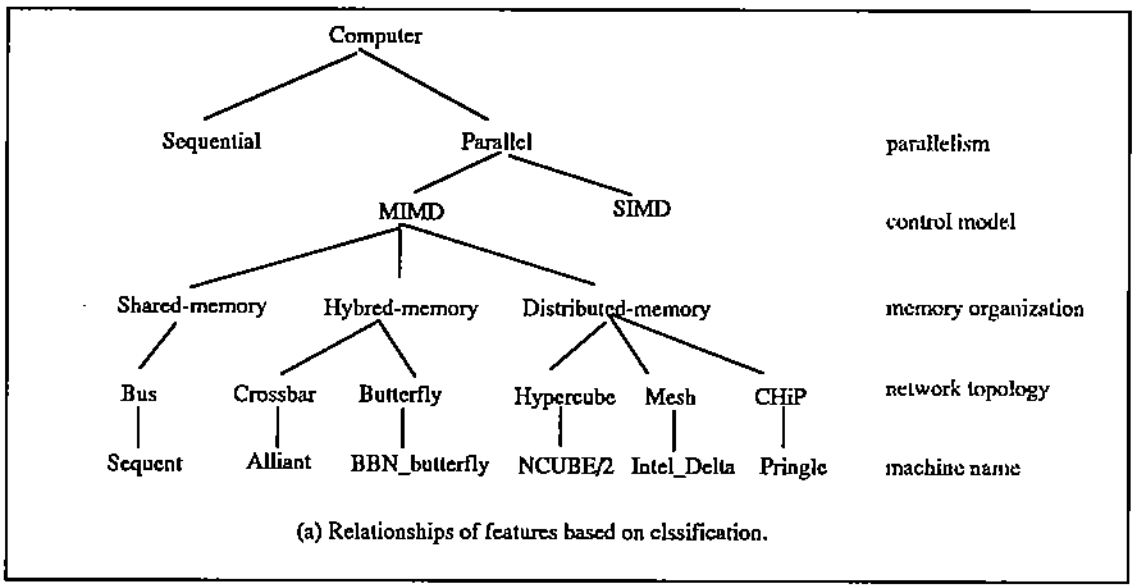
Static and dynamic knowledge representation

Although most features of a parallel architecture are fixed after the configuration of the system is set up, some features of the machine should be allowed to change dynamically. Allowing dynamic modification to some machine features has the following advantages. First, conceptual decomposition of the machine is possible. This means that the programmer can decompose the computational model to match the algorithm decomposition such as divide-and-conquer algorithms. Second, users are allowed to define "virtual machines" based on the existing machine features and knowledge. This is ideal for testing new machine designs or programming heuristics. Third, computer vendors usually provide many different configurations to suit specific needs of the users. Fourth, specification for similar architectures can be more elegant and compact. The representation scheme we defined above allows the representation of both static and dynamic knowledge in a uniform structure. Machine features that can be modified at run time are marked as dynamic features by setting the attribute dynamicFeature to be true. Dynamic machine features include the number of processors used in computing, system load, algorithmatic network topologies, and task control strategies. Features that are static overtime are termed static features and attempts to modify static features will be rejected by the system.

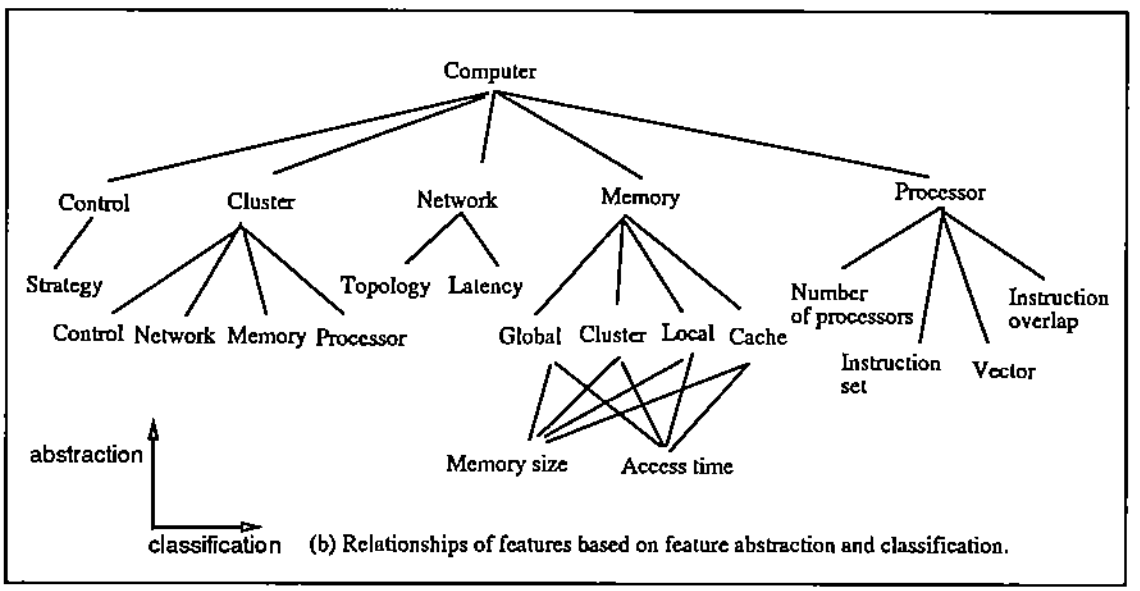
Flexibility in knowledge characterization and organization

Knowledge characterization is the basis for knowledge organization. Machine features can be characterized in different ways under different constraints. For example, features of a parallel computer can be categorized by the physical component modules (such as processing units, memory modules, communication medium, clusters, and control) or by the functionality of the features (such as program partitioning, data decomposition, process scheduling, memory utilization, communication minimizing, and synchronization). Each of these modules can be further characterized by smaller modules. For instance, the physical configuration of the memory hierarchy is composed of global memory modules, cluster memory modules, local memory modules, and cache memory modules. Features of the modules can be refined by further decomposition. Different ways of organizing the machine features have different advantages and tradeoff. Our knowledge representation

scheme allows the application to choose the desired way of organizing the knowledge according to the requirements of the application. Figure 3 shows two different ways to represent the hierarchical structures of the parallel computers based on different characterizing schemes.



(a) Relationships of features based on classification.



(b) Relationships of features based on feature abstraction and classification.

Figure 3. Two highly simplified examples for characterizing parallel architectures.

In figure 3 (a) the organization is simpler but features are not as detailed as the ones shown in figure 3 (b). For applications requiring minor optimization, the one in figure 3 (a) may be enough, but for more complicated optimization, a finer representation such as the one in figure 3 (b) would be better. As a matter of fact, the representation can be

refined by adding more feature objects and relationship functions as the system implementation progresses. Thus, this refinement can be done incrementally without affecting the part of the software that has already been done, although the behavior of the system should be improved with more detailed machine knowledge. One good strategy in adding new machine features is to check if this added feature can distinguish between conflicting knowledge and help in making a better decision. On the other hand, new features may lead to discovery of new relationships or heuristics.

Various abstraction levels of features

A key idea in automatic generation of high performance parallel programs is to express the knowledge of the machine and the program at an appropriate level of abstraction. Abstracted features of the machines range from high level concepts such as "shared-memory MIMD" or "distributed memory MIMD" or "topology of the interconnection network" to detailed properties such as "vector startup time" or "memory reference costs." The most appropriate abstraction level for the specification of the machine knowledge depends on the current state and goals of the compiler and the types of applications that utilize the parallelism of the machine. Different types of applications will require different levels of abstraction to express the computation model of the application. In some cases, it is determined by the extent of optimization that the compiler is seeking. For example, to decide the patterns of data movement, only the topology of the communication network is required. But to get the optimal data movement, other information such as the dimension of the hypercube, the startup and unit costs for message transfers, network protocols, and optimal communication/computation ratio for the architecture are needed. Allowing different abstraction levels simplifies the implementation of parallel compilers, since different tasks of the compilers can deal with computational models at different abstraction levels that are suitable for the tasks. On the other hand, it complicates the implementation of the knowledge representation. Our program representation scheme allows dynamic abstraction of the knowledge by encoding the relationship between the knowledge at different abstraction levels. And the machine knowledge can be mapped to the appropriate abstraction level at the run time of the compiler.

Global visibility and knowledge hiding

Data abstraction hides the internal data representation of an object from the uses of the object. This concept has been proven to be useful in conceptual abstraction of the objects and in the practice of modular programming. On the other hand, there are many cases where the ability to access the states of the objects is desirable. For example, the objects that define the relationships among objects are better visible globally. This is particularly useful in keeping the representation scheme flexible enough so that accommodating new architectures is easy. The framework of the representation needs to provide local readability so that the features can be assessed and understood individually. Our representation scheme provides some mechanisms for knowledge to be either hidden in objects or to be visualized globally. Inheritance is a mechanism that allows compact representation and also provides information hiding. A two directional object-value search provides the function of global visibility, while uniform access procedures provide knowledge hiding.

Inference support and knowledge encapsulation

The knowledge representation scheme needs to support reasoning for intelligent decision making. This includes a uniform structure for systematic access and support for representing heuristics. The properties of the system can be accessible to external systems through queries, but the details of the representation should be invisible to the external systems. The knowledge representation scheme provides a hook to link the related knowledge to basic machine features. This knowledge encapsulation is combined with the hierarchical structure so that a walk through the hierarchy gives a complete picture of the architecture. This feature can be used to limit the scope of the system to a single target machine and it creates a personalized system as described in section 5.4.

The object-oriented methodologies we adopted make the knowledge encapsulation modular and provide a good foundation for reasoning supports. Intermediate results created by the inference mechanism based on the machine features can also be treated as machine features and receive a uniform treatment.

5. Implementation of the Knowledge Representation System

5.1. A Machine Knowledge Representation System

The machine knowledge representation system consists of a machine feature database, an inference engine, an SQL relational database interpreter, and an interactive machine feature specification system.

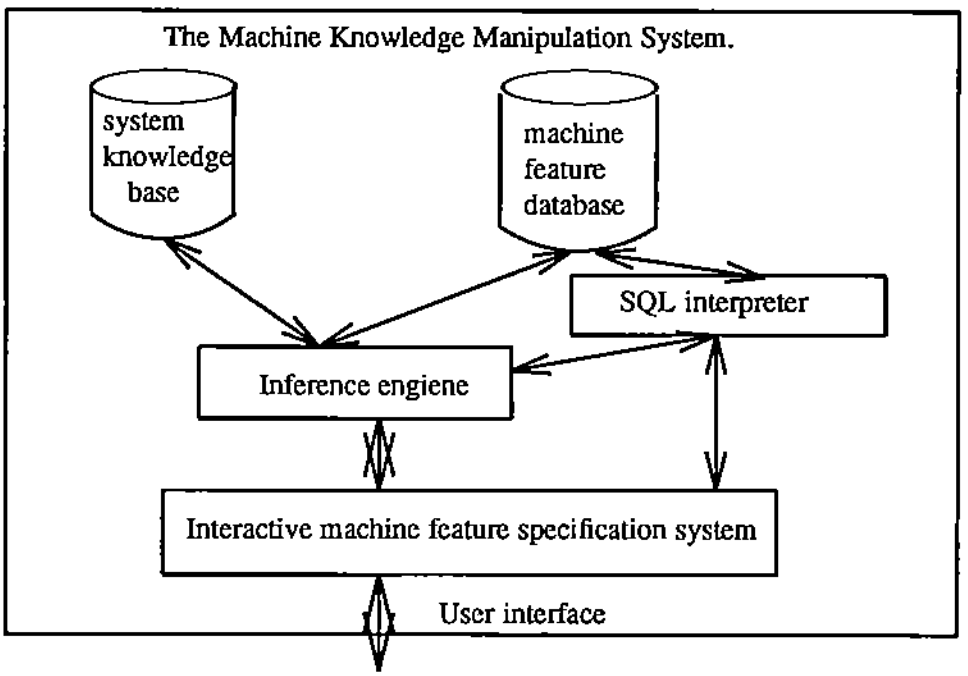


Figure 4. The machine knowledge manipulation system.

The machine feature database contains three kinds of knowledge about machine features: knowledge for feature definitions, knowledge for feature usages, and knowledge of features of parallel computers. The inference engine can be used to compare and deduce features to help the specification and classification of the features. A subset of the SQL relational database language is implemented in Prolog to compare the features of the machines and help the knowledge expert to abstract machine features from heuristics; this provides a powerful mechanism for the manipulation of machine knowledge. The interactive machine feature specification system provides the man-machine interface for interactive specification and manipulation of the features of parallel computers. The interactive machine feature specification system is interfaced to both the SQL database server and the inference engine so that the user can query or analyze features of the machines.

The machine knowledge manipulation system is implemented on top of the C-Prolog. The system is menu-driven and the user is allowed to pick a fact or predicate known to the system from the menu or to specify a new fact interactively. An interface to an X-Window front-end is under construction. Details of the procedures for machine feature installation are given in the next section.

To effectively utilize parallel computers, a program optimization system needs to have enough knowledge in two areas: the hardware features of the machine and the heuristics of using the machine. The machine knowledge manipulation system can assist the parallel compiler writer to manipulate the parallel machine knowledge by providing the following functionalities: specification of new machine features to the system, specification of machine features for a new machine, finding relations of a feature with other features, comparing features of different machines, and supporting the reasoning for intelligent compilers. The last three abilities are especially important when analyzing the machine features to construct the system or collecting new heuristics to enhance the capability of the system.

The process of installing new knowledge includes identifying, translating and representing new machine features and heuristics. Human interaction is needed for this process, but systematic system assistance can reduce the complexity of the task significantly.

5.2. Machine Feature Abstraction and Installation

The process of collecting machine features is illustrated in figure 5. The figure shows four components in the process: installing features for the machine, installing heuristics for the machine, installing new machine features to the system, and installing new heuristics to the system. These procedures are interrelated and can be carried out interactively.

5.2.1. Interactive Machine Feature Specification

New machines can be added to the system interactively with a user interface that allows the user to specify the features one by one. The system composes queries to ask for the features of the new machine based on its knowledge in its database and the machine features specified so far. A top-down approach based on the hierarchical

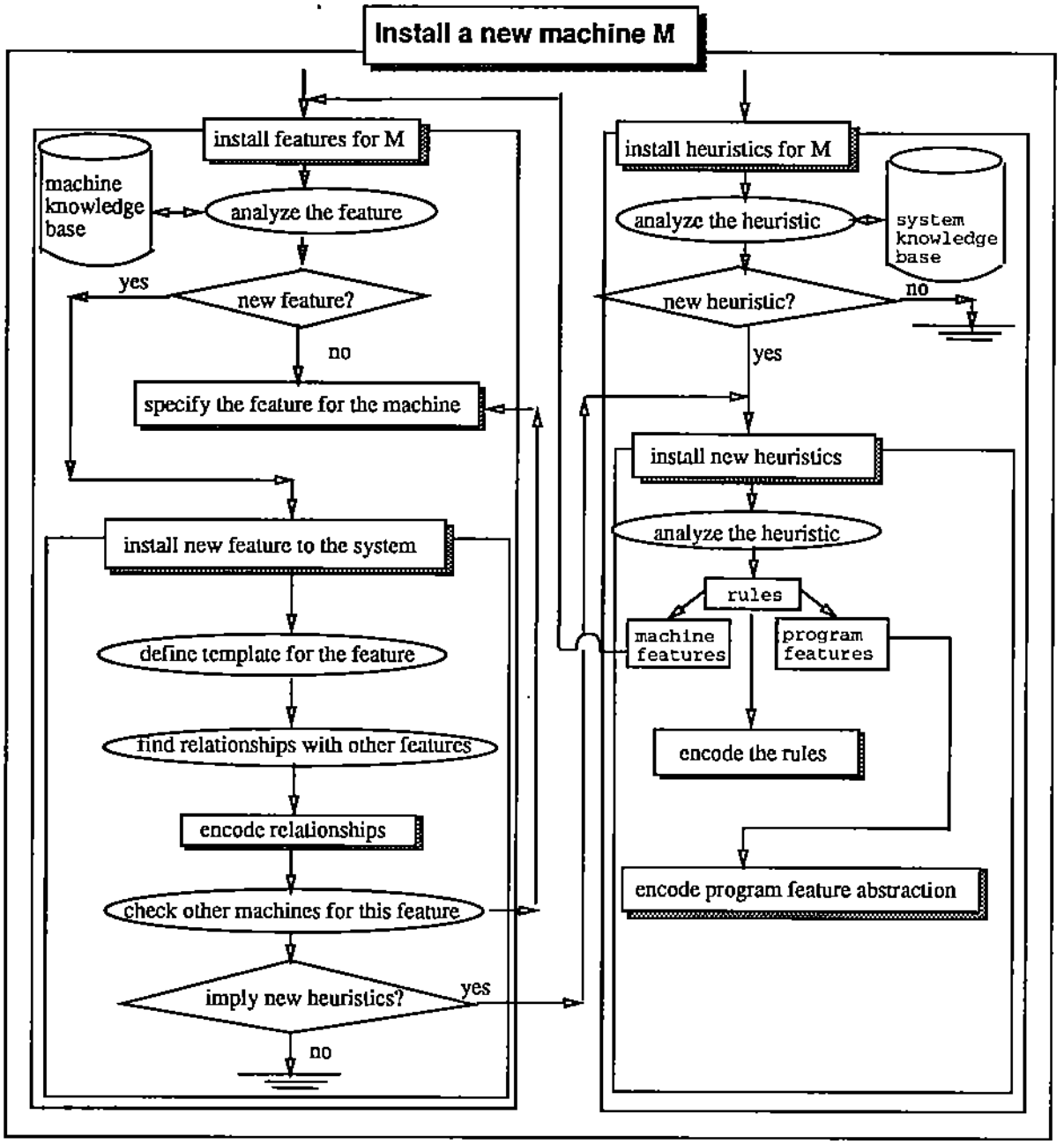


Figure 5. Machine feature specification and installation.

structure of the known machine features is adopted and the query session begins with high level specification of the machine, such as computational modes, distributed or shared memory, and gradually gets to the details of the machine features. The user does not need to know the structure of the machine features. Based on the pre-conditions and the organization of the features, the system is intelligent enough to ask only for the related features. There are also commands to allow the user to input features that the system did not ask or

does not know at all.

This interactive machine feature specification contains three kinds of activities:

1. The system asks the system programmer values of the related features of the machine.
2. The programmer specifies the features that the machine does not know to ask.
3. If the feature specified by the programmer is a new feature to the system, the user needs also to specify relationships between the feature and other features possibly with help from the machine knowledge manipulation system. The system will also help the user to specify the value of the new feature for machines already in the database of the system but whose corresponding feature values have not been specified yet.

To specify a feature that the system already knows (has feature object definition for the feature), the system prompts the user for the value of the feature with a menu dynamically constructed at run time. The user can also input the values of the features through the keyboard.

A new feature is added to the system by defining the template of the new feature (by defining the feature object) first. This process is illustrated by the example below.

After the new feature object is defined, the next step is to find relationships between the new object and other features. This process is normally non-trivial and the reasoning ability of the system may provide the user with help. After the user specifies some basic relations, the system tries to help by finding all features that are related to these features and providing this information to the user.

Finally, after the new feature is installed, an attempt is made to relate this new feature to the parallel computers already known to the system. In other words, the system will try to enhance its data base by figuring out whether other machines in the database have this feature, and if so what the values of the features are for these machines. The user will have to decide whether the new feature can be applied to particular target machines in the knowledge base of the system, but the system will be able to rule out machines that cannot have the feature based on the relationship specified by the user.

```
Entry selected: (change target machine).

Restructure the program for the current target machine ( ncube ) ?  (y/n): n

Select the target machine:          Page 1
     1   alliant
     2   ceder
     3   cyber205
     4   ncube7
     5   ncube
     6   none of the above

Please choose one of the above labels:
 (end with (return)) 6
Target machine name? 'crayXMP-4'.

Knowledge about the machine ncube is currently in my database.
Is it OK for me to remove it and replace it with knowledge of cray ?  (y/n): y

There are no feature values defined for machine 'crayXMP-4'!
Do you want to create it? (y/n): y

 Initialize machine database for ( crayXMP-4 ):

When I ask you the value of the feature that you select,
do you need explanation for the feature? (y/n): y

Please specify the value of the feature number_of_clusters:

 Explanation:  number_of_clusters is
    the number of clusters in the system.


Please specify the value of the feature number_of_processors:

 Explanation:  number_of_processors is
    the number of processors in a cluster.
Please give me a(n) integer :  4
(end with dot and (return)) .

Please specify the value of the feature computation_mode:

 Explanation:  computation_mode is
    the computational mode of the system.

Please select value of  computation_mode  from the following:          Page 1
     1   SIMD
     2   SISD
     3   MIMD
     4   MISD

Please choose one of the above labels: 3

Please specify the value of the feature method_of_scheduling:

 Explanation:  method_of_scheduling is
    the method of processor scheduling.
Please select value of  method_of_scheduling  from the following:          Page 1
     1   data_driven
     2   demand_driven
     3   data_flow
     4   control_flow
     5   self_scheduling
     6   other

Please choose one of the above labels:
 (end with (return)) 4

Please specify the value of the feature clock_cycle_time:

 Explanation:  clock_cycle_time is
    number of cycles per second (in MHz).
Please give me a(n) real_number :  117.5.

Please specify the value of the feature peak_mips:
```

Figure 6. An interactive session to specify features of a target machine.

After the machine features are installed, the heuristics of using the machine can be installed. For a given heuristics, one problem is to figure out what machine features are involved in the heuristic. The machine feature comparison and deduction provided by the

```
Before I start, I would like to know the degree of interaction you
want this process to have.

Do you need explanation for the process? (y/n): n
Want a short explanation in front of questions that I ask you? (y/n): n
Should I ask your confirmation after each entry is specified ? (y/n): n

Name of the new feature to define: vector_pipelines.

Specify format for the feature:  vector_pipelines :


 Formating feature: vector_pipelines  (pre-condition of the feature)
Please specify the pre-condition of the feature: vector_pipelines : has_vector_c
apability.

 Formating feature: vector_pipelines  (value type of the feature)
What type will the value of the feature vector_pipelines be?          Page 1
     1   integer
     2   real_number
     3   atom
     4   enum
     5   integer-range
     6   list-of-integer
     7   list-of-real_number
     8   list-of-atom
     9   list-of-enum
    10   list-of-term
    11   list-of-above
    12   bool

Please choose one of the above labels: (end with <return>) 6


What should I say if the user ask me what is feature: vector_pipelines :
> Vector pipelines, first gives the number of load and store pipelines, then giv
e number of floating point multiply and add pipelines.'.

 Formating feature: vector_pipelines  (verify conditon)
Please specify the verify condition to check that the feature value is correct.
Please list variables in front of the condition!: no.

Here is a list of attributes about the feature: vector_pipelines
==========================================================
Pre-condition:
    has_vector_capability
Type of the value:
    [list-of-integer,'[]']
Feature explanation:
    Vector pipelines, first gives the number of load and store pipelines, then g
ive number of floating point multiply and add pipelines.
Verify condition for the value:
    no

Is the above listing correct? (y/n): y

The format of the feature is initialized by the following predicate:
feature_attribute(vector_pipelines,[has_vector_capability,[list-of-integer,'[]']
,'Vector pipelines, first gives the number of load and store pipelines, then giv
e number of floating point multiply and add pipelines.',no])

Do you want to define format of other features? (y/n): y
Name of the new feature to define: machine_cycle_time.
Specify format for the feature:  machine_cycle_time :

 Formating feature: machine_cycle_time  (pre-condition of the feature)
Please specify the pre-condition of the feature: machine_cycle_time : no.

 Formating feature: machine_cycle_time  (value type of the feature)
What type will the value of the feature machine_cycle_time be?        Page 1
```

Figure 7. An interactive session to specify template of a feature.

machine feature manipulation system is very useful. To specify a heuristic, the user uses menus to specify the preconditions and the actions of the rules. The menu can lead users through the hierarchy of machine features from the top down and the knowledge base keeps a list of abstractions of the program features so that the user can relate machine features and the program features to the heuristics. The structure of the hierarchy and the

computational models help the user to analyze the relationship between the features and the heuristics. After the related features are picked, the system uses information in the feature objects to generate the preconditions and actions for the heuristics and translate them into rules. If the heuristic involves features that are not in the knowledge base, then the new features need to be installed.

As we noted above, getting a complete description of the machine features is difficult but is usually unnecessary. Using a partial list of the machine features to represent the machine adds two requirements in manipulation of the machine knowledge:

1. Knowing what features are relevant to the system and
2. Knowing the relationship of the new knowledge with the existing knowledge of the system.

One simple methodology to decide what machine features to represent is based on the heuristics of the system. When installing the heuristics, users represent the heuristics and methodologies of utilizing the machine parallelism as a function of the machine features and program properties. After the user finishes the specification of the heuristics, the system collects the list of machine features used and compares the list with the list of features that it already knows. In this way, new features can be discovered and installed systematically.

The procedures outlined above rely on two things: human interaction and the reasoning ability of the knowledge manipulation system to help the human expert sort out the complex relationship between the heuristics and the machine features. Systematic knowledge manipulation can significantly reduce the complexity of the machine knowledge abstraction and installation process. From our point of view, this is one of the basic requirements for all retargetable software systems.

5.3. Feature Deduction And Comparison

Heuristics are knowledge without a theoretical background. In order to generalize the heuristic to other parallel computers, heuristics need to be analyzed to determine the fundamental elements behind the heuristics. In this way, a new target machine can utilize the heuristic if the machine possesses all the features involved in the heuristic.

The ability to analyze relationships between machine features is needed because not all machine features are represented at the same level. Some features may be derived from other features and some may be preempted by other features. Similarly, when trying to abstract the features of a machine or distill effective features from a new heuristic, it is often necessary to compare features of the machines. The machine knowledge representation scheme we propose supports both operations plus other reasoning mechanisms. A knowledge base that features a simplified SQL database language plus inference mechanism is implemented to support the task of analyzing machine features and heuristics.

For example, it is possible to collect all machines that have feature F or find all features that can be derived by a set of features with the simple reasoning mechanism we described above.

Feature deduction is supported by the relation functions which are part of the object-oriented representations. Feature comparison of different machines is available by the implementation of a relational database. For example, we can find all common features or different sets of features of two machines with a relational database command: "find all common features of A and B" or "find all features A subst B."

5.4. Specializing System Knowledge

The advantage of having a general purpose machine knowledge manipulation system is that knowledge can be accumulated and shared among different architectures. The price paid is that when reasoning is performed, the performance suffers because of the added tests for checking the applicability of the heuristics at run time. We use a methodology called *knowledge caching* to improve the performance of the inference system. The method works as follows: At the compiler construction, maintaining and enhancing phases, the system encodes the knowledge with the multiple target paradigm. At the distribution phase, specialized single target parallel compilers are built by validating or invalidating the rules in the knowledge base. This is done by pre-evaluating the rules based on the features of the target machine. All conditions that are implied by the static features are eliminated from the rules, and the resulting simplified rules are "cached" in the specialized compiler. Also, rules that are disqualified by the static features are deleted from the knowledge base of the new compiler. This approach has the advantage of code re-use and knowledge

sharing but does not suffer loss in system efficiency.

Our knowledge representation scheme also provides a runtime mechanism to optimize feature access. The same procedure can be used at run time to further eliminate redundant conditions or invalid rules based on the dynamic features. With the knowledge cache mechanism, the most recently accessed or generated objects are kept in the system memory so that subsequent access will be much cheaper. The resulting parallel compiler is specialized for the target machine with the features of the machine implicitly built into the rule base of the system. The rules for the program transformation can be linked with the machine features by attributes of the features. Therefore, after the features of the machine are specified, a personalized knowledge base can also be built for the particular architecture and thus improve both space utilization and execution efficiency.

6. Other Applications Of The Representation Scheme

The machine knowledge manipulation system we described above can be applied to many other software systems that need details of the target machines. Two such examples are given here.

6.1. Distributed Computing Environments

A distributed computing environment is a system that contains a set of loosely connected computers. Although not every distributed system needs detailed machine information of the computers in the system, some applications do require the system to have low level knowledge of its members. An interesting example of this is as follows: in a network that contains a wide spectrum of architectures (for example, a network of workstations, graphic workstations, main frames, and supercomputers), and applications. Suppose the goal of the operating system is to assign a task to a machine that is most appropriate (the objective may be adaptive) for the application; there is no way that the system can make smart decisions without a clear understanding of the capabilities of each machine in the system and some information about the tasks. Our machine knowledge manipulation system allows the system to possess and manipulate knowledge of many computer systems and support the match of machines and applications. Thus, it is possible to build a smart task scheduler for distributed, heterogeneous, computing environments based on the

machine knowledge manipulation system.

6.2. Flexible Parallel Computing System Simulation Systems

A product design cycle consists of requirement, specification, design, prototyping, testing, and modification. The design of new computer architectures usually goes through this cycle many times before the final product emerges. At each iteration of the design cycle, requirement, specification, and design of the machine may be changed because of technical difficulties, marketing considerations, and other unexpected problems. Any changes can affect the subsequent phases and complicate the designing problem. Building a hardware prototype is very time-consuming and expensive. In contrast, an electronic prototype can shorten the design-testing cycles and decrease the product-developing lead time. The machine knowledge manipulation system we discuss here provides a foundation for building software simulation systems for parallel computers. When coupled with the performance evaluation system discussed in [Wang90b], a very flexible general purpose parallel architecture simulation system can be built. Under this model, revising the architecture design is relatively easy since the machine knowledge manipulation system provides easy modification of the machine feature entries. Furthermore, when integrated with the program transformation system, the resulting system has two significant advantages:

- The domain that the architecture is targeting can be used to test and evaluate the design before a hardware prototyping system is built. Problems in design can be discovered at the early phase of the developing cycle.
- The experience accumulated from other parallel computers can be applied to the new machine. Thus, a great deal of heuristics for using the machine exist even before the machine is actually built.

7. Conclusions

In this paper, we have presented a machine knowledge representation scheme for parallel computers that supports reasoning. Intelligent reasoning is possible because decisions can be made from analysis of machine features. The machine knowledge manipulation system forms the basis of a parallel programming environment we are implementing which can restructure the program structure intelligently.

The knowledge representation scheme we propose has the following significant features:

1. Knowledge sharing. Separating the machine knowledge from the system heuristics allows heuristics to be shared among different parallel computers.
2. Object-oriented presentation. The object-oriented representation scheme allows modular and elegant knowledge representation and ease of manipulation. With abstraction and classification operations, the parallel machines can be abstracted into different levels of computational models.
3. Tolerance. Incomplete machine specification as well as incomplete system knowledge is allowed in this representation scheme. When a feature is not present, the system simply assumes that the target machine does not have it. Even though the performance may suffer, the more detailed knowledge about a machine the system contains, the better the approximation of the computational model is to the real machine. However, the system works even with incomplete knowledge of the machine so that knowledge about a new machine can be incorporated incrementally.
4. Flexibility in the organization of the knowledge. The machine knowledge can be organized on different criteria. For example, at the higher layer, the machine knowledge can be classified on the levels of parallelism (the multiprogramming level, multiprocess level, inter instruction level, and instruction level). At the lower layer, the machine knowledge can be grouped on the physical organization of the system (processing units, memory hierarchy, communication network/bus, clusters of processors, and control unit).
5. Test beds for complex systems. The proposed representation scheme allows easy construction of test beds for complex software or hardware systems. Features and relation functions can be added or removed to test the consequence of the action. This property of the system can also be used to evaluate or study the effectiveness of the features, relation functions or heuristics.

Although we are targeting the methodologies to the parallel compilers, the methodologies can be applied to any software systems that require detailed knowledge of the underlying architectures, such as parallel simulation systems, distributed operating

systems, and runtime environments.

8. REFERENCES

[BWJALG90] F. Bodin, D. Windheiser, W. Jalby, D. Atapattu, M. Lee, and D Gannon, "Performance Evaluation and Prediction for Parallel Algorithms on the BBN GP1000," *Proceedings of the 1990 International Conference on Supercomputing*, 1990, 401-413.

[Fisher84] J. Fisher, "The VLIW machine: A multiprocessor for Compiling Scientific Code," *IEEE Computer*, July, 1984, 45-53.

[Flynn66] M. J. Flynn, "Very High Speed Computing Systems," *Proc. IEEE, vol. 54*, 1966, 1901-1909.

[Händler77] W. Händler, "The Impact of Classification Schemes on Computer Architecture," *Proc. 1977 Intl. Conf. on Parallel Processing*, 1977, 7-15.

[Hwang84] K. Hwang, "Supercomputers - Design and Applications," 1984.

[HwBr84] K. Hwang and F. Briggs, "Computer Architecture and Parallel Processing," McGraw-Hill, 1984.

[HwDe89] K. Hwang and D. DeGroot (editors), "Parallel Processing For Supercomputers & Artificial Intelligence," McGraw-Hill, 1989.

[KGSF84] A. Kapauan, D. Gannon, L. Snyder and T. Field, "The Pringle Parallel Computer," *Proc. of the 11th International Symposium on Computer Architecture, IEEE*, 1984, 12-20.

[KWGCS84] A. Kapauan, K. Wang, D. Gannon, J. Cuny and L. Snyder, "The Pringle: An Experimental System for Parallel Algorithm Design and Testing," *Proc. of the 1984 International Conference on Parallel Processing*, Keller, editor, 1-8.

[KDLS86] "Parallel Supercomputing Today and the Cedar Approach," in *Science* Vol 231, Feb. 1986, 967-974.

[Minsky75] M. Minsky, "A Framework for Representing Knowledge," in P. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, 1975, 211- 277.

[Nau83] D. Nau, "Expert Computer Systems," *IEEE Computer*, Feb, 1983, 63-85.

[Padua79] D. Padua, "Multiprocessors: Discussion of Some Theoretical and Practical Problems," Ph.D. Thesis, University of Illinois, Urbana-Champaign, Nov. 1979.

[PfBrNo85] G. Pfister, W. Brantley, D. George, S. Harvey, W. Kleinfelder, K. McAuliffe, E. Melton, V.A. Norton, J. Weiss, "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture,", *Proc. of the 1985 International Conference on Parallel Processing,* 1985, 764-771.

[PfNo85] G. Pfister, V.A. Norton, "Hot Spot Contention and Combining in Multistage Interconnection Networks," *Proc. of the 1985 International Conference on Parallel Processing,* 1985, 790-797.

[Schw80] J. Schwartz, "Ultracomputer," *ACM Transactions on Programming Languages and Systems,* Vol. 2, No. 4, October, 1980, 484-521.

[Smith82] A. Smith, "Cache Memories," *Computing Surveys,* Vol. 14, No. 3, September, 1982, 473-530.

[Veid85] A. Veidenbaum, "Compiler Optimizations and Architecture Design Issues For Multiprocessors," Ph.D. Thesis, Dept. of Computer Science, University of Illinois, Urbana-Champaign, CSRD Rpt. No. 520, 1985.

[Wang85] K. Wang, "An Experiment in Parallel Programming Environment: The Expert Systems Approach," in K. S. Fu, editor, *Some Prototype Examples for Expert Systems,* TR-EE 85-1, Purdue University, Mar. 1985, 591-624.

[WaGa89] K. Wang and D. Gannon, "Applying AI Techniques to Program Optimizations For Parallel Computers," in K. Hwang and D. DeGroot, editors, *Parallel Processing for Supercomputers and Artificial Intelligence,* McGraw-Hill, 1989, 441-485.

[Wang90a] K. Wang, "A Performance Prediction Model For Parallel Compilers," Tech. Report, CSD-TR-1041, CER-90-43, Department of Computer Science, Purdue University, Nov. 1990.

[Wang90b] K. Wang, "Array Reshaping - A Mechanism for Optimizing Array Storage On Parallel Architectures," Tech. Report, CSD-TR-1042, CER-90-44, Department of Computer Science, Purdue University, Nov. 1990.

[Wang90c] K. Wang, "Managing Data Synchronization Automatically For Distributed-Memory Architectures," Tech. Report, CSD-TR-1043, CER-90-45, Department of Computer Science, Purdue University, Nov. 1990.

[Wang90d] K. Wang, "A framework For Intelligent Parallel Compilers," Tech. Report, CSD-TR-1044, Department of Computer Science, Purdue University, Nov. 1990.

[Wang90e] K. Wang, "Heuristic Guided Pre-Optimized Algorithm Substitution For Parallel Computers," Tech. Report, CSD-TR-1055, CER-90-50, Department of Computer Science, Purdue University, Dec. 1990.

[Yew88] P-C Yew, "Architecture of the Cedar Parallel Supercomputer," in *Parallel Systems and Computation* Paul and G.S. Almasi, editors, Elsevier Science Publishers, 1988, p137-148.