Computer-Aided Design, Vol. 29, No. 1, pp. 37–51, 1997
Published by Elsevier Science Ltd
Printed in Great Britain.
0010-4485/97/$17.00+0.00

PII: S0010-4485(96)00047-4

Towards multiprocessor feature recognition

William C Regli, Satyandra K Gupta* and Dana S Nau†

The availability of low-cost computational power is enabling the development of increasingly sophisticated CAD software. Automation of design and manufacturing activities poses many difficult computational problems—significant among them is how to develop interactive systems that enable designers to explore and experiment with alternative ideas. As more downstream manufacturing activities are considered during the design phase, computational costs become problematic. Creating working software-based solutions requires a sophisticated allocation of computational resources in order to perform realistic design analyses and generate feedback.

This paper presents our initial efforts to employ multiprocessor algorithms to recognize machining features from solid models of parts with large numbers of features and many geometric and topological entities. Our goal is to outline how improvements in computation time can be obtained by migrating existing software tools to multiprocessor architectures. An implementation of our approach is discussed. Published by Elsevier Science Ltd

Keywords: distributed computing, feature-based modelling, feature recognition, multiprocessor solid modelling

The availability of low-cost computational power is enabling the development of increasingly sophisticated CAD software. Software tools designed to reduce time-consuming build–test–redesign iterations are becoming essential for increasing engineering quality and productivity. Examples include tools for finite element analysis, mechanism analysis, simulation, and rapid prototyping. Such tools have become crucial components for research in collaborative engineering and engineering design.

Automation of the design process and construction of such tools, however, pose many difficult computational problems. To realize the advantages of collaborative engineering, more downstream engineering activities are considered during the design phase. As design is an interactive process, developing techniques to manage computational costs better is critical in systems that enable designers to explore and experiment with alternative ideas during the design stage. Achieving reasonable levels of interactivity between design and downstream activities (such as analysis, process planning, and simulation) requires an increasingly sophisticated allocation of computational resources in order to perform design analyses and generate feedback.

It is becoming increasingly evident that one necessary component of an automated design analysis tool is a subsystem for recognizing manufacturing features directly from a CAD or solid model. This problem has been the focus of extensive research over the last decade. Feature recognition is used for a variety of applications, including the generation of process plans[53,14,15], translation between design and manufacturing features, and generation of redesign suggestions[5,6]. What has also become evident is that feature recognition, for realistic classes of parts with multiple and interacting feature interpretations, requires extensive geometric reasoning and is computationally expensive. Hence, generating the features from a part is a computational bottleneck within an integrated design system.

In this paper we present our initial efforts toward developing a methodology for recognizing a class of machining features using multiprocessor algorithms on a distributed system. Feature recognition has been approached using a variety of techniques, some of which are easier to parallelize than others. In previous work[27], we described serial *trace-based* algorithms for finding feature instances from solid model data. A trace represents the information in the solid model of the part produced by an instance of a feature.

The techniques presented in this paper demonstrate the feasibility of migrating existing serial feature recognition systems to take advantage of multiprocessor computing technologies. We report results indicating that trace-based feature recognition methodologies are particularly well suited for parallelization. The basic steps in our approach are:

(1) *Task initialization.* Initialization is performed at four levels: (1) the types of features to be recognized; (2) the types of trace information used to construct the feature instances; (3) the decomposition of the geometry and topology of the traces; and (4) the simplification of the part geometry to reduce the costs to solid modelling operations.
(2) *Task distribution.* The problem is divided using the task decomposition—isolating independent portions of the recognition problem and identifying a suitable computational resource for solving it.
(3) *Synthesis of results.* The results obtained by each separate processor are combined into a global solution. This solution set can then be passed on to

National Institute of Standards and Technology, Manufacturing Systems Integration Division, Building 220, Room A-127, Gaithersburg, MD 20899, USA
* Rapid Manufacturing Laboratory, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA
† Department of Computer Science, Institute for Advanced Computer Studies and Institute for Systems Research, University of Maryland, College Park, MD 20742, USA
Paper received: 29 May 1995. Revised: 3 May 1996

Chuang and Henderson[3] explore graph-based pattern matching techniques to classify feature patterns based on geometric and topological information from the part. Efforts at Carnegie Mellon University[23,28] have employed graph grammars for finding features in models of injection moulded parts. Recently, Corney and Clark[4] have employed graph-based algorithms to find general feature classes from $2\frac{1}{2}$-dimensional parts.

Gadh and Prinz[9] were the first to describe techniques for combating the combinatorial costs of handling complex industrial parts (i.e. those with thousands of topological entities). They point out that, in such cases, traditional knowledge-based, decomposition, and pattern-matching techniques are computationally impractical because the fundamental algorithms (i.e. frame-based reasoning or subgraph pattern matching) are inherently exponential. Gadh and Prinz's method is to abstract an approximation of the geometric and topological information in a solid model and find shape features in the approximation. Their approach employs a differential depth filter to reduce the number of topological entities. A second pass maps the topological entities onto structures called 'loops'. In their work, features are defined using the higher-level loops as opposed to being defined as patterns in the boundary representation's geometry and topology. This approach significantly reduces the number of entities that need to be searched to build feature instances. While this kind of approach holds much promise for addressing combinatorial problems, it does not address how to extend the techniques to better handle interacting features and non-linear (non-faceted) solid models.

Fields and Anderson[9] present an approach to feature recognition that overcomes some of the representation and efficiency problems common in previous work. Unlike pattern-based or decomposition-based recognition methodologies, they categorize sets of faces on the surface of the part into classes of general machining features: protrusions, depressions, and passages. The shapes within each class, while sharing many operational similarities, may vary in geometry and topology. For each of their feature classes, they present a linear-time algorithm for identifying features.

Trace-based feature recognition

Most relevant to the work in this paper are the recent trace-based feature recognition methodologies. Fundamentally, a trace-based approach to feature recognition attempts to reconstruct feature instances from the information that they contribute to the final geometric model of the product.

The work of Marefat and Kashyap[2] presented an early trace-based technique. They expanded on the work of Joshi and Chang[18], augmenting it with hypothesis testing techniques. In Marefat and Kashyap's method, information from the solid model is used to generate hypotheses about the existence of features. These hypotheses are tested to see if they give rise to valid feature instances.

Vandenbrande and Requicha[33] were the first to formalize trace-based (or hint-based) techniques for constructing features from information in a solid model. In the work of Vandenbrande, the traces are used to fill 'feature frames' in a frame-based reasoning system. After filling frames with the trace information present in the part, the system classifies the partial frames and attempts to complete information for producing promising frames using a variety of geometric reasoning and computational geometry techniques. This work has recently been enhanced and extended by Han and Requicha[14,15].

Regli *et al.*[27] present an approach for guaranteeing completeness of a recognition algorithm, i.e. it describes how one can define a class of features and verify that a particular approach was capable of producing all features in that class. They present feature recognition as an algorithmic problem in which traces are found by traversing the geometry and topology of the part and then used to construct feature instances. They formally describe the behaviour of their algorithm and calculate a general measure of its complexity. This approach has been employed for automated design analysis[13] and automated redesign[5,6].

Many aspects of the feature recognition problem are still open and active areas of research. Among these are: recognizing and representing interacting features[33], incremental recognition of features during feature-based design[20,14,15], modelling alternative feature interpretations and completeness[21,27], and reasoning about the manufacturability of features[13].

APPROACH TO FEATURE RECOGNITION

In this section we outline a basic trace-based feature

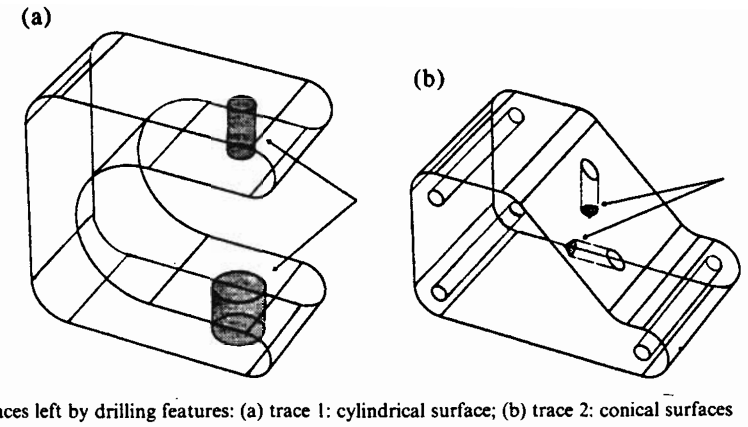

Figure 1 An example of traces left by drilling features: (a) trace 1: cylindrical surface; (b) trace 2: conical surfaces

of manufacturing it. A *feature-based model* is a set of feature instances that models a single, unique interpretation of the part. The feature recognition problem can be defined as follows: given a collection of machining features $\mathcal{M} = \{M_1, M_2, \ldots, M_j\}$, a part P, and a piece of stock S, find the set $\mathcal{F}$ of instances of feature types from $\mathcal{M}$ recognized from P and S. The feature set $\mathcal{F}$ is a finite set of features the set being composed of the union of the alternative feature-based models for the part[27].

Trace-based recognition of features

A *trace* represents the information in the solid model of the part produced by an instance of a feature. The basic components of a trace-based feature recognizer are the following:

(1) A finite set $\mathcal{M}$ of feature types. In the context of this paper, $\mathcal{M}$ is made up of the simple feature domain defined earlier.
(2) Each feature type M in $\mathcal{M}$ has associated with it a finite set of trace types $t_{M_1}, t_{M_2}, \ldots, t_{M_k}$. Trace types are developed below in this subsection.
(3) For each trace type t_{M_i}, there is a procedure $\mathcal{P}_{t_{M_i}}()$ such that $\mathcal{P}_{t_{M_i}}()$ constructs, from instances of the traces and the solid model of the part and stock material, instances of features of type M capable of producing the trace t_{M_i}.

Note that a feature type M might have several different types of traces associated with it; also a feature instance f might leave several different traces on the model of the part. Conversely, a trace type might produce one or more feature instances (e.g. the cylindrical surface of a through hole can be considered as two different drilling features, one in each direction along the axis of the cylinder). In this work we are focusing on feature traces that can be identified on the boundary of the part (i.e. traces left in $(f) \cap b(P)$ where $b(P)$ is the boundary of P).

An outline for a generic algorithm for trace-based recognition of features can be presented as follows:

(1) Input a collection of feature types $\mathcal{M}$, a solid model for the part P, and a solid model for the initial stock material S.
(2) From P and S, identify the set $\mathcal{T}$ of all potential traces present.

There are several ways in which the traces can be identified (for example, previous research has included hypothesis testing[21] and frame-based reasoning approaches[33]). The traces in this paper can be identified by examining the topology in the boundary representation of Δ.

(3) For each potential trace t in $\mathcal{T}$ do: If t matches a t_{M_i}, call the procedure $\mathcal{P}(t_{M_i})$ and construct (if possible) feature instances, $f_1, f_2, \ldots, f_n$, of type M. Add these to the set $\mathcal{F}$ of all feature instances.

Detailed presentation of trace-to-feature algorithms is beyond the scope of this paper. Interested readers are referred to related work which contains detailed examples of such algorithms[12,14,27,33] for all of the above traces.

Example trace types

For illustrative purposes, the task of recognizing basic drilling and end-milling features can be accomplished using the following traces types:

1. *Drilling features.*

Trace 1. Any convex cylindrical surface s_c in the delta volume created by the side surface of a drill during a drilling operation.
Rationale: This trace type is used to build instances of drilling features when a portion of their side surface remains on the boundary of the delta volume. An example of this trace is illustrated in *Figure 1a.*

Trace 2. A convex conical surface s_f in the delta volume created by the side surface of conical tip of a drilling tool.
Rationale: This trace type is used to build an instance of a drilling feature when only a portion of its ending tip surface remains on the boundary of the delta volume. An example of this trace is illustrated in *Figure 1b.*

2. *End-milling features.*

Trace 1. A planar surface s_p in the delta volume created by the cutting tip of an end-mill. This trace is used to build instances of end-milling features when only a portion of their bottom surfaces are present on the boundary of the delta volume.
Rationale: This trace type is used to determine the profile of end-milling features. Given an edge $e_1 = \langle v_1, v_2 \rangle$ of the planar surface s_p, orientations and locations for potential milling features can be obtained from other edges* $e_2 = \langle v_3, v_4 \rangle$ in the delta volume for which the vertices v_1, v_2, v_3, v_4 are coplanar. An example of end-milling trace 1 is given in *Figure 3a.*

The following two traces are used to build instances of end-milling features when only a portion of their side surfaces are present on the boundary of the delta volume. In these cases, the end-milling features may extend completely through the stock material. Examples of such features include through pockets and profiles.

Trace 2. A cylindrical surface in the delta volume as a surface created by the side cutting surface of an end-mill. An example of end-milling trace 2 is given in *Figure 3b.*
Rationale: The profile of a milling feature might comprise curved edges, for example, the corner radii created when a round tool machines a convex corner. This trace type uses these curved surfaces to determine the orientation of potential through features.

Trace 3. A planar surface in the delta volume, considered as a face created by the side cutting surface of an end-mill during the same machining operation. *Figure 3c* shows an example of milling trace 2.
Rationale: For some instances of through milling features, all that may remain are walls. This trace type begins with a single planar wall and, by considering other planar surfaces in the delta volume, obtains orientations for potential through milling features from the normal vectors; i.e. two non-parallel planar surfaces can be used to determine the orientation of

* Note that in the solid model of the delta volume, the edges e_1 and e_2 might be non-linear curves, e.g. they could be elliptical.

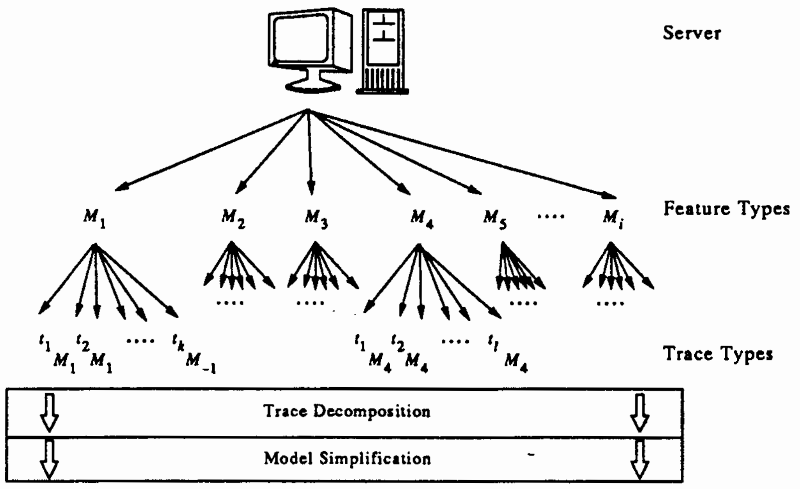

Figure 6 Divide-and-conquer parallelization based on feature types and trace types

feature instance f in different ways using different traces. There are two possible approaches to handling this redundancy. One method is to delete duplicate features while building the final feature set $\mathscr{F}$. A second approach, and the one that we will employ, is to identify the traces capable of producing equivalent feature instances and handle them together on the same processor, removing duplicates as they are found. This introduces another level of parallelization by dividing the set of traces found into independent subsets. In this way redundancies are addressed at the level at which they occur, thus simplifying the task of building the final feature set $\mathscr{F}$.

Parallelizing feature recognition produces other, less obvious benefits. In particular, a large portion of the costs in a feature recognition system are due to the complexity of geometric computations and geometric reasoning. When isolating independent problem subtasks, one can make geometric and topological simplifications that identify the information in the original part needed to build and verify the feature instances. In this way, many of the subproblems may require only a fraction of the information present in the solid models of the original part and stock.

Distributed methodology

For the example domain of the previous section, our approach is to have a central computing resource act as a server to set up the problem and transmit subtasks to client machines distributed on the network, as illustrated in *Figure 5*. Each of the individual client processors is given an independent portion of the particular global feature recognition problem.

A distributed algorithm

Recalling the serial trace-based algorithm of subsection 'Traced-based recognition of features', we present an outline for a multiprocessor trace-based feature recognition algorithm. There are two main components to this system: a parent algorithm and a child algorithm. Note that these algorithms partition the problem at several levels, as shown in *Figure 6*. The parent algorithm is as follows:

Parent algorithm

```
(1) Input a collection of feature types $\mathscr{M}$, a solid model for the part $P$, and a solid model for the initial stock material $S$. Initialize the set $\mathscr{F}$ of recognized features, $\mathscr{F} = \emptyset$.
(2) For each feature type $M$ in $\mathscr{M}$, fork a new process* on a free resource (i.e. a CPU or machine) and do
    (a) For each trace type $t_{M_i}$ for feature type $M$ do
        (i) Find the set $T_{t_{M_i}}$ of traces of type $t_{M_i}$ present in the CAD model of the part $P$.
        (ii) Use the set $T_{t_{M_i}}$ to divide the problem into independent subtasks, $\tau_1, \tau_2, \ldots, \tau_j$.
        (iii) for each $\tau_i$ do
            (A) *Decompose* the part $P$ using the $\tau_i$—result $P'$. Trace decomposition is discussed in more detail later.
            (B) Fork a new process on a free resource to call the child recognition algorithm on $P'$.
        (iv) Let $F_{t_{M_i}}$ be the set of features returned by the child.
(3) $\mathscr{F} = \mathscr{F} \cup_{\forall t_{M_i}} F_{t_{M_i}}$.
(4) Return $\mathscr{F}$.
```

The child algorithm constructs, when possible, feature instances from the traces. It is executed over the available processors as follows:

Child algorithm

```
(1) Input a feature type $M$, a trace type $t_{M_i}$, a set of instances of $T_{t_{M_i}}$ of trace $t_{M_i}$, and solid models for the part $P'$, and the stock material $S$.
(2) *Simplify* the solid model of the part $P'$—result $P''$. Model simplification is discussed later.
(3) Call $\mathscr{P}(t_{M_i})$ to build feature set $F_{t_{M_i}}$.
```

*To fork a process is to start a separate task running within a multitasking operating system. In current practice on multiprocessor systems this can also be accomplished by starting a thread[24,31]. A thread can be thought of as an independent subprocess that can be executed on its own separate CPU, if one is available.

possibly by the same operation. End-milling features with multiple traces (e.g. a bottom surface divided into multiple subfaces) can be isolated and identified. For the part in *Figure 7a*, this results in the grouping of traces shown in *Figure 7d*.

(3) *Decomposition for end-milling trace 2.* Group cylindrical surfaces with equivalent axes.
Rationale: This groups all potential corner radii and curved walls for end-milling features with the same machining orientation. For the part in *Figure 7a*, this results in the grouping of traces shown in *Figure 7e*.

(4) *Decomposition for end-milling trace 3.* Group planar surfaces with normals perpendicular to a common vector; i.e. for each grouping there is a vector **v** such that, for all surfaces s_i and s_j in the grouping, normal $(s_i) \cdot \mathbf{v} = \text{normal}(s_j) \cdot \mathbf{v} = 0$.
Rationale: This groups traces for end-milled features based on machining orientation; hence through features that can be machined in the same orientation are placed in the same group. For the part in *Figure 7a*, this results in the grouping of traces shown in *Figure 7f*.

The above decomposition groups those traces from the part which might produce equivalent feature instances. In this way, redundancies can be eliminated at the subprocess level and later recombination of results can be facilitated.

Part simplification

The objective of part simplification is to reduce the amount of data that must be considered by each processor to a minimum amount sufficient to construct feature instances from the traces it has been given. The goal is to reduce the cost of operations during feature recognition. For example, one can reduce the number of geometric and topological entities while still retaining the information required to construct feature instances from the particular trace. In this way, geometry which does not affect the feature trace under consideration can be eliminated.

This section describes Step 2 of the *Child Algorithm*, in which the solid models of the part and stock are simplified based on the trace information and feature types. In each case, the geometry and topology of the model for the part P is modified to P' as follows:

(1) *Simplification based on drilling trace 1.* Given a cylindrical surface c in the delta volume of radius r, P' contains all the portions of P that lie within r of the axis of c.
Rationale: This simplification retains enough information to check for interference between the cutting tool and the final part. To check for interference between the workpiece and the machine tool, this radius may be enlarged depending on the size of the tool assemblies available in the particular set of manufacturing resources.

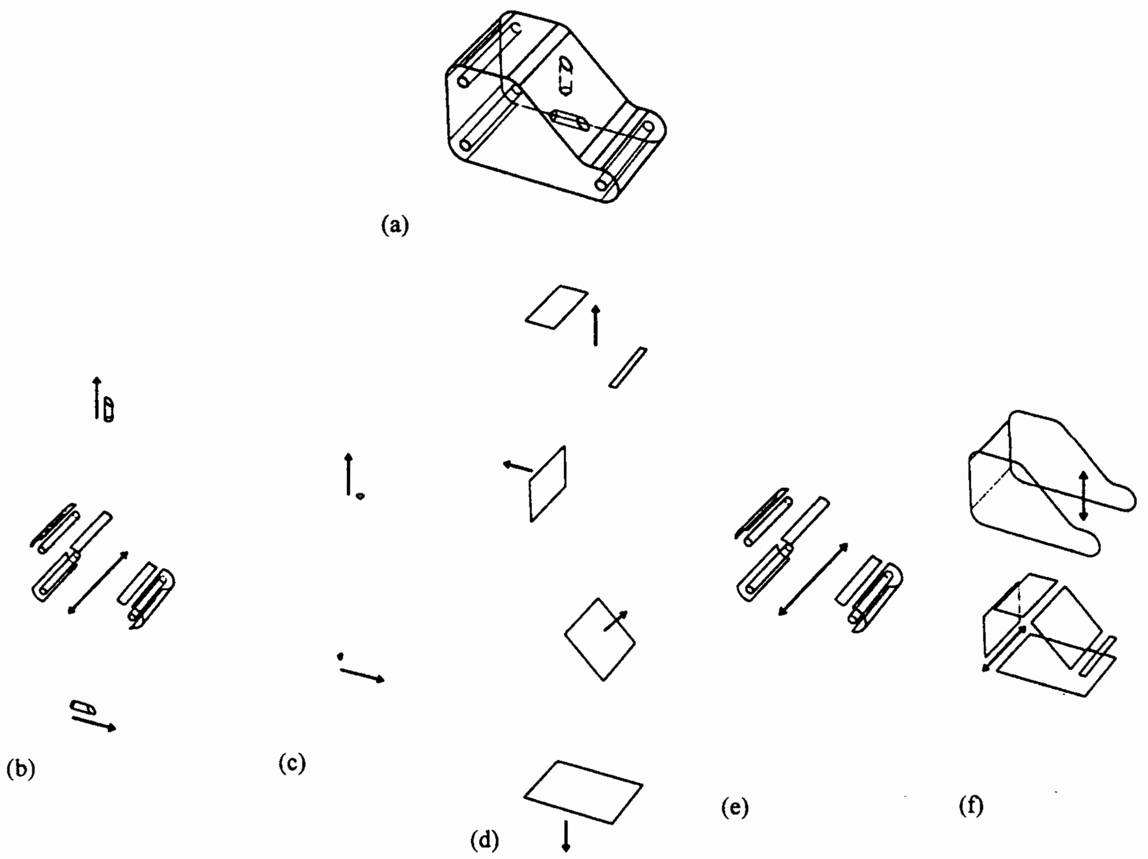

Figure 7 An example part and its trace decomposition. The arrows in each figure denote the orientation vector **v**, for the features that might have created these traces: (a) an example part; (b) drilling trace 1; (c) drilling trace 2; (d) milling trace 1; (e) milling trace 2; (f) milling trace 3

from the entire part, and the processing required in this case might be costly.

Figure 9 shows an example part and four illustrations of part simplification for end-milling trace 1. In the figure, the planar faces are being considered as traces indicating potential bottom surfaces of several end-milled features; vector **v** denotes the orientation of the potential feature. In each case, the trace information is used to eliminate the portion of the part lying below the trace (in the direction $-\mathbf{v}$)—information that does not get considered when building a feature instance in direction **v**. Note that in making this rudimentary simplification the number of geometric and topological entities to be considered is greatly reduced.

Potential for computational improvement

We can expect the speedup to be no more than a factor of K, where K is the number of processors available. In reality, the task decompositon to set up parallelization incurs some added cost, as does the recombination of results at the end. These additions are negligible, however, when compared with the costs incurred to perform the recognition process on the subproblems.

Within a trace-based methodology, the overall complexity of recognition depends on two factors: the difficulty in generating the set $\mathscr{T}$ of potential traces, and the complexity of the methods for generating feature instances from traces.

A rough upper bound on the size of $\mathscr{T}$ can be computed from the model of the part and the types of traces by counting the number of geometric and topological entities. The complexity of the feature construction routines is more difficult to assess and is where the majority of the computational costs occur. Much of this cost is due to geometric queries and reasoning used to find the parameters of feature instances. While there is no authoritative reference on the general complexity of solid modelling operations such as Booleans, sweeps, and the like, indications are that these operations account for the majority of the computational cost during feature recognition[33]. The complexity of Boolean operations appears to lie between $O(n^2)$ and $O(n^4)$ or $O(n^5)$ time, depending on the particular configuration of geometric entities and many implementation-specific details.

The fact that these basic solid modelling routines are at least quadratic in the size of the model implies that small reductions in the number of entities in the model translate into large reductions in computational cost.

In the next section, we provide rough estimates of both the speedup factor and the reduction in the number of geometric and topological entities achieved by our approach.

IMPLEMENTATION AND RESULTS

A proof-of-concept implementation of this distributed feature recognition methodology, dubbed *F-Rex*, has been done in c++ using version 3.0.1 of the AT&T c++ compiler from SUN Microsystems running on networked SUN SPARC Stations. F-Rex employs version 1.5.1 of Spatial Technologies' ACIS© solid modelling system and version 3.14 of the NIH c++ Class Library developed at the National Institutes of Health. Additional tools include Ithaca Software's HOOPS© Graphics System and the Tcl/Tk embeddable command language and user interface toolkit from the University of California at Berkeley.

F-Rex is the feature recognition subsystem for IMACS, an interactive manufacturability analysis tool under development at the University of Maryland's Institute for Systems Research. One of the fundamental goals of IMACS is to provide interactive feedback and redesign suggestions to the user. Multiprocessor algorithms have provided IMACS with a means of handling computational bottlenecks.

F-Rex runs on a cluster of SUN workstations; processes communicate over the Internet using UNIX-based and TCP/IP-protocol-based network software utilities and shared disk storage. The geometric computations required for task initialization are implemented with direct c++ calls to the ACIS kernel; distributed processes are invoked using UNIX remote shell commands; and the resulting feature set is generated by examining the features produced by each processor and eliminating redundancies.

The data for the examples below have been collected using six processors, one SPARC Station model 10, one model 2, and 4 IPX models. In this version of the implementation, when the number of tasks is greater than 6, the tasks are distributed evenly over the available processors.

These timing results represent the elapsed clock and CPU times and are not absolute measures of the intrinsic difficulty of the feature recognition problem—this example domain is not directly comparable to those of other feature recognition efforts. Further, there are hidden costs in the implementation not directly related to the recognition of feature templates (such as feature accessibility analysis) and these algorithms and their implementation can certainly be improved. The results are intended to provide a rough indication of the time-lag experienced by the user of the system. More significant than any precise calculation of elapsed time is the speedup factor between the serial and parallelized algorithms. Measurements of elapsed CPU time are summarized in *Table 1.*

Example 1

The example part in *Figure 10a*, taken from Reference 33, contains 21 part faces. Vandenbrande and Requicha[33] report identifying 7 features (3 slots, 3 open pockets, and a step) in 2.5 min on a SUN 4/360. The OOFF system[33] handles a wide variety of machining features and process planning constraints; hence it is not directly comparable

Table 1 Estimated elapsed CPU times for each example

Example	Serial (s)	Distributed set-up (s)	Recognition (s)
1	54	0.96	9.8
2	116	3.5	43.1
3	127	1.2	6.4
4	>1800	75	700
5	>1800	19.5	701.5

many cylindrical curved surfaces and that few of the feature instances interact; also note that the decomposition techniques handle all of the drilling features on the same processor.

Example 5

The example part in *Figure 9* is a shuttle intended to move along a guideway, with many of the feature instances added to reduce weight. The solid model of this part contains 281 faces. In serial, F-Rex takes over 1 h to find more than 100 feature instances. When running distributedly, F-Rex took 2 min to set up the task decomposition and approximately 32 min to find the features—a speedup of approximately 2× (200%). In this case, simplification resulted in a 43% reduction in the number of geometric and topological entities that had to be considered. Note that in this example nearly every feature interacts with every other feature and that calculation of accessibility volumes for feature instances is rather complex.

Discussion of results

These preliminary results confirm that performance gains can be made through effective parallelization of algorithms. One issue is how to systematically identify *a priori* how to produce effective decompositions (i.e. some classes of features might prove more amenable to this approach). In general, however, it is difficult to assess what a typical decomposition and its speedup factor will be.

We believe that the considerable variation in our experiments between parallel and serial speedup is due to three primary factors. First, the complexity and particular shape of the parts themselves. The example part in *Figure 10a* has only one curved surface, while that in *Figure 9* contains many dozens. A more general analysis of speedup factors would require testing the software against a set of benchmark parts of varying degrees of complexity, e.g. parts with many features, parts with difficult surfaces, parts with both.

The second factor in these variations is the environment for the experiments, which were conducted on a busy network of heterogeneous multiuser machines. The data were collected under everyday operating conditions and is intended as part of our demonstration of the feasibility of the technique. Because it was not feasible to create controlled experimental conditions, the data are only presented as a general indication of the technique's potential.

The third factor contributing to the variability among examples (and among feature recognition systems in general) is the treatment of feature accessibility for machining. In the system used as a basis for the approach in this paper[27], each machining feature has associated with it an *accessibility volume* which approximates the non-cutting portion of the cutting tool and tool assembly (an example from Reference 26 is shown in *Figure 12*). Testing each feature to ensure that the accessibility volume does not interfere with the final part requires a considerable amount of geometric computation—computation which varies greatly depending on the shape of the individual part.

We believe that parallelized trace-based feature recognition is highly suitable for parts in which the feature instances themselves are relatively simple, but numerous. It is not as well suited to problems where the feature instances themselves have very complex geometric configurations.

CONCLUSIONS

The focus of this research was to demonstrate the feasibility of using multiprocessor architectures to enable large increases in computational power for geometrically intensive CAD problems. As collaborative engineering pushes more downstream manufacturing issues into the design phase, the need to build effective and interactive CAD software systems requires an increasingly sophisticated allocation of computational resources.

The contributions described in this paper include our initial work toward an approach for performing trace-based feature recognition using a multiprocessor architecture. We present a commonly addressed collection of features and illustrate how to identify a task

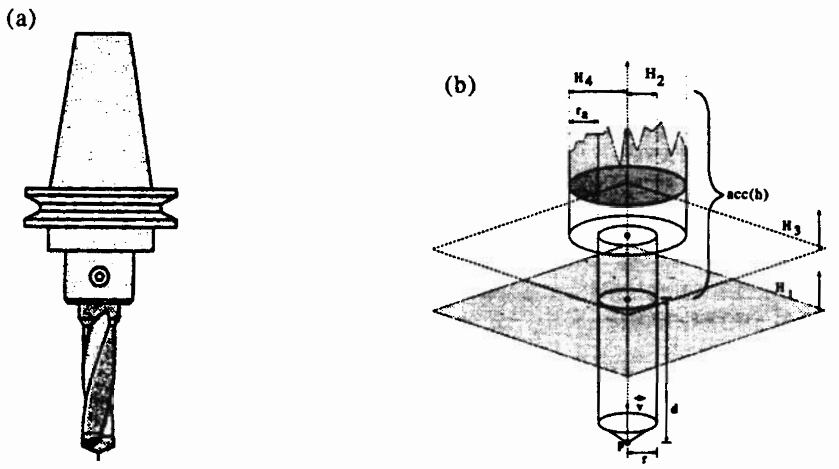

Figure 12 An illustration of a tool assembly and accessibility volume for drilling features. Testing accessibility conditions adds significantly to the cost of recognizing features for certain types of parts: (a) a drilling tool assembly, from Reference 29; (b) drilling feature *h* and accessibility volume acc(*h*)

Department of Statistics and Computer Science, West Virginia University, April 1993.
20. Laakko, T. and Mäntylä, M., Feature modelling by incremental feature recognition. *Computer-Aided Design*, 1993, **25**(8), 479–492.
21. Marefat, M. and Kashyap, R. L., Geometric reasoning for recognition of three-dimensional object features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1990, **12**(10), 949–965.
22. Narayanaswami, C. and Franklin, W. R., Determination of mass properties of polygonal csg objects in parallel. In *Symposium on Solid Modeling Foundations and CAD/CAM Applications*, ed. J. Rossignac and J. Turner, Austin, TX, June 1991. ACM SIGGRAPH, ACM Press, 1991, pp. 255–267.
23. Pinilla, J. M., Finger, S. and Prinz, F. B., Shape feature description using an augmented topology graph grammar. In *Proceedings NSF Engineering Design Research Conference*. National Science Foundation, 1989, pp. 285–300.
24. Powell, M. L., Kleimna, S. R., Barton, S., Shah, D., Stein, D. and Weeks, M., Sunos 5.0 multithead architecture. Technical Report SunSoft, 2550 Garcia Avenue, Mountain View, CA 94043, 1991.
25. Prabhakar, S. and Henderson, M. R., Automatic form-feature recognition using neural-network-based techniques on boundary representations of solid models. *Computer-Aided Design*, 1992, **24**(7), 381–393.
26. Regli, W. C., Geometric algorithms for recognition of features from solid models. PhD Thesis, The University of Maryland, Collage Park, MD, 1995.
27. Regli, W. C., Gupta, S. K. and Nau, D. S., Extracting alternative machining features: An algorithmic approach. *Research in Engineering Design*, 1995, **7**(3), 173–192.
28. Safier, S. A. and Finger, S., Parsing features in solid geometric models. *European Conference on Artificial Intelligence*, 1990.
29. Sandvik Coromant *Catalog CMP90-R94.2*, 1994.
30. Strip, D. and Karasick, M., Solid modeling on a massively parallel processor. *International Journal of Supercomputing Applications*, 1992, **6**(2), 175–192.
31. SunSoft, *Multithreaded Programming Guide*. Sun Microsystems, 2550 Garcia Avenue, Mountain View, CA 94043, 1994.
32. Umar, A., *Distributed Computing: A Practical Synthesis*. Prentice-Hall, Englewood Cliffs, NJ, 1993.
33. Vandenbrande, J. H. and Requicha, A. A. G., Spatial reasoning for the automatic recognition of machinable features in solid models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1993, **15**(12), 1269–1285.

William Regli is a National Research Council Postdoctoral Research Associate in the National Institute of Standards and Technology's Manufacturing Systems Integration Division. His research interests include integration of distributed manufacturing systems, Internet technology, and computer-integrated design and manufacturing.

Dr Regli received a BS in Mathematics and Computer Science from Saint Joseph's University in Philadelphia in 1989 and a PhD in Computer Science from the University of Maryland at College Park in 1995. He is the recipient of the University of Maryland's Institute for Systems Research Outstanding Graduate Student Award (1994–1995), NIST Special Service Award (1995), and General Electric Corporation Teaching Incentive Grant (1994–1995), among other awards. He is a member of ACM, ASME, IEEE, AAAI, and Sigma Xi and on the editorial board of the journal IEEE Internet Computing. *Dr Regli has authored or co-authored more than 40 technical publications. Copies of recent papers are available at http://elib.cme.nist.gov/msidstaff/william.regli.html.*

Satyandra K Gupta is a research scientist in the Robotics Institute at Carnegie Mellon University. His research interests include design for manufacturability, computer-aided process planning, automated redesign, and concurrent engineering.

Dr Gupta received a BE in Mechanical Engineering from University of Roorkee in 1988. He received an MTech in Production Engineering from Indian Institute of Technology, Delhi in 1990. He received a PhD in Mechanical Engineering from University of Maryland in 1994, where he has been supported by a Graduate School Fellowship and an Institute for Systems Research Graduate Fellowship. Awards received by Dr Gupta include a Gold Medal for first rank in BE (1988), a Gold Medal for the best BE Project, the ISR Outstanding Systems Engineering Graduate Student award (1993–94), and a Best Paper Award in ASME's Computers in Engineering Conference (1994).

Dana S Nau is a professor at the University of Maryland, in the Department of Computer Science and the Institute for Systems Research. He is also an affiliate professor in the Institute for Advanced Computer Studies (UMIACS) and the Department of Mechanical Engineering. His research interests include AI planning and searching, and computer- integrated design and manufacturing.

Dr Nau received his PhD in Computer Science from Duke University in 1979, where he was an NSF graduate fellow. He has more than 150 technical publications; copies of recent papers are available at http://www.cs.umd.edu/users/nau. He has received an NSF Presidential Young Investigator award (1984–89), the ISR Outstanding Systems Engineering Faculty award (1993–94), and several 'best paper' awards. In 1996 he was named a Fellow of the AAAI (American Association for Artificial Intelligence).