Knowledge Systems Laboratory **May 1990**
Report No. KSL-89-68

The Parallel Solution of Classification Problems

by
Hirotoshi Maegawa

Knowledge Systems Laboratory
Department of Computer Science
Stanford University
Stanford, California 94305

and

Corporate Research Laboratories
Sony Corporation
6-7-35 Kitashinagawa
Shinagawa, Tokyo 141, Japan

This research was supported by DARPA Contract F30602-85-C-0012 and NASA Ames Contract NCC 2-220-S1.

Table of Contents

Abstract

We developed a problem solving framework called ConClass capable of classifying continuous real-time problems dynamically and concurrently on a distributed system. ConClass provides an efficient development environment for describing and decomposing a classification problem and synthesizing solutions. In ConClass, designed concurrency of decomposed subproblems effectively corresponds to the actual distributed computation components. This scheme is useful for designing and implementing efficient distributed processing, making it easier to anticipate and evaluate the system behavior. ConClass system has an object replication feature in order to prevent a particular object from being overloaded. An efficient execution mechanism is implemented without using schedulers or synchronization schemes liable to be bottlenecks. In order to deal with an indeterminate amount of problem data, ConClass dynamically creates object networks to justify hypothesized solutions and thus achieves a dynamic load distribution. We confirmed the efficiency of parallel distributed processing and load balancing of ConClass with an experimental application.

1. Introduction

In this paper we describe a framework for the parallel solution of classification problems. We developed a framework called ConClass (Concurrent Classification) on a distributed-memory multiprocessor system. ConClass is based on the inherent parallel characteristics of classification problems and is capable of solving real-time problems continuously and concurrently.

Classification is one of the more commonly used problem solving methods and has been used in diverse areas such as engineering, biology, and medicine. The classification problem solving model provides a high-level structure for the decomposition of problems, making it easier to recognize and represent similar problems. Classification is the act of identifying an unknown phenomenon as a member of a known class of phenomena. The process of identification is that of matching the observations of an unknown phenomenon to the features of known classes. Classification problem solving is described in detail in [Clancey 84, 85].

Most previous systems solve classification problems in series in terms of classification processes and deal with static data [Buchanan 84, Bennett 78, Rich 79, Brown 82]. Related work on classification is based on static aspects as well [Cohen 85, Bylander 86]. Recent AI research has, however, focused on real-time problem solving such as that surveyed in [Laffey 88]. For example, problems from the engineering field are dynamic in domains such as continuous signal understanding and manufacturing diagnostics.

Our motivation was to develop a framework capable of describing and solving continuous real-time classification problems in parallel. We implemented the ConClass system based on the inherent parallel nature of classification problem solving.

Another area of our research interest concerned finding out the fundamental requisites of parallel distributed classification and implementing an efficient framework based on such intrinsic features. ConClass achieved efficient parallel computation and linear speedup against the number of processing elements.

We developed ConClass on a simulated distributed-memory multiprocessor system called CARE [Delagi 87a] using a distributed processing language called LAMINA [Delagi 87b].

We implemented an experimental classification system in ConClass to evaluate the performance of ConClass. This system classifies observed aircraft by using continuous abstract radar signal data.

This research is a part of the Advanced Architectures Project at the Knowledge Systems Laboratory at Stanford University, a section of which has been dedicated for research of distributed processing [Rice 89].

In subsequent sections we describe the methodology of parallel classification problem solving, the implementation of ConClass on a distributed system, and finally, an evaluation using the experimental application.

2. Parallel Classification Methodology

2.1. Problem Decomposition

A classification problem can be structured as a directed acyclic graph whose nodes are decomposed subproblems. A classification solution of decomposed subproblem can be supplied to other subproblem solvers. A solver may synthesize other classification solutions. Propagation of problem data and solutions is hierarchically organized in this manner. Thus, classification problem solving can be organized intrinsically hierarchical and distributed.

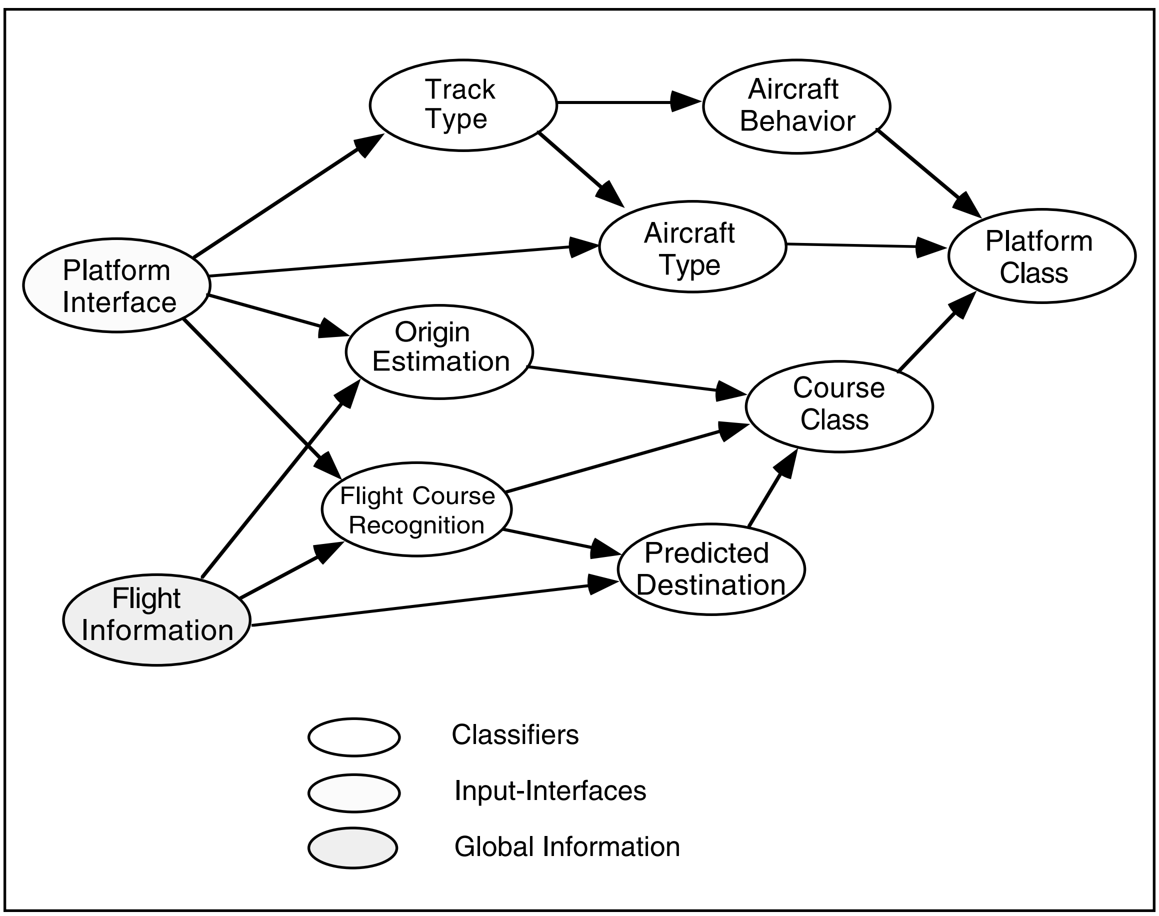

Figure 2.1. Hierarchical Classification Configuration

Figure 2.1 shows an example of a problem solving system which classifies aircraft represented by radar signals. We denote such an aircraft problem object *a platform*. This system solves classification subproblems such as aircraft type and flight course recognition and then provides final classifications such as commercial, military, or a smuggler's aircraft. Example solutions are shown in Figure 2.3. Attribute values of platforms change over time and the classification state of the system varies according to these data. The solutions may change due to global information such as flight plans and ground circumstances as well. Classification system can accept different kinds of problem inputs such as aircraft location and velocity, observed maneuverability, and radar signature.

Thus, classification problem solving has characteristics suitable for parallel processing using decomposed subproblem solvers especially for continuous dynamic problems. We implemented ConClass using such inherent characteristics.

The parallel processing in ConClass is designed using decomposed classification subproblem solvers. The ConClass system represents those problem solvers as parallel processing elements and allocates them directly on computational hardware components. This decomposition scheme makes designed concurrency effectively correspond to actual parallel computation. The scheme makes it easier to anticipate and evaluate the system behavior for obtaining efficient concurrency. We call a decomposed problem solver *a classifier*. We call the problem solving network consisting of the classifiers *the classifier network*. A classifier whose classification is derived by other classifiers is denoted a super-classifier of those classifiers. A classifier whose solutions is synthesized in other classifiers is called a sub-classifier of those classifiers. Classifiers can act concurrently and dynamically when problem data and solutions are propagated over time.

Problems in ConClass may be created dynamically such as an aircraft platform captured by a radar system. In addition, ConClass is capable of manipulating multiple sources of a continuous problem. ConClass has an object which links problem objects to the entrance classifiers. We call the linking object *an interface-object*. When a problem is created dynamically, the problem object receives references to the entrance classifiers from an interface-object and starts sending them problem data.

A classifier has known classes of phenomena into which problem objects are classified as solutions. We call such a class *a classification-category*. If a classifier succeeds in classifying, it sends the solution to its abstract classifiers, that is, super-classifiers. The abstract classifiers classify the problem by using the solution and propagate their classification solutions in the same manner. Each classification computation is a decomposed subproblem solving mentioned above. The ConClass classification system may have more than one of the most abstract classifiers to obtain different kinds of solutions to the entire classification problem.

2.2. Load Distribution

The research goal of ConClass is to implement efficient distributed processing as well as to develop a framework for describing and decomposing classification problems. In order to distribute decomposed problem solvings, we have two schemes: replication of objects and dynamic distribution of problem solving tasks.

A classifier acts when it receives a solution from its sub-classifier and when the sub-classifier changes the solution. Therefore, classifiers that are lower in the hierarchy usually execute a larger amount of classifications than those higher in the hierarchy. Even in the same level of the hierarchy, classification computations may differ between the classifiers. In order to achieve efficient load balancing to such objects which can have varying execution burdens, ConClass replicates classifiers and interface-objects. The replicated objects have the same functions and need no communication between themselves. A problem object's data are assigned to the replicated objects by hashing their object identifications. The number of replicated objects can be determined by domain knowledge, statistics, and user design. Although ConClass does not perform dynamic replication of objects currently, it is possible to replicate objects at execution time because problem solutions between replicated objects can be independent.

ConClass manipulates an unknown number of problem objects which are dynamically created. Increasing the number of these objects may cause the classifier network to become overloaded. Therefore, ConClass uses the classifier network to produce hypotheses of individual problem objects as solutions and creates another network to maintain these hypotheses. *A hypothesis* is a classification solved by one of the most abstract classifiers. ConClass executes classifications to justify the created hypotheses on the dynamically created network. We call such a created network *an instance network*. Instance networks are organized for individual classification problem objects independently. Classifications after a hypothesis is formed are computed on the different network so that this scheme provides dynamic load distribution. We call the classification process to make hypotheses *the initial hypothesis formation* and call the justification of hypotheses *the dynamic hypothesis maintenance*.

Initial Hypothesis Formation:

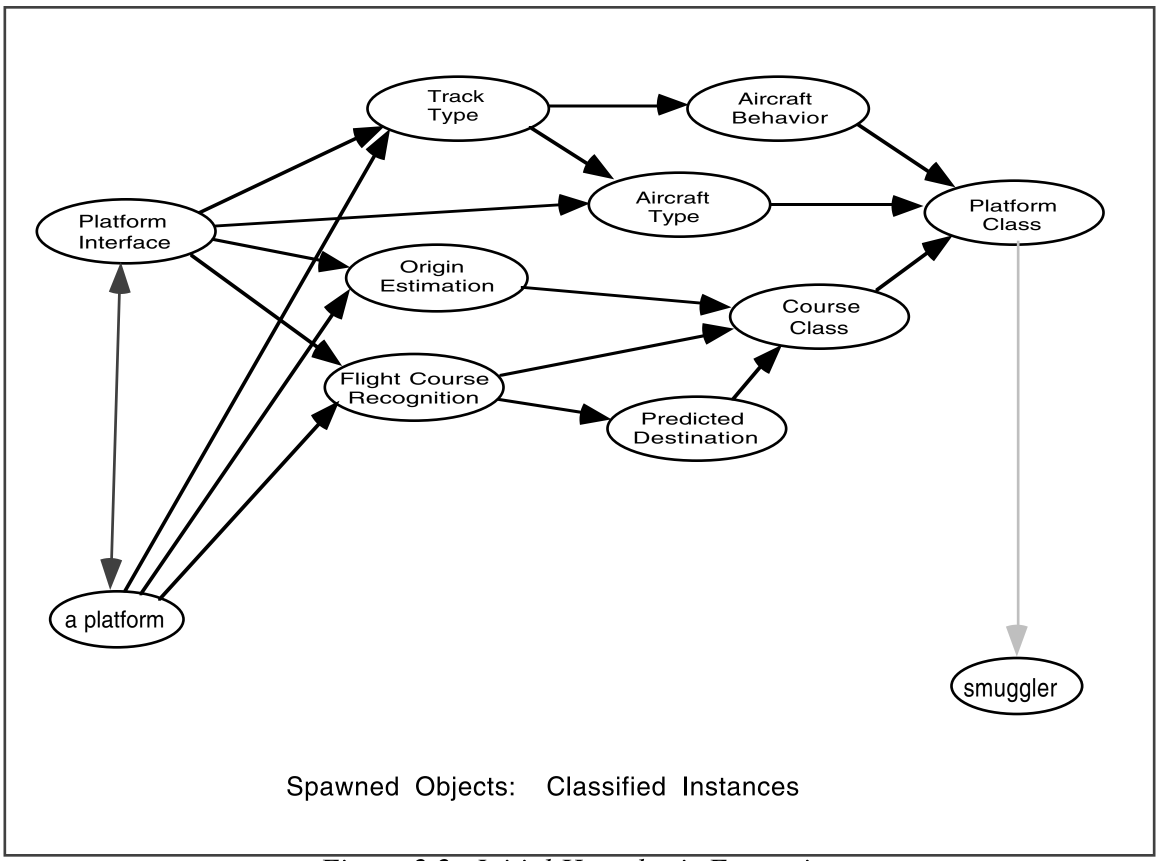

Figure 2.2. Initial Hypothesis Formation

A hypothesis formed by one of the most abstract classifiers is called *an initial hypothesis*. When the classifier network succeeds in forming an initial hypothesis, the classifier instantiates the classification-category corresponding to the hypothesis as a newly created computation object. This production scheme of an initial hypothesis for a given problem on the classifier network is the initial hypothesis formation as shown in Figure 2.2. We call an instantiated object of a classification-category *a classified-instance*.

Dynamic Hypothesis Maintenance:

One of the most abstract classifiers, which forms a hypothesis, makes its sub-classifiers instantiate the classification-categories which derive this hypothesis. These classifiers propagate the instantiation of classification-categories to their sub-classifiers in the same manner. When the entrance classifiers make classified-instances, the problem object which receives the hypothesis obtains links to those classified-instances. The reason for creating classified-instances backwards in this method is so that only required classified-instances are instantiated. The set of created classified-instances is an instance network.

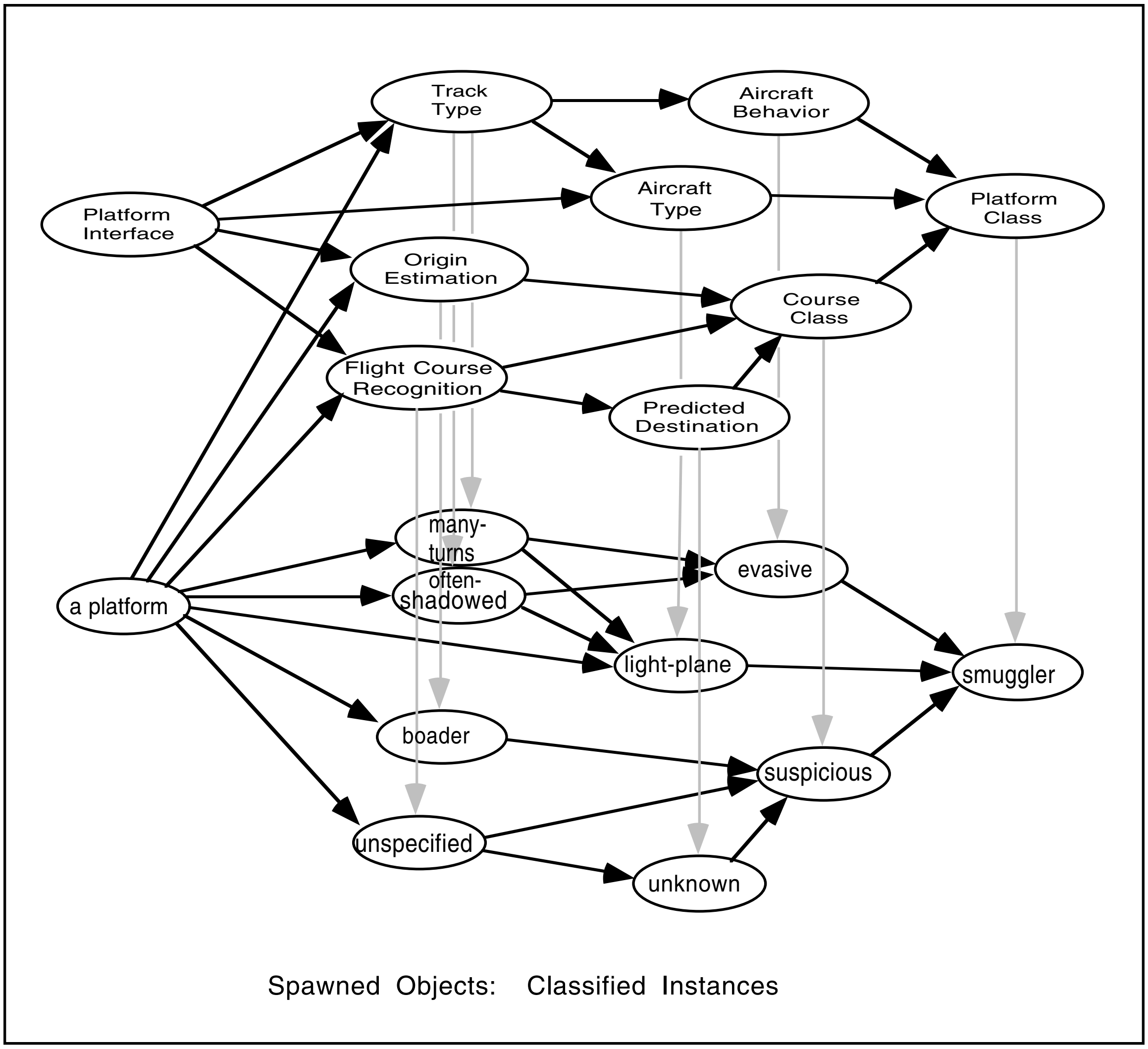

Figure 2.3. Dynamic Hypothesis Maintenance

Problem data is propagated through its instance network to justify the classification solutions in the classified-instances. If a classification of a classified-instance gets disproved, the classified-instance discards itself, propagates the negated solution to its super-classifiers and super-classified-instances, and notifies its sub-classifiers of its elimination. These sub-classifiers discard their classified-instances which have derived only the disproved classification solution. If a sub-classified-instance derives another classification solution, it is retained. If a super-classified-instance is discarded by the negation, it executes the same procedure. When the most abstract classified-instance is discarded, the hypothesis as a problem solution is denied. This scheme of instantiating, justifying, and discarding an instance network is the dynamic hypothesis maintenance as shown in Figure 2.3.

While instantiating and discarding an instance network, the problem may vary continuously. When a classifier receives a solution from its sub-classifier after creating an classified-instance, the problem data for the corresponding classification-category is forwarded to the classified-instance. After a classifier recognizes an instantiation in its super-classifier, it sends the problem data to the super-classified-instance directly. If a classified-instance is eliminated, the problem data is forwarded to its classifier. A classified-instance is acting for a while for the data forwarding even after it is discarded. The classified-instance is actually discarded when its sub-classifier discards its reference.

A problem which has a hypothesis may succeed in forming another hypothesis so that the classifier network continues to work classifying the problem. In this circumstance a classifier does not invoke classifications for the instantiated classification-categories. Therefore, the classifier network can reduce its computation load after the hypothesis has been formed. When a problem has more than one hypothesis simultaneously, those classified-instances needed for both of these hypotheses are shared on the same instance network.

An instance network is organized only after a solution is formed in one of the most abstract classifiers, in order to create longer-lived classified-instances. Less abstract classification acts more frequently due to the problem propagation scheme of ConClass. If a classifier lower in the hierarchy instantiates a classified-instance, it may be quickly eliminated by a reclassification. Creation of such an ephemeral distributed object is expensive to manage.

3. Implementation of ConClass

3.1. Computational Environment

We developed ConClass on a simulated distributed-memory multiprocessor system called CARE [Delagi 87a] on a Lisp machine, Explorer[1], and implemented ConClass in a distributed processing language called LAMINA [Delagi 87b].

CARE is a distributed-memory, asynchronous message-passing architecture. CARE is simulated by a general, event driven, highly instrumented system called SIMPLE. CARE models 1 to 1000 processor-memory pairs communicating via a packet-switched cut-through interconnection network. Message delivery between processing elements is reliable, but messages are not guaranteed to arrive in the order of origination.

LAMINA is the basic language interface to CARE and consists of Common Lisp [Steele 84] and Flavors [Weinreb 80] with extensions. The extensions provide primitive mechanisms and language syntax for expressing and managing computational locality in each processing element and concurrency between processing elements. Three styles of

[1] CARE currently runs on Explorer, Symbolics, Sun-3, and DEC Station 3100.

programming are supported: functional, shared-variable, and object-oriented. ConClass system is implemented in the object-oriented language subset of LAMINA where we represented ConClass objects such as classifiers by means of LAMINA objects.

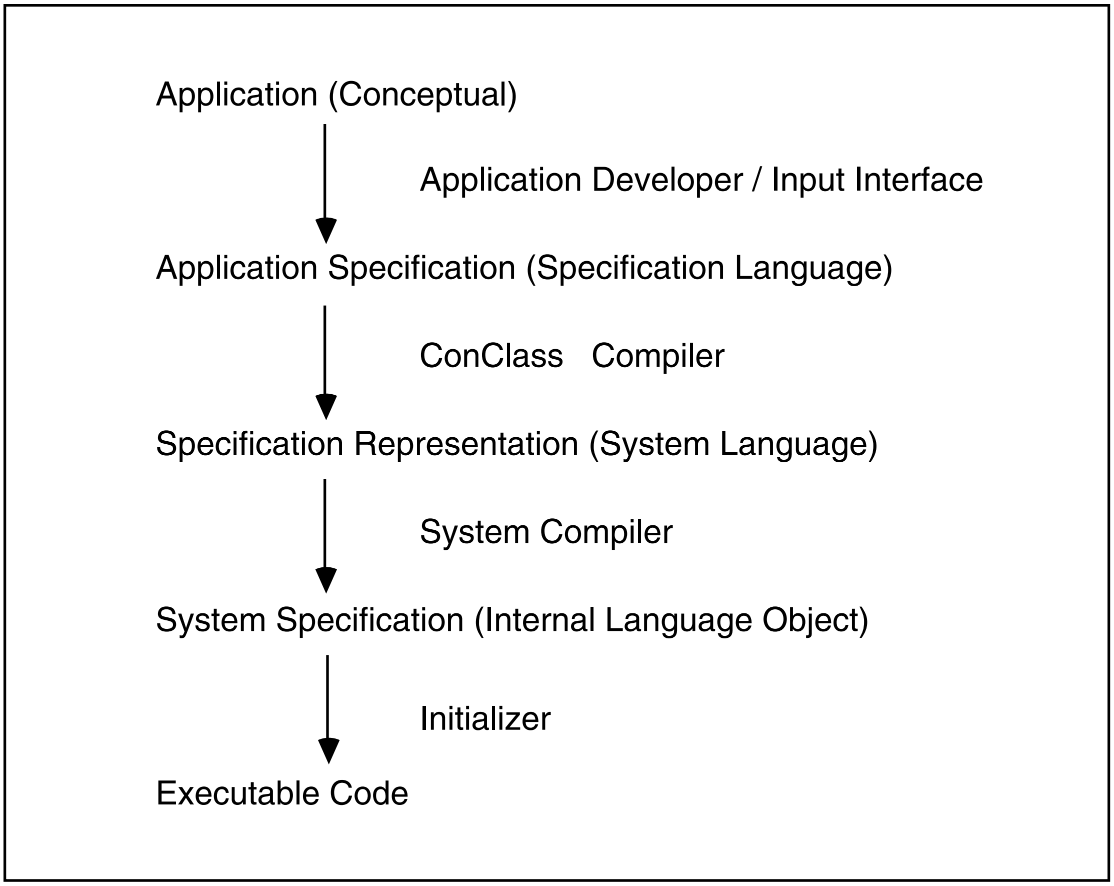

Figure 3.1. Representation Specification Hierarchy

3.2. Problem Description and Solving

Figure 3.1 shows the hierarchy of representation specifications and execution components in ConClass system. ConClass provides application developers with an environment for describing and decomposing classification problems. Figure 3.2 shows an example classifier definition in the application specification. The ConClass compiler translates application descriptions to object definition representations.[2] The initializer initiates LAMINA objects according to the definitions and allocates them on CARE processing elements.

ConClass system provides a development environment where application developers can specify classifier definitions and relationship between classifiers and interface-objects. Classification in a classifier is composed of classification-categories. A known class of phenomenon into which unknown phenomena are classified is defined by describing *templates* in the classification-category. *A template* is a conjunction of attribute values used

---

[2] We do not have the ConClass compiler implemented to date. We specified the experimental system described in section 4 directly in the object specification representation. However, the scheme for representing and decomposing problems was efficient for development.

in the classification-category and its classifier. An attribute can specify its condition by means of a value, a set of values, or a range of numerical values. A template succeeds in matching a problem object when the problem object's attributes satisfy the template value conditions. *An attribute* is problem data or a classification solution brought by the other classifiers or problem objects. *An attribute* can also be computed from other attributes anew in a classifier. A classification-category may have more than one template and it succeeds in classifying if the conditions of one of the templates are satisfied.

```
#| classifier-definition |#
(Speed
  :Classification-Categories (Slow-Current-Speed
                              Medium-Current-Speed
                              Fast-Current-Speed
                              Slow-Max-Speed
                              Medium-Max-Speed
                              Fast-Max-Speed)
  :Classifier-Inputs ((X-Position Input-Interface)
                      (Y-Position Input-Interface)
                      (Z-Position Input-Interface)
                      (X-Velocity Input-Interface)
                      (Y-Velocity Input-Interface)
                      (Z-Velocity Input-Interface))
  :Classifier-Database ()
  :Classifier-Attributes (Current-Speed Max-Speed)
  :Super-Classifiers (Track-Type Current-Platform-Behavior)
  :Interface-Objects (Input-Interface)
  :Output-Objects ()
  :Locations (25 26 27 28)
  :Dynamic-Site-Positions ())

#| classification-category-definition |#
(Slow-Current-Speed
  :Classifier Speed
  :Category-Inputs ()
  :Category-Database ()
  :Category-Attributes ()
  :Classification-Templates
  ;;; templates: (template ... )
  ;;; template: (template-slot ... )
  ;;; template-slot: (var (capture-values lock-values)
                          (capture-confidence lock-confidence))
  ;;; values: (:set value ... ),
             (:range (:open value) (:close value)),
             (:range t (:open value)), ...
   ((1.0 (Current-Speed ((:range (:open :infinity-) (:close 5000))
                         (:range (:open :infinity-) (:close 6000)))
                        (.7 .5)))))
```

Figure 3.2. Sample Classifier Definition

Problems manipulated in ConClass can be continuous and dynamic. If problem data causes attributes to vary around the threshold of template conditions, a classification may change frequently. Therefore, a template condition can be specified by a set of two kinds of values which we call *capture value* and *lock value*. *A capture value* and *a lock value* are used when unknown problem objects are classified and when classified solutions are justified, respectively. The range of a lock value needs to be larger than that of a capture value.

Classification solutions and attributes can carry confidence values. When a template's condition is satisfied, its classification confidence is the minimum value of attribute confidences in the template. A template can specify a minimum requirement on its confidence value, which must be satisfied for a successful classification. If the conditions of more than one template are satisfied, the classification confidence takes the maximum value of those template confidences. Some kinds of symbolic confidences are allowed to be used. This scheme is one of the most conservative methods to calculate confidences. A detailed description of confidence is illustrated in [Buchanan 84].

When a classifier succeeds in classifying, if it is one of the most abstract classifiers, it instantiates an initial hypothesis. Otherwise, it propagates the classification solution with specified attribute values to its super-classifiers. When a classification confidence is changed significantly, the change is propagated to the super-classifiers or to the super-classified-instances.

Application developers describe definitions of classifiers involving classification-categories and relationships between classifiers and interface-objects. Developers also define attributes of interface-objects. Developers need to define procedures for evaluating attributes and those confidences used in matching problem objects to the templates. ConClass generates the definition of a classified-instance according to the definitions of the corresponding classification-category and its classifier. The example shown in Figure 3.2 is the definition of a classifier with one of its classification-categories, which is to classify aircraft speed. This is a definition used in the experimental application described in Section 4.

3.3. Special Internal Controls

ConClass does not use physical synchronization schemes which may result in a saturation effect. ConClass incorporates embedded control features to manage a variety of asynchronous aspects of distributed processing.

We can use managers or schedulers responsible for creating and maintaining dynamic objects, synchronizing different processes, and coordinating searches. However, such agents may limit the system throughput when managing synchronization. Our related work reports various problems about such scheduling [Noble 88, Muliawan 89]. Schedulers can be overloaded, however, there are no clear-cut rules for the decomposition of such objects. ConClass uses no scheduler objects and handles no physical synchronization between objects.

In ConClass, variation of a problem object is propagated on the classifier network and instance networks by classifying and reclassifying the problem. The creation and elimination of instance networks are achieved by means of the propagation of creation and discard requests of classified-instances between objects, respectively. These propagation schemes do not require synchronization. However, such propagations may occur simultaneously and cause state conflicts in an object. For example, a classifier may receive a request for a classified-instance creation from its super-classifier while its sub-classifier is sending a message of disproving the classification. Each object of ConClass manages various requests efficiently considering the state transitions of instantiation and use no synchronization which may make other objects idle.

Classifiers and interface-objects are represented by means of LAMINA objects as described above. Classifiers and interface-objects communicate with each other using the message passing facilities of CARE and LAMINA. Messages between objects in ConClass are not guaranteed to arrive in the order of origination because the message passing on CARE is asynchronous. For example: An object may receive stale data later than brand-new data. When a classified-instance is discarded shortly after being created, its sub-classifier may receive a discard request earlier than a creation one for its related classified-instances. ConClass adopts embedded features to properly manipulate all messages that are in the wrong order.

Classified-instances and problem objects are created dynamically and those references are propagated to other objects. A dynamic object is typically created on a different processing element than that of the creator according to the object allocation scheme described below. Although the creator does not receive a created object's reference until a later time, it keeps indirect reference to the new object, which can be used to send messages. The creator sends the indirect reference to other objects if the new object is being created so that the messages from those objects to the created one are sent via the creator indirectly. The direct reference is later propagated to those objects that hold an indirect reference automatically.

These features were useful for implementing the ConClass system.

3.4. Load Balancing

It is one of the goals of parallel distributed processing to allocate objects over processing elements such that the work they do is balanced as evenly as possible. We adopted the same modified random load balancing used by our related work [Nakano 88] to allocate classified-instances. This scheme involves random selection for dynamic objects from the set of all processing elements excluding those used by static objects if there are fewer static objects than processing elements. Otherwise, the dynamic object is allocated randomly from the set of all processing elements. The random allocation of classified-instances is reasonable because it is difficult to predict that any given classified-instance will be busier than another and because it is not suitable to allocate on the basis of statistics concerning non-permanent objects. In fact, empirical evidence suggests that in the absence of such load knowledge, random allocation is optimal [Nakano 88].

We allocated another sort of dynamic object, problem object, evenly on the processing elements dedicated to dynamic objects. Because problem objects, for example, aircraft platforms, exist more permanently, processing elements are assigned using a round-robin method.

Static objects, classifiers and interface-objects, can be replicated as much as desired as described above. These objects are allocated to the processing elements dedicated to static objects in advance according to domain knowledge, statistics, and user definition[3].

4. Performance Evaluation

We implemented an experimental application system in ConClass and confirmed the efficiency of the ConClass system.

---

[3] We defined the allocation of static objects in implementing the experimental application system because of the absence of the ConClass compiler.

4.1. Experimental Problem

We have been developing an aircraft radar signal interpretation system called AirTrac for tracking and classifying aircraft. The AirTrac system is composed of three major modules: Data Association, Path Association, and Platform Interpretation. Data Association accepts aircraft signal reports of multiple radar systems at regular time intervals and periodically abstracts the radar signal reports into observation records for individual signal tracks [Nakano 88]. Path Association reports hypothesized platforms to which the periodic observation records are associated to form tracks for the same aircraft [Noble 88, Muliawan 89]. Platform Interpretation analyzes and interprets information contained in platforms and provides continuous real-time assessments about the observed aircraft.

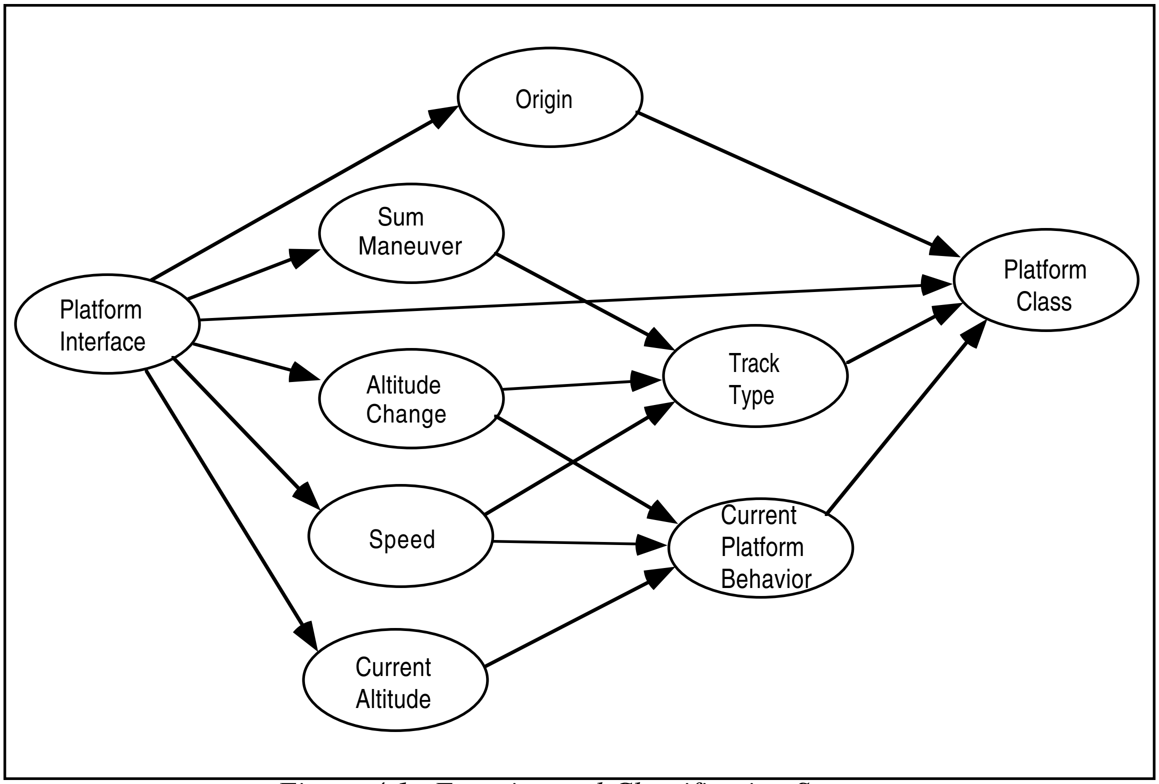

Figure 4.1. Experimental Classification System

The problem selected for our experiments is a simplified experimental implementation of AirTrac's Platform Interpretation module. The configuration of the experimental system is shown in Figure 4.1. Each sub-problem solver is implemented by means of a classifier and its classification facilities. This system consists of eight classifiers which have between two and ten classification-categories. The system input is a series of simplified emulated aircraft platforms which have aircraft position and sizes information.

The experimental application requires the following:

- The classification system is a hierarchy of classifiers which have multiple fan-in and fan-out.
- The classifier network has cut-through connections.
- The problem data is continuous and problem solving in each classifier is potentially dynamic.

These requirements are in order to evaluate the experimental system in an environment where configuration and computation are uneven between problem solvers. The experimental system meets these requirements.

4.2. Experimental Results and Analysis

We experimented with the data of 50 aircraft platforms which appeared in real-time successively and were classified and reclassified typically three times in the classifications lower in the hierarchy. The experiment has two parameters: the number of processing elements and the data rate. The numbers of processing elements used were 8, 16, 32, 64, and 256. The data rate is the frequency at which problem data is fed into the application system. We can change the data rate by altering the sampling frequency of observed problem data. This scheme, however, will change the frequency of classification in relation to the data rate. In order to maintain the classification quality between the data rates, we changed the data rate by altering the time interval of feeding the same set of data. Thus, the experimental application system was performed using the time of the emulated data while we evaluated the performance of ConClass system using the simulation time of the CARE system.

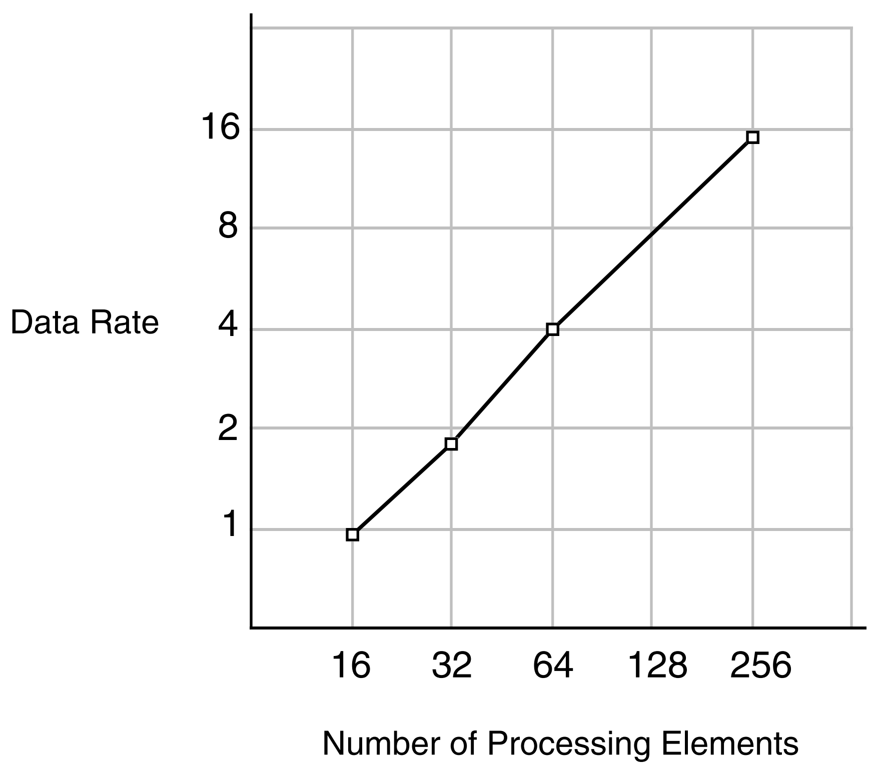

Figure 4.2. Speedup Curve (based on sustainable data rate)

The experimental system achieved a linear speedup against the number of processing elements as shown in Figure 4.2. The speedup was based on the sustainable data rate, the maximum data rate for which all measured latencies stabilize and do not increase over time. See appendices for the observed latencies from which the speedup was evaluated. The load balancing in ConClass uses two methods: replication and allocation of static objects, and the assignment of processing elements for dynamic objects. We assumed that we could optimize these factors using domain knowledge and statistics. Therefore, we fine-tuned the factors in the experiment so as to optimize the results. Another reason for the optimization was to evaluate the processing speed with respect to achieving an efficient concurrency. In addition, we implemented the ConClass system paying attention to even the execution efficiency of Lisp functions. This was to more precisely evaluate the system overhead for parallel processing.

The CARE simulation system has an user interface where we can observe a variety of statistics and latencies of CARE components. Figure 4.4 shows the processor utilization graph whose upper half specifies utilization of evaluators which execute actual data computation. The lower half specifies that of operators which manage the communication between processing elements. In a typical classification situation, for example, ConClass was able to use 28 to 30 processing elements at a time out of a possible 32. Including the initialization of ConClass, which brought about considerable computation, the overall average of concurrent utilization was 21 processing elements. Because the classification computation in ConClass is coarse-grained, the operators are not busy.

In ConClass, the concurrency designed by an application developer can correspond effectively to actual computational hardware components. We were able to implement the experimental application system efficiently using this scheme. It was easy to estimate replication and allocation of objects and assign processing elements. The ConClass development environment for describing and decomposing classification problems was useful. ConClass execution facilities excluding schedulers and various synchronization schemes improved the efficiency of parallel distributed processing.

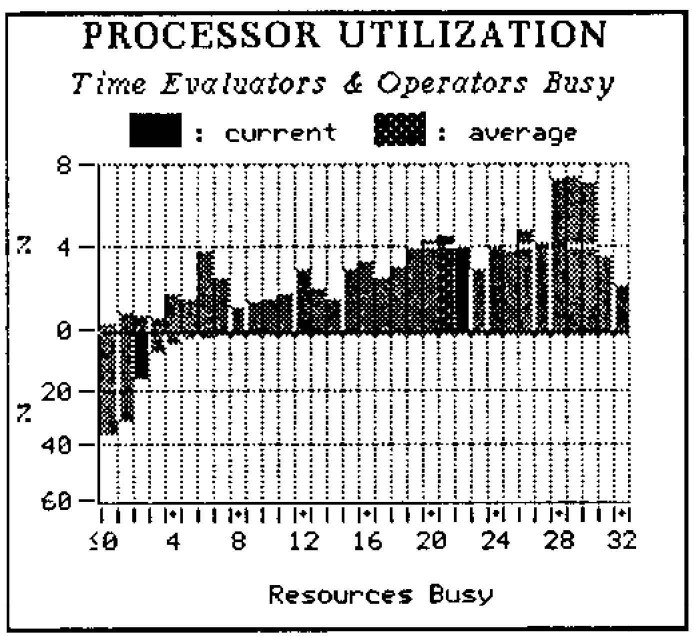

Figure 4.3. Processor Utilization

Table 4.1 shows the frequencies of messages sent between objects to solve the experimental problem. These frequencies correspond to sustainable data flows. Most messages are propagated between classifiers and between classified-instances. Messages can be sent between the classifier network and instance networks while creating and discarding classified-instances. Data propagation messages can be sent from classified-instances to their related but uninstantiated classification-categories. The former messages are not frequent because classified-instances are long-lived according to the instantiation scheme of ConClass as described above. The latter situation is fairly rare due to the classification problem's structure. The ratio of number of messages to classified-instances decreases as the number of processing elements increases. This is because the instantiation time becomes longer compared to the experimental simulation length. However, this is not a factor which can affect the independence between the classifier network and instance networks. Although more dynamic problems may increase the interactions between the two kinds of networks, the experimental results show the efficiency of a dynamic load distribution of ConClass.

Table 4.1. Message Frequencies

Messages		Number of Messages (Percentage)			
	Processing Elements	16	32	64	256
To Classifiers		5448(51.0)	5445(51.2)	5527(56.1)	5409(79.3)
	From Instances	313 (2.9)	268 (2.5)	389 (4.0)	131 (1.9)
To Classified-instances		5248(49.0)	5181(48.8)	4317(43.9)	1409(20.7)
	From Classifiers	167 (1.6)	180 (1.7)	273 (2.8)	206 (3.0)

5. Conclusions

In this paper we have described the parallel solution of classification problems. The developed framework, ConClass, is capable of classifying continuous real-time problems dynamically and concurrently.

ConClass provides a high-level structure for describing and decomposing classification problems. The ConClass classification system can handle multiple sources of problem inputs as well as dynamic global information. A ConClass application can use static knowledge to solve problems in the system. Such a high-level framework was useful in implementing the experimental application in ConClass.

Classification problem solving can be structured hierarchically by means of decomposing problem and synthesizing solutions. We implemented the ConClass framework based on this characteristic so that decomposed problem solving modules were directly represented as distributed processing components. Therefore, the concurrency designed by developers is effectively reflected in the actual parallel computation and this scheme makes it easier to anticipate and evaluate the system behavior. Moreover, a decomposed classification problem solver, consisting of a classifier and its classified-instances, is very uniform in terms of its basic structure and execution mechanism. These features are useful in the design of concurrency and the implementation of efficient distributed processing. The classification execution in ConClass is intrinsically parallel, in contrast to our previous problem solving frameworks [Brown 86, Nii 89, Saraiya 89] which report various problems of parallel processing.

We implemented the replication features of static objects for preventing a particular object from being overloaded. The dynamic creation of problem objects may cause the system load to increase. We incorporated the load distribution scheme by means of dynamically creating instance networks which maintain hypotheses as solutions of problem objects. We implemented an efficient execution mechanism for ConClass without using schedulers or synchronization schemes which are liable to be bottlenecks. We confirmed the efficiency of the parallel processing and the load balancing of ConClass by an experiment.

ConClass is a concurrent problem solving framework using a structural hierarchy of classification problem and continuity of problem data. Real-time problem solving systems are increasing in importance and we realize the advantage of the ConClass framework. Furthermore, ConClass suggests a construct for dynamic information fusion and multiple assessments. AirTrac, a part of which we selected as an experimental application, is an example: AirTrac fuses information such as radar signal, flight plans, ground information, aircraft knowledge, and geography. AirTrac reports real-time assessments such as aircraft classifications and predictions of flight courses and aircraft actions. The hierarchical structure of decomposing a problem and synthesizing solutions are useful and effective for implementing these functions.

Acknowledgments

Many of these ideas were improved by discussions with Harold Brown. I would like thank him for providing valuable suggestions throughout this project. I would also like to thank Nakul Saraiya for his tireless support for using the CARE/LAMINA system. I am indebted to the members of the Advanced Architectures Project for nurturing a rich research environment. The Symbolic Systems Resorrces Group of the Knowledge Systems Laboratory provided excellent support of our computing environment. Special thanks are due to Edward Feigenbaum for his continued leadership and support of the Knowledge Systems Laboratory which made this research possible. George Fukuda provided an opportunity to do the reported research.

This research was supported by DARPA Contract F30602-85-0012 and NASA Ames Contract NCC 2-220-S1.

References

[Bennett 78] Bennett, J., Creary, L., Engelmore, R. and Melosh, R., SACON: A Knowledge-Based Consultant for Structural Analysis, *Technical Report HPP-78-23,* Stanford Univ. (1978).

[Brown 86] Brown, H. D., Schoen, E. and Delagi, B. A., An Experiment in Knowledge-based Signal Understanding Using Parallel Architectures, *Technical Report KSL-86-69,* Stanford Univ. (1986) and *Proceedings of DARPA Expert Systems Workshop* (1986) 93-105.

[Brown 82]	Brown, J. S., Burton, R. R. and De Kleer, J., Pedagogical, Natural Language, and Knowledge Engineering Techniques in *SOPHIE I, II, and III,* in Sleeman, D. and Brown, J. S. (Ed.), *Intelligent Tutoring Systems,* Academic Press (1982) 227-282.
[Buchanan 84]	Buchanan, B. G. and Shortliffe, E. H., *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project,* Addison-Wesley Publishing (1984).
[Bylander 86]	Bylander, T. and Mittal, S., CRSL: A Language for Classificatory Problem Solving and Uncertainty Handling, *AI Magazine* 7(3) (1986) 66-77.
[Clancey 84]	Clancey, W. J., Classification Problem Solving, *Technical Report STAN-CS-84-1018,* Stanford Univ. (1984) and *Proceedings of the AAAI-84* (1984) 49-55.
[Clancey 85]	Clancey, W. J., Heuristic Classification, *Technical Report KSL-85-5,* Stanford Univ. (1985) and *Artificial Intelligence* 27 (1985) 289-350.
[Cohen 85]	Cohen, P., Davis, A., Day, D., Greenberg, M. Kjeldsen, R., Lander, S., and Loiselle, C., Representativeness and Uncertainty in Classification Systems, *AI Magazine* 6(4) (1985) 136-149.
[Delagi 87a]	Delagi, B. A., Saraiya, N. P., Nishimura, S. and Byrd, G. T., An Instrumented Architectural Simulation System, *Technical Report KSL-86-36,* Stanford Univ. (1987) and *Proceedings of DARPA Expert Systems Workshop* (1986) 106-118.
[Delagi 87b]	Delagi, B. A., Saraiya, N. P. and Byrd, G. T., LAMINA: CARE Applications Interface, *Technical Report KSL-86-67,* Stanford Univ. (1987) and *Proceedings of the Third International Conference on Supercomputing* (1988) 12-21.
[Laffey 88]	Laffey, T. J., Cox, P. A., Schmidt, J. L., Kao, S. M. and Read, J. Y., Real-Time Knowledge-Based Systems, *AI Magazine* 9(1) (1988) 27-45.
[Muliawan 89]	Muliawan, D., Performance Evaluation of a Parallel Knowledge-Based System, *Technical Report KSL-89-51,* Stanford Univ. (1989).
[Nakano 88]	Nakano, R., Minami, M. and Delaney, J., Experiments with a Knowledge-Based System on a Multiprocessor, *Technical Report KSL-89-16,* Stanford Univ. (1989) and *Third International Supercomputing Conference,* Information Sciences Institute (1988).
[Nii 89]	Nii, H. P., Aiello, N. and Rice, J., Experiments on Cage and Poligon: Measuring the Performance of Parallel Blackboard Systems, *Technical Report KSL-88-66,* Stanford Univ. (1988) and in Gasser, L. and Huhns, M. N. (Ed.), *Distributed Artificial Intelligence II,* Morgan Kaufmann (1989).

[Noble 88] Noble, A. C. and Rogers, E. C., AIRTAC Path Association: Development of a Knowledge-Based System for a Multiprocessor, *Technical Report KSL-88-41,* Stanford Univ. (1988).

[Rich 79] Rich, E., User Modeling via Stereotypes, *Cognitive Science* 3 (1979) 355-366.

[Rice 89] Rice, J., The Advanced Architectures Project, *Technical Report KSL-88-71*, Stanford Univ. (1989) and *AI Magazine* 10(4) (1989) 26-39.

[Saraiya 89] Saraiya, N. P., Delagi, B. A. and Nishimura, S., Design and Performance Evaluation of a Parallel Report Integration System, *Technical Report KSL-89-16,* Stanford Univ. (1989).

[Steele 84] Steele Jr., G. L., *Common Lisp: The Language,* Digital Press (1984).

[Weinreb 80] Weinreb, D. and Moon, D., Flavors: Message-passing in the Lisp Machine, *AI memo 602,* MIT AI Lab. (1980).

Appendices

A1. Problem Data Profile

Figure A1.1 specifies the data used in the experimental classification system. This figure shows the numbers of total and new problem objects at every data input.

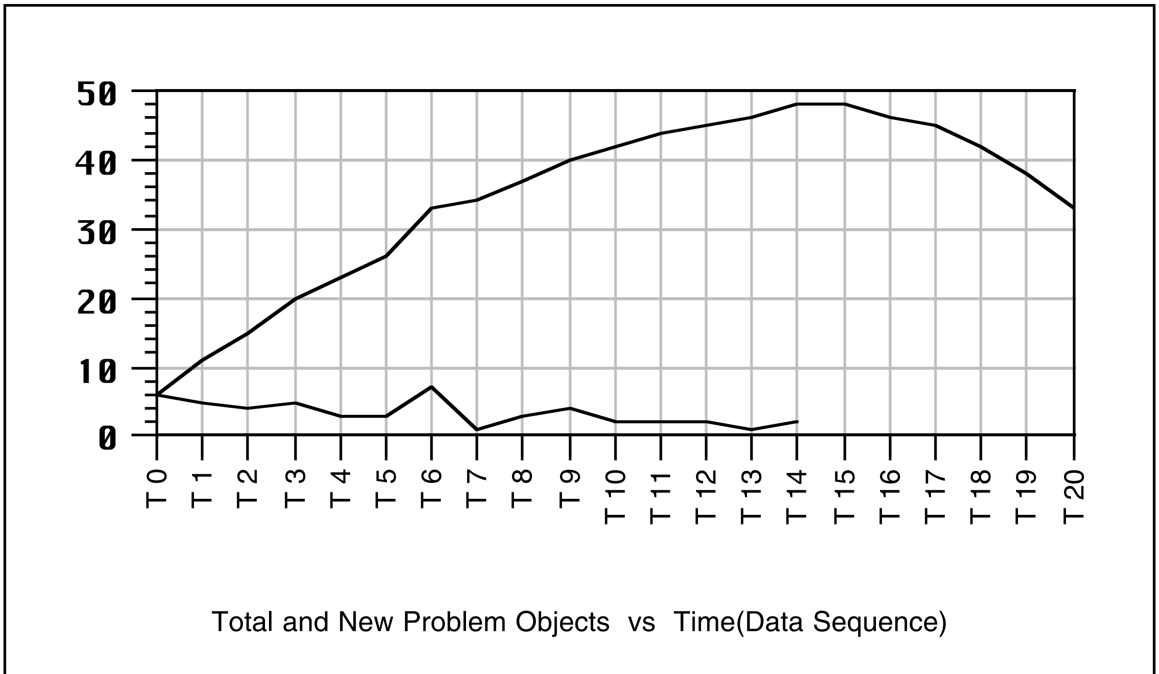

Figure A1.1. Problem Data Profile

A2. Latencies

The following are observed latencies from which we evaluated the sustainable data rates. We observed the latencies of forming initial hypotheses, making other hypotheses, and disproving those hypotheses. We compared the latencies with the same object allocation on each set of processing elements. We denote a processing element a PE and use the legend specified in Figure A2.1 for the latency figures in this section.

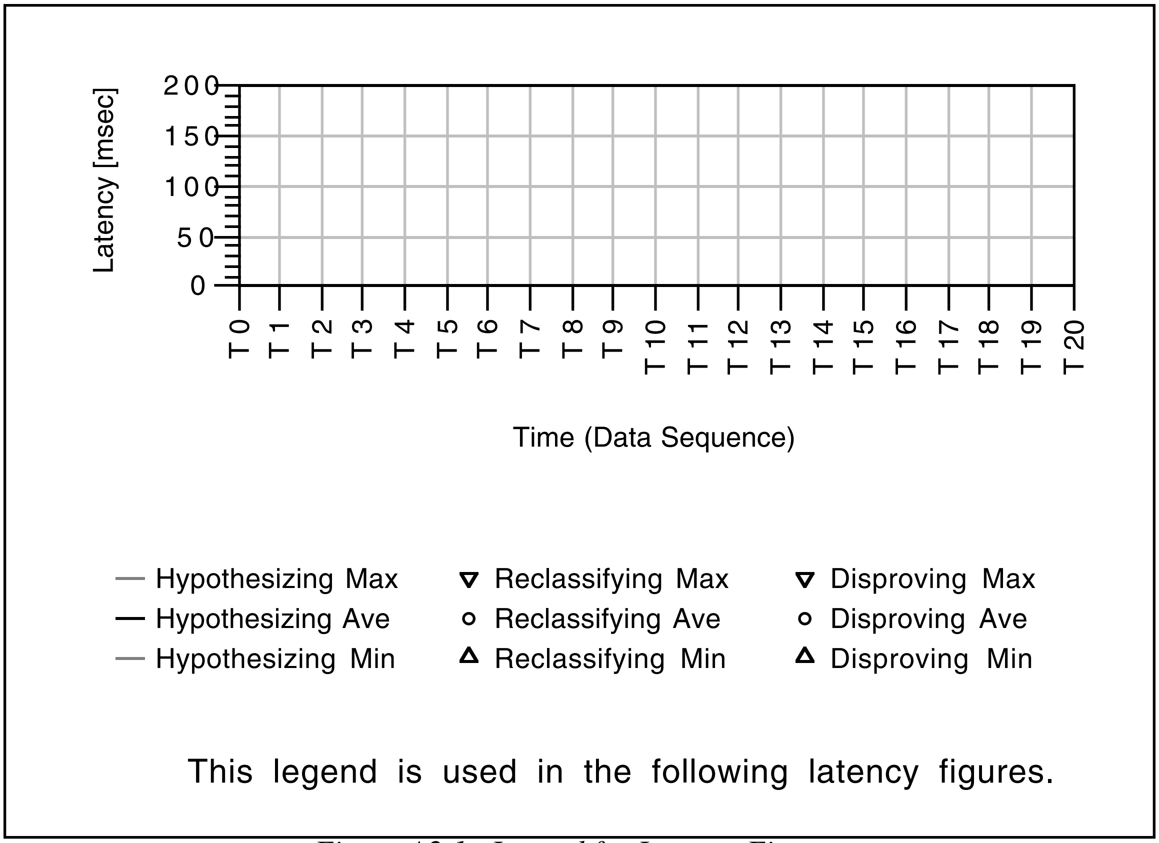

Figure A2.1. Legend for Latency Figures

A2.1. Latencies at Sustainable Data Rates

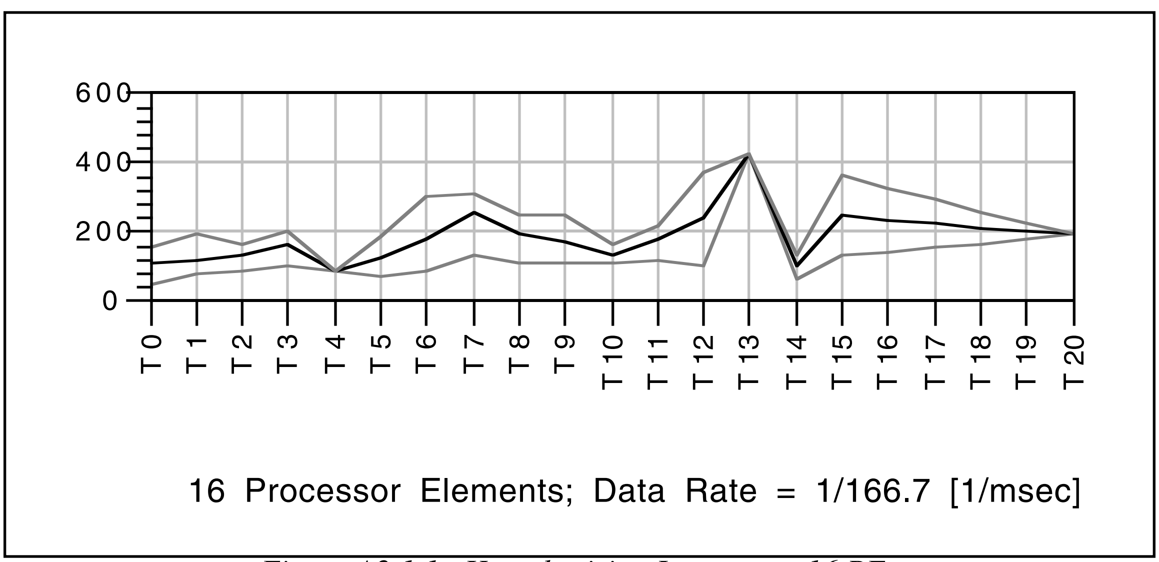

Figure A2.1.1. Hypothesizing Latency on 16 PEs

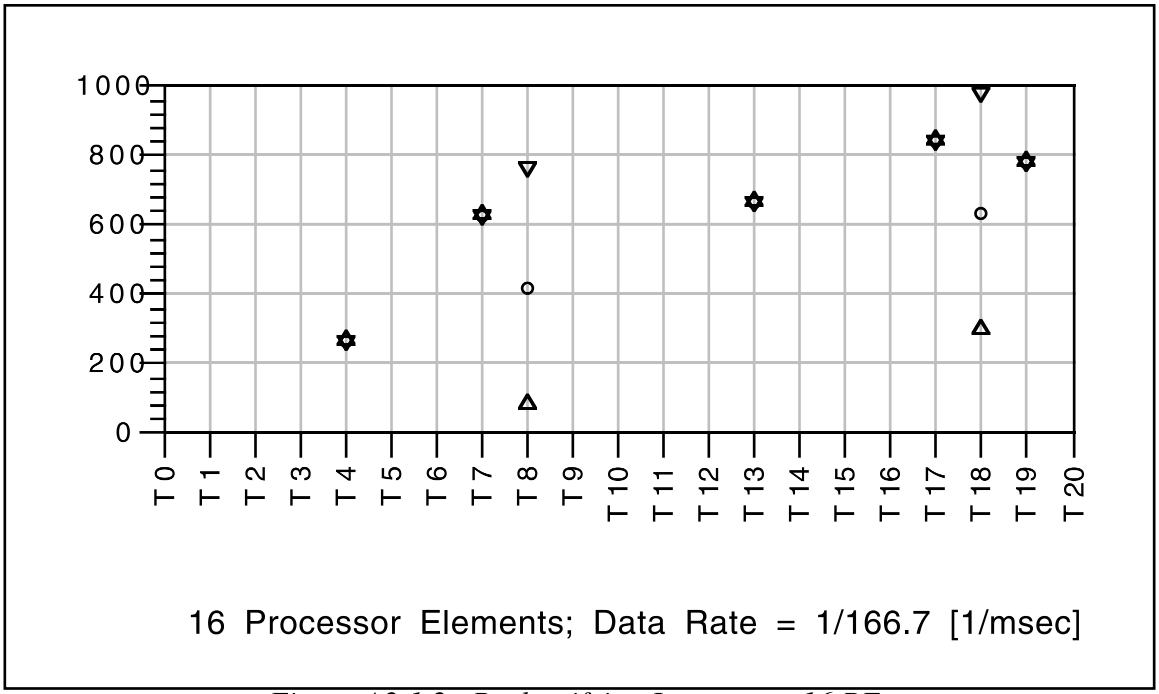

Figure A2.1.2. Reclassifying Latency on 16 PEs

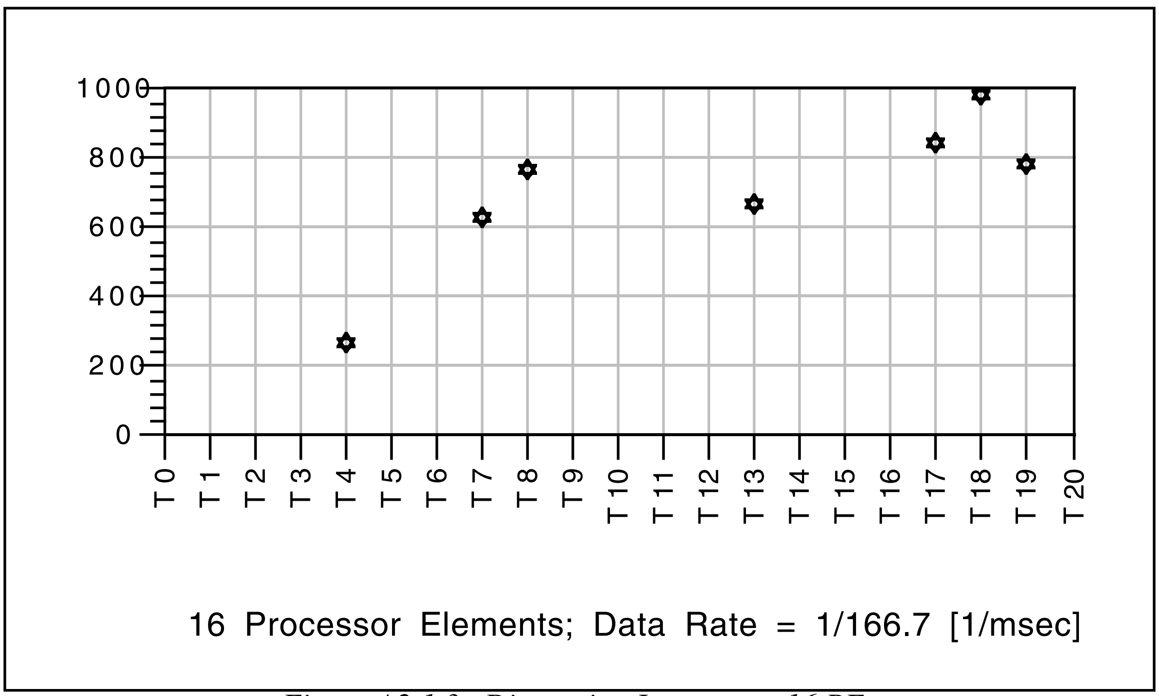

Figure A2.1.3. Disproving Latency on 16 PEs

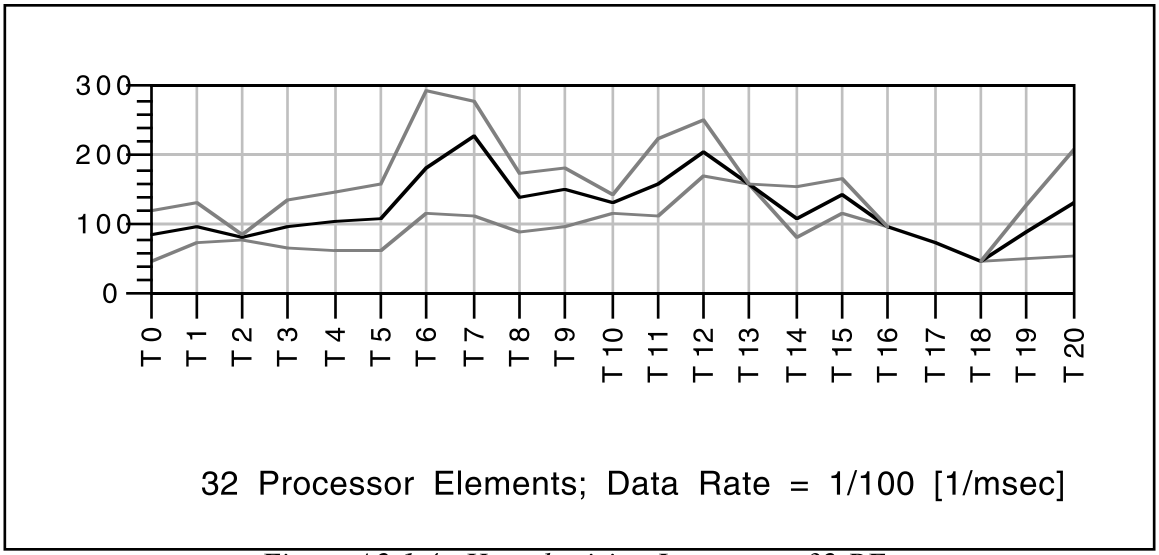

Figure A2.1.4. Hypothesizing Latency on 32 PEs

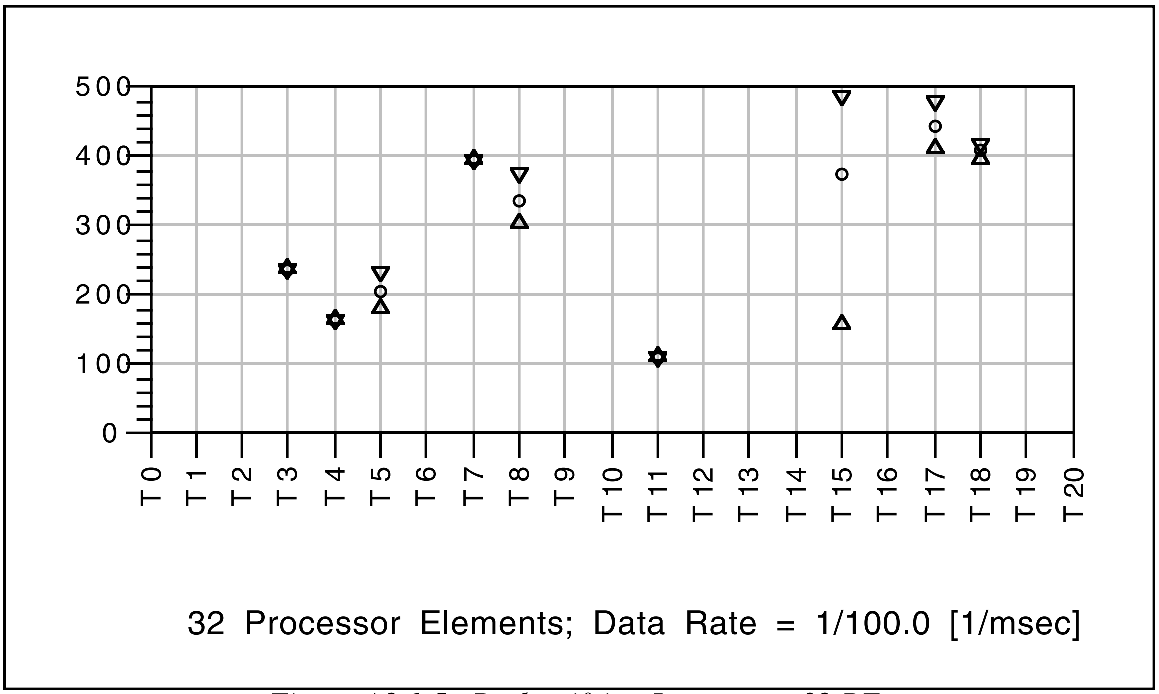

Figure A2.1.5. Reclassifying Latency on 32 PEs

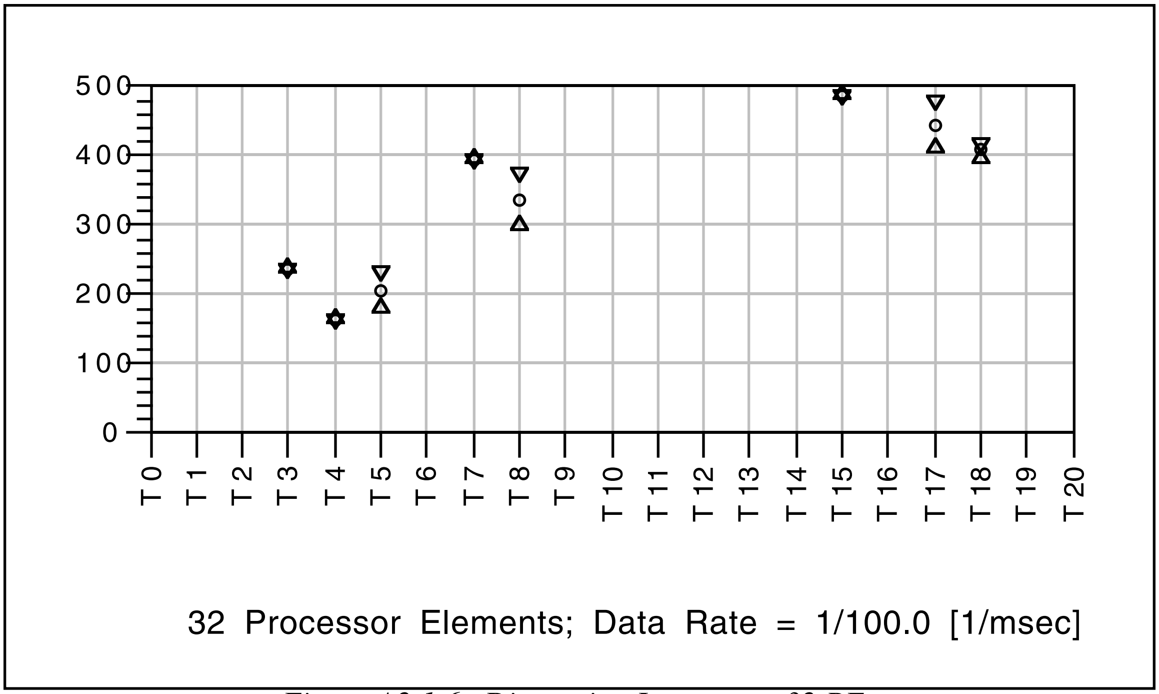

Figure A2.1.6. Disproving Latency on 32 PEs

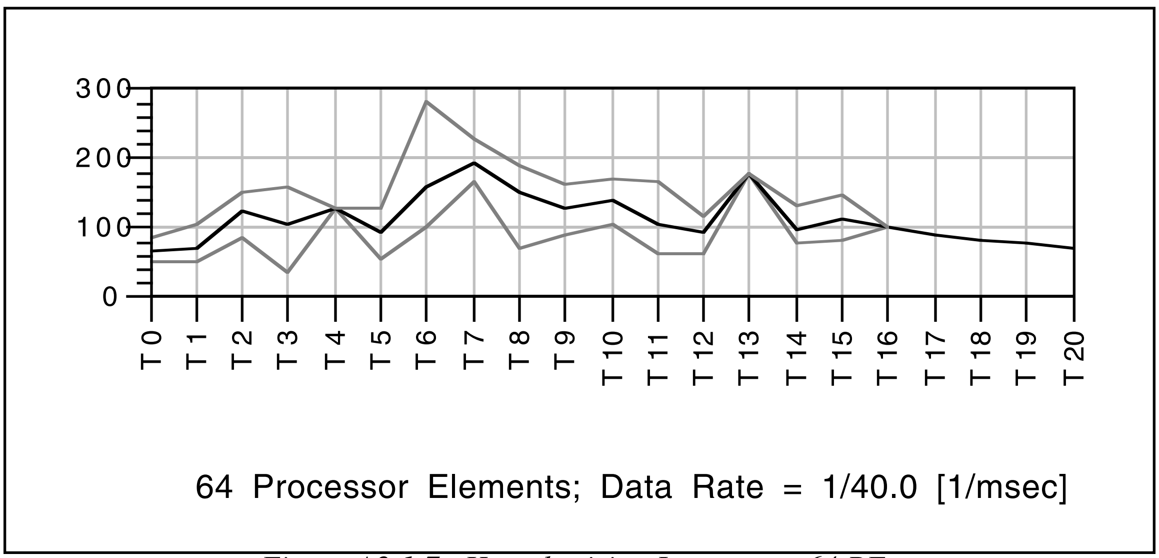

Figure A2.1.7. Hypothesizing Latency on 64 PEs

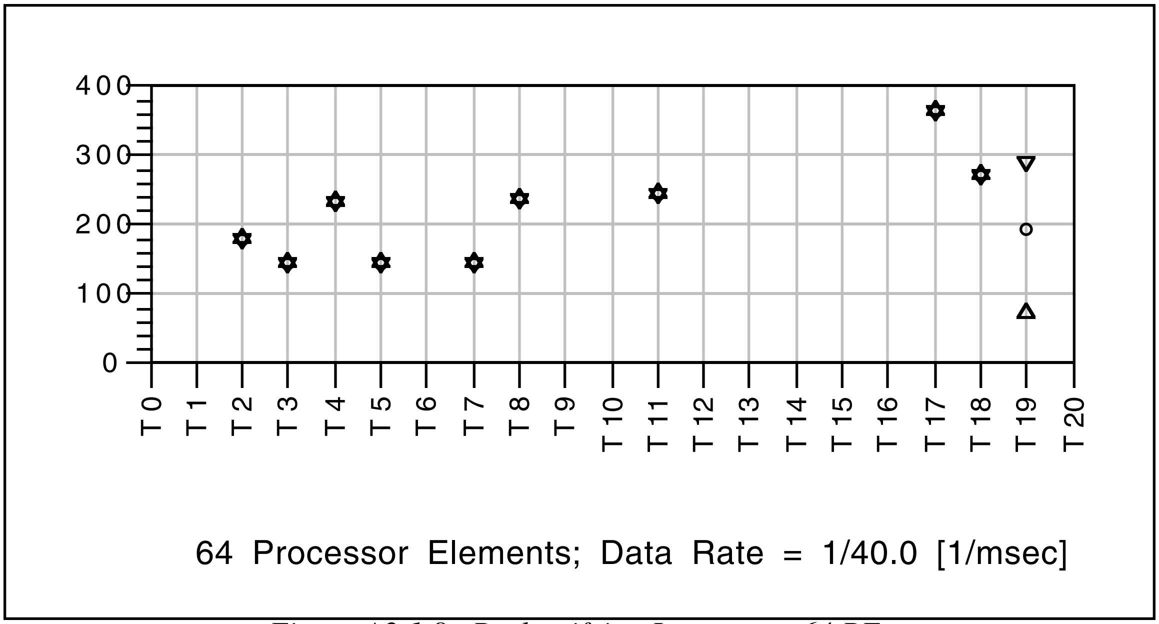

Figure A2.1.8. Reclassifying Latency on 64 PEs

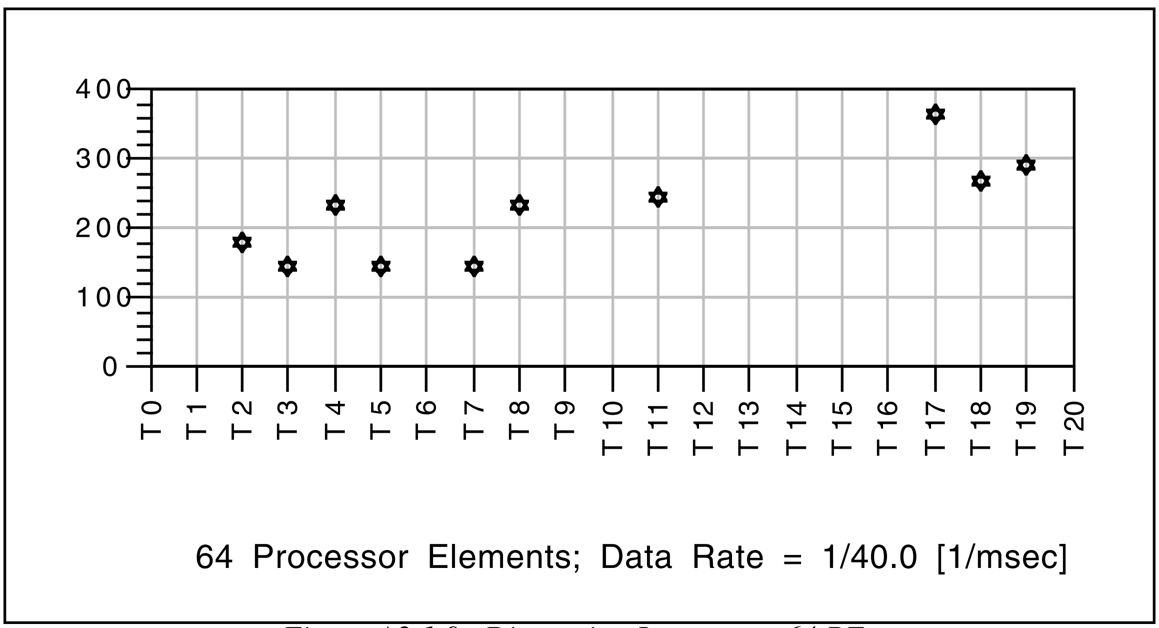

Figure A2.1.9. Disproving Latency on 64 PEs

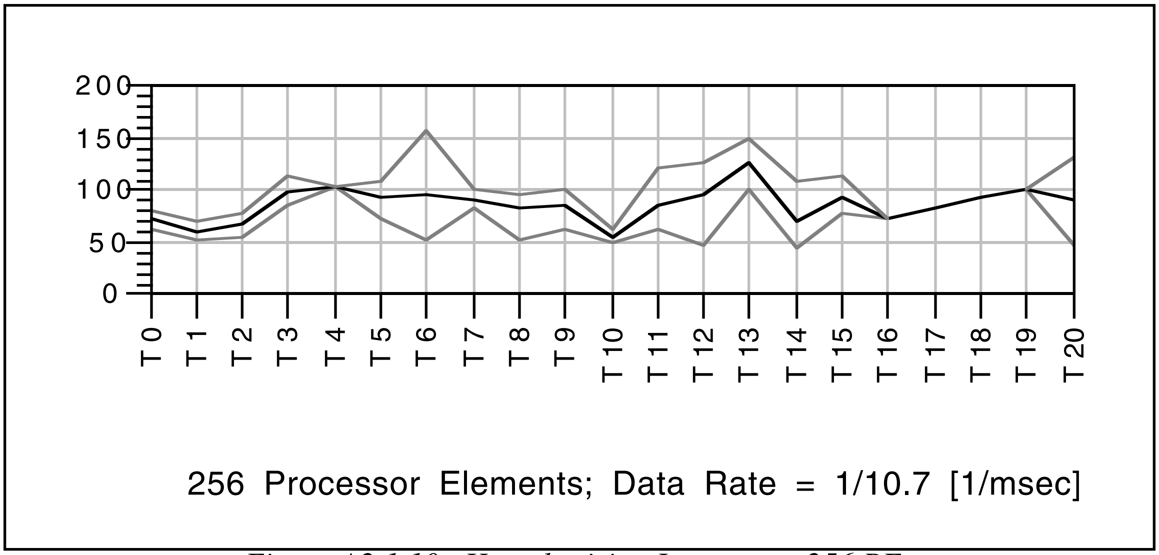

Figure A2.1.10. Hypothesizing Latency on 256 PEs

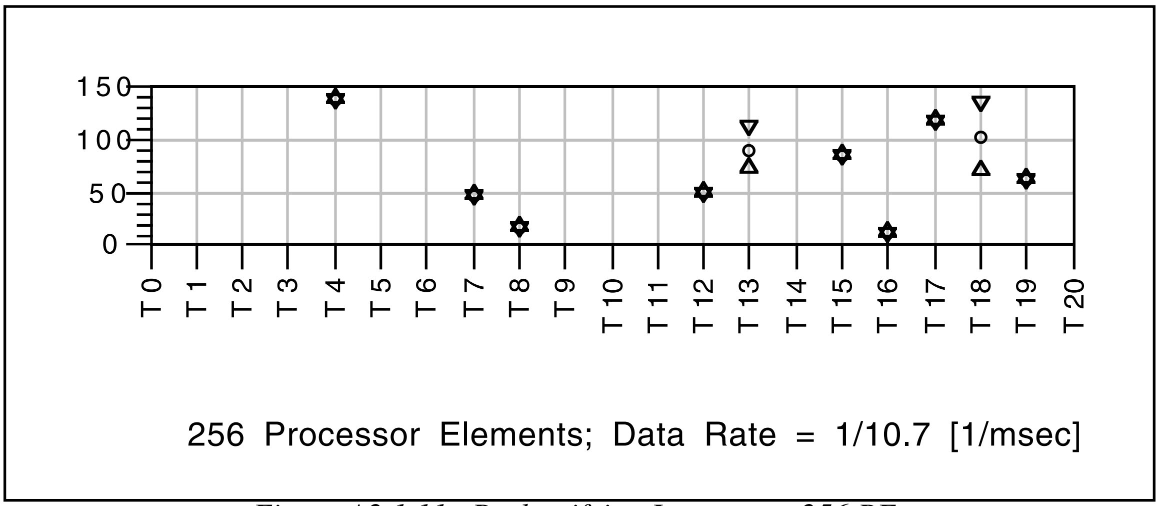

Figure A2.1.11. Reclassifying Latency on 256 PEs

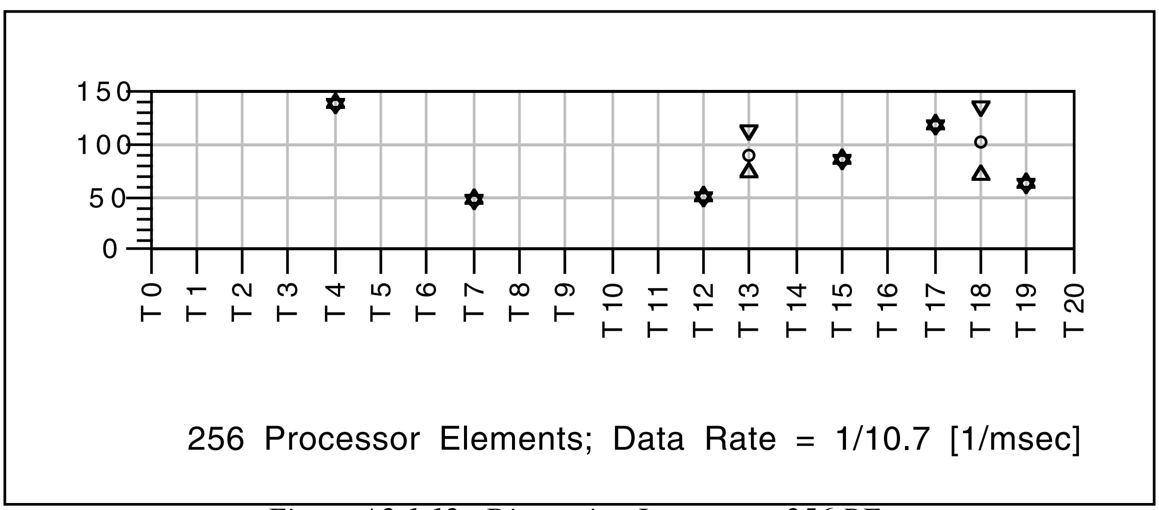

Figure A2.1.12. Disproving Latency on 256 PEs

A2.2. Sample Latencies at Overloaded Data Rates

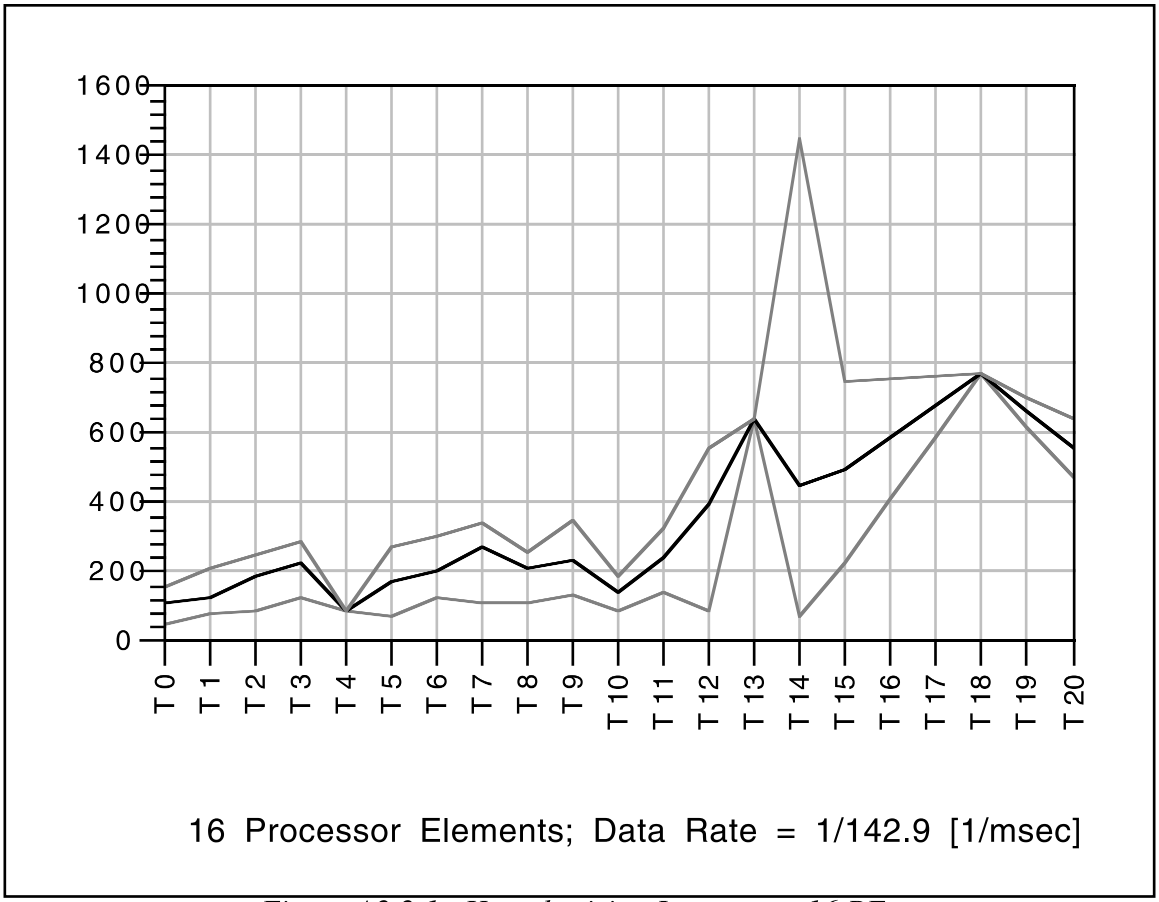

Figure A2.2.1. Hypothesizing Latency on 16 PEs

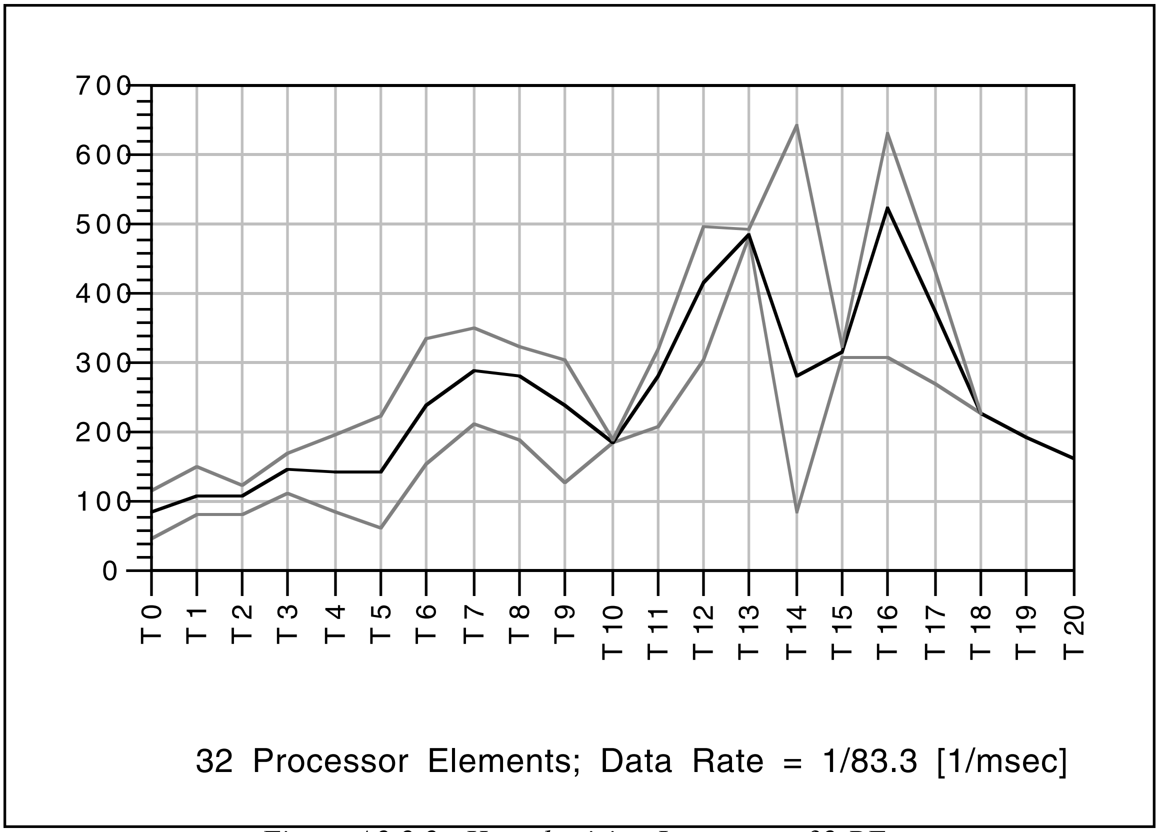

Figure A2.2.2. Hypothesizing Latency on 32 PEs

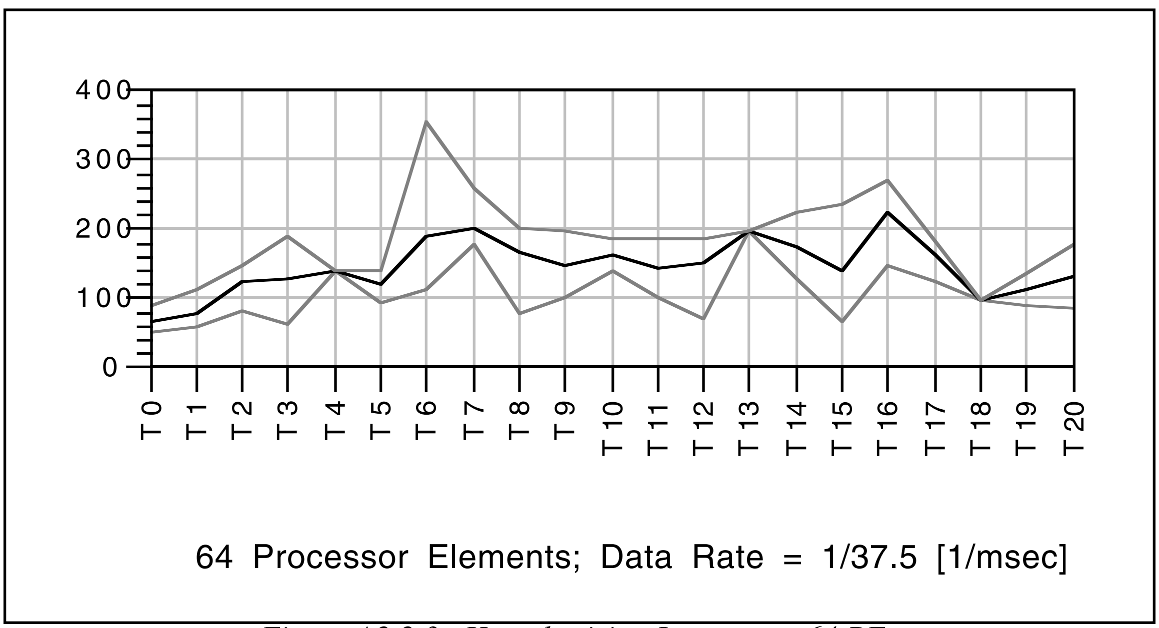

Figure A2.2.3. Hypothesizing Latency on 64 PEs

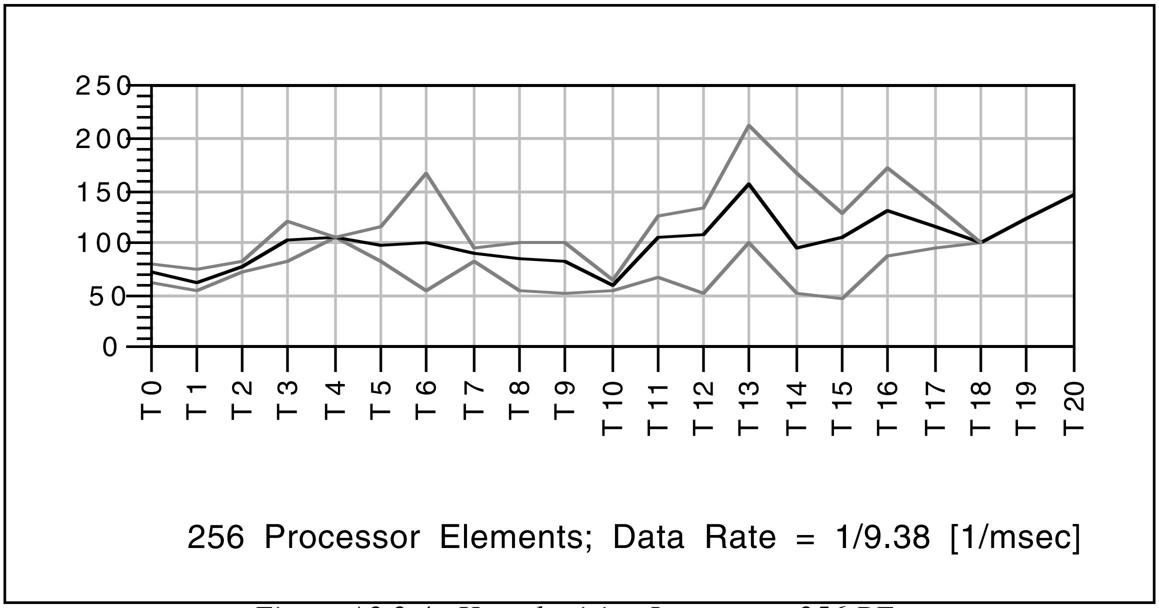

Figure A2.2.4. Hypothesizing Latency on 256 PEs