January 1991

PTHOMAS: An adaptive information retrieval system on the connection machine.

Robert Oddy
Syracuse University

Bhaskaran Balakrishnan
Syracuse University

PTHOMAS: AN ADAPTIVE INFORMATION RETRIEVAL SYSTEM ON THE CONNECTION MACHINE

ROBERT N. ODDY and BHASKARAN BALAKRISHNAN
School of Information Studies, Syracuse University, Syracuse, NY 13210, U.S.A.

Information Processing & Management vol. 27, No.4, pp. 317-335.
Pergamon Press Inc.

Abstract

This paper reports the state of development of **PThomas,** a network based document retrieval system implemented on a massively parallel fine-grained computer, the Connection Machine. The program is written in C*, an enhancement of the C programming language which exploits the parallelism of the Connection Machine. The system is based on Oddy's original **Thomas** program, which was highly parallel in concept, and makes use of the Connection Machine's single instruction multiple data (SIMD) processing capabilities. After an introduction to systems like **Thomas,** and their relationship to spreading activation and neural network models, the current state of **PThomas** is described, including details about the network representation and the parallel operations that are executed during a typical **PThomas** session.

1. INTRODUCTION

Information Retrieval teems with opportunities to exploit parallel computer architectures. No matter what model the system is based upon, data organization functions and retrieval algorithms require processes to be replicated on many thousands of information items. However, it is not at all obvious how parallelism in hardware can best be used. There is a great deal of experience in creating viable implementations on serial computers, and their performance with Boolean or best-match retrieval methods, applied to very large databases, may be hard to beat (Salton & Buckley, 1988a; Stone, 1987; Stanfill et al., 1989). Perhaps it is significant that these methods arose in the milieu of serial computing. Notable among the efforts to exploit highly parallel computers (array processors) are the work of Willett and his colleagues (Pogue & Willett, 1989; Carroll *et al.,* 1988) and Stanfill and Kahle (1986) on best-match approaches using text signatures, and Rasmussen and Willett (1989) on clustering. However, in the work described here, we have begun to use a parallel computer for the retrieval of document citations, not simply to speed up a method, but to explore the conceptual kinship between an unconventional structural model for information retrieval and fine-grained parallelism. The system we shall describe is called **PThomas** (P for parallel and also for Phoenix). It is based on the **Thomas** program, described by Oddy (1975) and Ofori-Dwumfuo (1982a), and is implemented in C* on the Connection Machine (Thinking Machines, Inc.). The original pseudo-code for **Thomas** is highly parallel in that it is expressed in terms of large sets and graph structures, as wholes. Moreover, the approach to information retrieval pursued in the **Thomas** design indicates, *in principle,* a highly parallel implementation, because decisions about whether documents should be retrieved are not made in isolation, but on the basis of a holistic view of their positions in the densely connected structure of literature, terminology, and authors in a domain. In addition, the system's view of the structure changes with every interaction with the user. Hence, a great deal of data has to be taken into account when making each individual retrieval decision.

The problem of precisely retrieving textual documents relevant to a user's information need has proved remarkably resistant to solution, and there are few really general statements that one can make about what will work well. One such statement, based on a large number of diverse experiments, conducted by many different workers (for example, Salton, 1971; Robertson & Sparck Jones, 1976; Vernimb, 1977; Dillon & Desper, 1980; Salton, Fox, & Voorhees, 1985), is that relevance feedback is an effective way of improving retrieval performance. A relevance feedback device is a program that accepts from the user judgments of relevance concerning earlier output, and adjusts the query in an attempt to improve the relevance of subsequent output. With a system that can adapt its searching to the user's judgments of its earlier output, the accuracy of the initial query is not so critical. This is why relevance feedback appears to be successful in experimental systems, regardless of the underlying retrieval model. We would go beyond this, and conjecture that *adaptability* in an information retrieval system is a factor that dominates most, if not all, other factors usually thought to contribute towards effectiveness from the users' perspective. The reason for our renewed interest in **Thomas** is its high potential for adaptability.

The **Thomas** system is an extension of the relevance feedback notion. Simple relevance feedback is usually seen as a way to refine a query so that it optimally (though not perfectly) represents a presumably static information need. On the other hand, **Thomas** builds and continually adjusts an image of the user's interest and, since the feedback loop is very tight, has the potential to follow a moving target. In **Thomas,** feedback is not an enhancement, it is the central and essential feature of the design. As a result, although the original system was restricted to a limited type of information (indexed, bibliographic records, even without abstracts), the following desirable features were nevertheless present to some degree:

1. It is not necessary for a user to articulate an information need with precision.
2. Emphasis is placed on *recognition* of useful or interesting features of information items, rather than specification of them.
3. The user language can therefore be very simple. (At the same time, the users can take as much, or as little, initiative as they wish.)
4. The system can continuously adjust its behavior in the light of the user's reactions to its outputs.
5. On the other hand, the system is not unduly disturbed by small errors of judgment on the part of the user. This is because (i) the user-image encompasses both the focus (or foci) of the search and a wider context, and (ii) the interpretation of the user's messages is always tentative.
6. The system design philosophy acknowledges that users are in problem-solving situations, and may change their minds about what is relevant, or may shift the focus of their searches. The system can take remedial action when it becomes clear that its user-image is seriously out of line with reality.

2. THOMAS

The first implementation of **Thomas** was a small-scale prototype programmed in PL360 for an IBM 360/67, and is described in Oddy (1975, and 1977a,b). It had a straightforward, easily learned user interface, appropriate to the teletype-mode terminals available. Experiments were conducted with this prototype by simulating dialogues, and the system was "played with" by a few actual users, although their behavior was not observed in any formal way. Oddy (1977a) describes how the system appeared to the user. We will describe it as briefly as possible here. The experience is more like browsing than planned and formulated searching. The user begins by entering one or more words, phrases, or names, without relating them to each other as one must in a Boolean system. The system's immediate response is a display containing a document citation, and a numbered list of authors and terms, which can be used by the user like a menu. (On occasion-rarely at the beginning of a dialogue, slightly more frequently towards the end-the system may respond with a display of associated terms.) The user is expected to react to the display, with a message in the following general form:

[Yes 1 No][menu-item-number . .]
[NOT menu-item-number. . .] [phrase. . . 1.

An example of one interaction is given in Fig. 1. "Yes" or "No" refer to the citation itself:
Does it look relevant or not. Menu-item-numbers that appear before the word NOT, identify descriptive elements of interest, and those after NOT are taken to be not of interest.

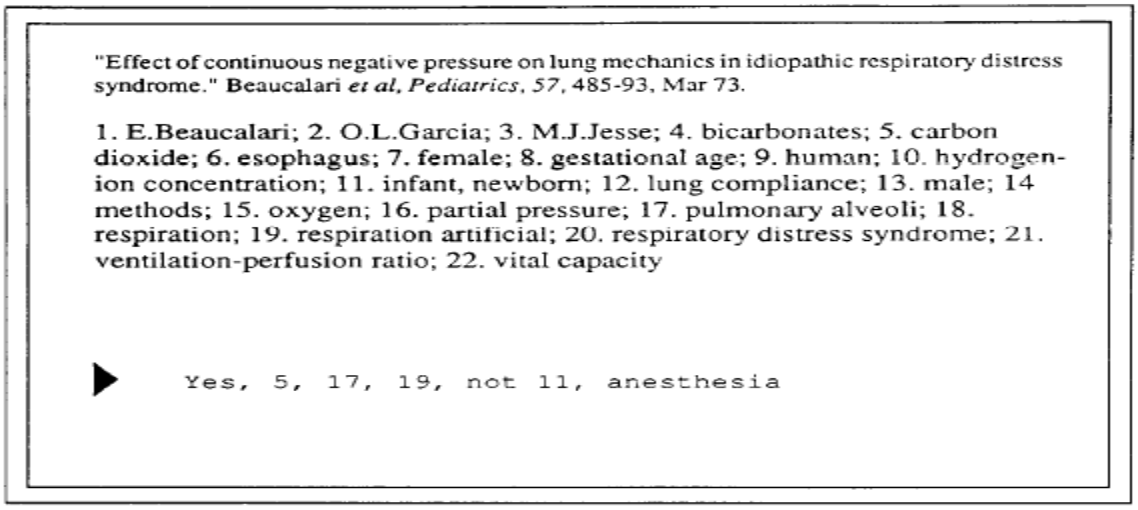

Fig. 1. Example of a displayed citation from Thomas, and the user's response.

New phrases of interest can be introduced by the user at any time, simply by appending them to a message. (The user's initial message is, in fact, a special case of the general message form.) All parts of the user's message are optional. In other words, the user is not obliged to react definitely to anything displayed by the system, which is in keeping with the view that information needs are hard to state. The system then responds with another citation and its descriptive elements; and so the dialogue proceeds. This completes the definition of the user language.

In situations where the user's recent reactions have not been very encouraging, the system may point this out, and display again a citation from earlier in the dialogue (or perhaps display some terms) in an attempt to re-establish an older, more successful context, or find new avenues.

So much for the user's perception of the system. The basis of the system's responses to the user is a continuously adjusted image of the area of interest to the user, as revealed by the user's messages. Feedback is immediate at each interaction, and the state of the image determines the next system output. The **Thomas** system follows a cognitive approach to communication, similar to that discussed by Hollnagel (1979), in that communication is mediated through the participants' models of each other, and of the world. As in Hollnagel's schema, this system models its user in terms of its own world model. This system structure can be expressed by the following high-level pseudo-code:

```
SET UP MODEL (image);
repeat
    message ← GET USER MESSAGE (image);
    INFLUENCE STATE OF MODEL (image, message);
    RESPOND TO USER (image);
until STOPPING CONDITION.
```

SET UP MODEL initializes the image of the present user. This is where prior knowledge of the individual's interests could be introduced, though both the original and more recent versions of **Thomas** can only begin an interaction with a clean slate. STOPPING CONDITION is simply a way of agreeing with the user that a search can finish. It is now necessary to say something about the representation of the user-image. The user is imagined to occupy some part of the knowledge space of the system. Thus, the image is composed of a subset of the system's knowledge about documents and some information, gathered during the dialogue (by INFLUENCE STATE OF MODEL), about the user's relationship to the database. **Thomas'** knowledge is derived from the data commonly available in bibliographic files, namely indexed citations (in Oddy, 1975, the indexing was the controlled, manually assigned, MeSH terms in a sample of the Medlars file). The form of the **Thomas** database is a network of documents, authors, and terms. The edges (or arcs) are typeless, and represent associations explicit in the index files and thesaurus.

The major component of the user-image is called the *context graph,* and is defined (dynamically) by a subset of nodes of the global network. The edges in the *context graph* are just those inherited from the global network when both ends points are included in the *contextgraph.* Other components of the image are:

1. *inhibited* nodes: a set of nodes rejected by the user;
2. *explicit* nodes: a set of nodes explicitly selected by the user;
3. *good* documents: a set of document nodes that have drawn a positive response from the user;
4. *accepted* documents: a set of document nodes that have been tacitly accepted *(i.e.*, not rejected);
5. *reviewed* nodes: a set of nodes (usually documents) that the user has been asked to evaluate for a second time;
6. *last-selected* nodes: a set of nodes explicitly or implicitly selected by the user in the last iteration of the dialogue;
7. *performance:* a number reflecting the recent history of user reactions.

An interpreted user's message has four parts:

- *reaction:* to the document displayed, as a whole;
- *selections:* a set of nodes selected from the display;
- *rejections:* a set of nodes rejected from the display;
- *requests:* new words, phrases, and names introduced.

The sets *selections and rejections* are derived from the message using certain simple, sensible assumptions about implicit choices. For example, if the user types "Yes" without mentioning any of the numbered items, **Thomas** puts all items in *selections.* If the message is "Yes" followed by some item numbers, then those items are regarded as selected while no assumptions are made about any other items in the display.

INFLUENCE STATE OF MODEL uses the interpreted user's message to update the various components of the user image. Nodes in the global network corresponding to the *requests* are found (the relationship is not assumed to be bi-unique), and adds these and also the *selections* to the *context graph.* Nodes added in this way are also removed from the *inhibited* set. *Rejections* are included in the *inhibited* set and excluded from the *contextgraph.* One way of extending the model is the introduction into the context graph of nodes connected to those that the user has chosen. In the case of nodes matching *requests,* this process is unrestricted, but in the case of *selections,* which represent relatively passive choices, only document nodes will be included. In no case will *inhibited* nodes be introduced in this way-hence these act as walls in the network.

A display contains a central node (usually a document node) followed by a list of nodes connected to the central one in the global network. Thus, nodes not currently in the context*graph* may be included. Their selection by the user is another way of extending the image, and conversely growth in certain directions will be inhibited if the user rejects nodes. Yet another heuristic for extending the context graph in a potentially useful way is to try to link up disconnected components, if such there be. Short paths are included from the global network if they can be found and are not blocked by inhibited nodes.

Having manipulated the image, **Thomas** uses its new state to determine the next display. The strategy depends on recent performance, and certain gross characteristics of the *context graph* (Is it connected? Are there any document nodes that the user has not yet seen?). In most circumstances, a document node is chosen from the context *graph* using a so-called *involvement measure,* which is designed to indicate the node's relative centrality to the image. It is the ratio of the number of edges incident with the node in the con*textgraph* to those incident with it in the global network. If *performance* is acceptable and the *context graph* is in one piece, the most central document will be chosen; otherwise a less central document may be displayed. As we have mentioned, all nodes linked to the document in the global network (i.e., authors and terms) are displayed with it.

From the technical point of view, the processing involved in **Thomas** includes:

1. Translation of words, phrases, and names typed by the user into node identifiers in the database-a fairly standard index search problem;

2. Maintaining sets of node identifiers, ranging in size from quite small (e.g., good documents, *selections)* to very large and volatile *(context graph).*
3. Finding all the nodes that are linked to any node in a given set, possibly under some constraint.
4. Computing an involvement measure for every one of a possibly large subset of the nodes in the *context graph.*
5. Determining the connected components in the *context graph,* and looking for short paths in the global network to join them.

For typical sequential computers some of these are lengthy processes. In the interests of building to prototype (in the early 1970s) that responded quickly to its user, a design goal of limiting the growth of the *context graph* became more important than it should, and the trial database was smaller than desirable for convincing experiments.

The tests done by Oddy indicated that, when the **Thomas** system was used to retrieve the same number of relevant documents as a Medlars boolean system *(i.e.,* was operating at the same recall level), its precision was not significantly different. However, the **Thomas** system was much easier to use (no explicit query formulation being necessary). According to a measure of effort based on the number of tokens of various types entered by the user, the effort expended by the **Thomas** user was estimated to be one third of that needed for the boolean system.

3. THOMAS II

Ofori-Dwumfuo (1982a,b, 1984) evaluated some modifications to **Thomas.** His version of the program, **Thomas II,** differed in a number of ways from Oddy's. First, it had no user interface, but a dialogue simulation package, in which information about actual searches was used to automatically generate user responses so that experiments could be run in batch mode. **Thomas II** allows for weighted edges in the network database. This has a number of implications for processing, the major one being that the involvement measure used to select nodes for display is computed from the sums of weights on incident edges instead of by simply counting them. To generate weights, Ofori-Dwumfuo computed a new network from the original unweighted one. The weight of the edge between two nodes is inversely related to the degrees of the nodes and the length of the shortest path between them. The motivation for this choice of weighting scheme was that one could, by following a single weighted link, approximate the behavior of a less constrained spreading activation in the network to find nodes associated with some of those already in the context graph. In an experiment, using the same test collection as Oddy had, Ofori-Dwumfuo observed a reduction in the measure of user effort of about 20%) when the weighted network was employed.

Other topics that Ofori-Dwumfuo investigated were the use of pre-established clusters in the global network, and the deletion of edges between term nodes that had been derived from the manually constructed thesaurus (MeSH in this case). In the cluster experiment, he divided the test collection into a few large disjoint clusters, and restricted the user-image construction to those clusters that contained nodes *explicitly* mentioned by the user (passive selection was not sufficient). Neither of these treatments had a significant effect on system performance. The experiments are difficult to interpret, however, partly because the interaction between the weighting scheme and the clustering and thesaurus were not explored in depth, and partly because the test collection was so small. These are questions in which we are very interested.

4. RELATED WORK-NETWORK REPRESENTATIONS IN IR

In the years since **Thomas** was first implemented, interest in knowledge representations, including networks, has grown in the information retrieval research community. Before describing our recent work on the parallel computer, we will briefly try to relate **Thomas** to other network-oriented information retrieval research. All notions of association in information retrieval imply the existence of a network structure; however, we will restrict our attention to those systems in which processing is explicitly conceptualized in network terms. We will further limit ourselves to models in which networks are used to represent the whole database, as opposed to the internal structure of individual documents.

The idea that the network structure of a database, made visible and manipulable, will help a user explore it has been discussed for many years (Bush, 1945; Doyle, 1961) and implemented in several systems in various forms, for example: BROWSE (Palay & Fox, 1981), 13R (Croft & Thompson, 1987), Hypertext (see the collection of articles in JASIS, May 1989). In these systems, users are encouraged to imagine themselves *in the network,* at a particular node, and able to step from node to node along the links. Various display devices are usually provided to help the user appreciate the environment, and make choices. The network is not used in this way in **Thomas,** because a goal was to make more holistic use of the structure in modeling the user, and the user's exposure to the network would

have quickly brought about information overload. **Thomas** has much more in common with a class of network processing techniques called *spreading activation.* Examples of work of this type in the information retrieval area are the Online Associative Query System (OAQS) of Preece (1981), the GRANT system (Cohen & Kjeldsen, 1987; Croft, Lucia & Cohen, 1988), and Salton & Buckley (1988b). To begin the retrieval process, in these systems, nodes corresponding to features of the query (terms, for instance) are made active. Then other nodes directly linked to them in the network receive impulses and may become active, and so on. After a certain limited number of cycles, nodes representing documents are examined, and those with the higher levels of activation are retrieved for the user. In a well-connected network, such as we find in information retrieval databases, activation can rapidly get out of hand, spreading over large portions of the database indiscriminately, so constraints are needed. Many methods have been employed: The number of activation cycles is usually severely limited: the strength of impulses leaving active nodes may decay with each cycle; that strength may be distributed between all outward links; and if links are of various types, there may be rules governing their selection in certain situations. Some of these techniques bear a resemblance to aspects of the psychological theory of spreading activation expounded by Collins & Loftus (1973, but the parameters of the models cited have usually been chosen in the light of experience with more conventional information retrieval research, and less frequently for psychological plausibility. In fact, Preece systematically demonstrates that many well-known information retrieval methods can be expressed in terms of a spreading activation model, by suitable choice of network parameters. At one fairly obvious level, spreading activation in **Thomas** is constrained by its procedures for building and maintaining the context graph which might be regarded as its set of active nodes. The growth of this graph is usually limited to one step from each selected node in each dialogue iteration. Also, regions within it can be pruned away (activation falls to zero). The strategy of trying to join disconnected components of the context graph by bringing in a path from the global network resembles a spreading activation process described by Quillian (1968) to find the conceptual connection between two words in semantic memory, but is applied to larger sub-structures.

The distinction between spreading activation (SA) and *neural network (NN)* or connectionist models (Rumelhart & McClelland, 1986) is not clear; indeed, spreading activation is a central component of any neural network. Features that tend to distinguish them are:

- Activation is allowed to continue for many more cycles in a neural network than in an SA model. Termination conditions can be complex, involving measurement of the stability of the network state, or some function of the overall activation level of the network.
- Representations in NNs are often thought of as distributed (concepts correspond to whole patterns of activity in regions of the network), whereas they are usually local in SA models (a node stands for a concept or object).
- Links in an NN model have numerical weights but no type, whereas SA models often assign category labels to the links (e.g., ISA, CAUSES).
- Unlike SA models, NNs typically have learning procedures, which bring about changes in the numerical parameters of the links and/or nodes, to improve the network's responses.

Belew (1986; 1989) and Kwok (1989) have described applications of neural network methods to document retrieval. In both cases, representations are essentially local (nodes represent documents, terms, or authors), but learning procedures are discussed, whereby relevance judgments are used to adjust the association weights and hopefully improve performance for future queries. Adaptation to an individual user, on the fly, is not yet incorporated in these models, although Belew mentions that the relevance judgments collected for longer term learning can be used to reactivate the network for a second round of searching. In contrast **Thomas** elicits frequent feedback from the user and responds (i.e., its model changes) instantly, but so far it has no long-term learning capability. Connectionist learning methods are slow. A large number of parameters must be adjusted, and only by small amounts with each training event (for stability). They are inappropriate for fast learning within the space of a single search. The learning rate also has implications for choice of representational level; at present, it is probably necessary to endow the system with high-level knowledge, through a local representation, even though it has all the problems of conventional indexing. We should remark that we are not comfortable categorizing the representation scheme of these models (and of **Thomas)** simply as *local.* The senses of terms and other node labels are dependent in these kinds of distributed processing on the structure as a whole (and in **Thomas** as modified by the context graph).

There are some approximate equivalences between **Thomas** processing and NN processing. We have already mentioned the simple, limited spreading activation processes of **Thomas,** namely moving nodes in and out of the

context graph. The way in which this system employs the user-image to determine its response is also related to spreading activation. If we regard a node's involvement measure (ratio of connectivity within the context graph to global connectivity) as an activation level, a similar value could be obtained by allowing many cycles of spreading activation to occur, within the context graph. One can visualize all nodes within the context graph as being continually active, so that high connectivity within it would lead to high activation levels, and links to regions outside the context graph as functioning like drains on the activation levels of nodes. In addition, inhibited nodes have the effect of reversing the sign of the signals emanating from them *(i.e.*, their outward links become inhibitory). **Thomas** will normally choose the most involved document node as its response to the user, just as a neural network program would choose the node with the highest activation level. However, when things are not going well, **Thomas** has strategies that deliberately ignore the obvious and choose nodes closer to the periphery of the context graph. We are not sure whether there are equivalent behaviors in NN systems.

We conclude that there are strong conceptual ties between **Thomas** and neural network models, but that there are also potentially fruitful areas of complementarity, which will be interesting to explore in more depth.

5. PTHOMAS IMPLEMENTATION

5.1. The Connection Machine

The primary emphasis of our new research endeavour to date has been to successfully implement a working version of **Thomas** on a fine-grained, highly parallel computer, the Connection Machine. By design, the Connection Machine is a Single Instruction, Multiple Data (SIMD) computer. It contains 32 K simple processing elements, each of which is in reality a processor-memory pair. Each processor has 47996 bits of memory available to user programs. The processors are arranged in four modules of 8 K, and a task may attach 1, 2, or 4 of these modules. It is also possible to increase the *apparent* number of processors beyond 32 K by specifying the "virtual processor" option, whereby each physical processor simulates several processors serially. In this case there is, of course, a memory tradeoff.

The Connection Machine is attached to a host machine, and all instructions come from programs run on the host. Every processor in the Connection Machine acts on a single stream of instructions that gets broadcast by microcontrollers. The data that each processor acts upon is usually stored in the memory attached to each processor. Processors may choose not to act on an instruction; this is governed by the setting of a "context bit." This facilitates instructions being executed on a subset of all the processors being used. The processors of the Connection Machine can also communicate with each other through a highly interconnected hypercube. The hypercube allows arbitrary communication among processors. In other words, processing units may access data stored in the local memories of other processing units.

There are several languages available to program the Connection Machine. These include high level languages like *Lisp and C* as well as lower level instruction sets. **PThomas** is implemented in C*, a Thinking Machines Corp. enhancement of the C programming language that facilitates access to the Connection Machine's processors as well as the sending of instructions to the processors. The current version of **PThomas** runs on a Connection Machine Model 2. Some clarification is appropriate here: The data stored in the processors of the Connection Machine are accessed/manipulated by instructions in a C* program. The program itself runs on the front-end. The front-end is an Ultrix machine (henceforth referred to by its node name, **cmx;** likewise, the Connection Machine that is used shall be referred to as CM2). The breakup of responsibilities between these two machines is elaborated on later in this section. The approach we have followed to create **PThomas** is to adapt the pseudocode of **Thomas** and **Thomas II,** especially the latter (Ofori-Dwumfuo, 1982a). In a sense, what we have done is parallelize parts of the original program, rather than starting from scratch. We recognize that this may have implications in terms of CM programming efficiency, but have yet to address the issue. Given below is a description of **PThomas** as it now stands.

5.2. Breakdown of responsibilities between cmx and CM2

From a functional standpoint, **PThomas** performs the same "dialogue loop" that its predecessors did (see the section on **Thomas).** This loop is useful because the delineation of responsibilities between the front-end and the Connection Machine can be defined in terms of this loop. Henceforth, all references to these steps will be in terms of the names of the higher level procedures in **PThomas** (given in boldface, such as: **respond_to_user( ))**
Simply put, most of the CM processing occurs in the INFLUENCE STATE OF MODEL stage of the loop. At a slightly lower level, the responsibilities may be broken up as:

- **cmx:** managing the text database, loading the database into CM2, interfacing with the user, searching for node identifiers corresponding to the user's response, and retrieving text for display.
- CM2: graph processing of the network part of the database/representation.

Figure 2 illustrates the overall architecture of **PThomas.** We dwell in more detail on the layout of the processors in the CM later. In the front end *per se,* DBASE holds information about the database. At present this is loaded from a file called **db2.** Each element of **DBASE** is a structure that contains the nodeid, an integer called **adt** which specifies whether the node is either a document/title, author, or term node, and a string element called **name** which holds the actual node name ("Keller," "hashing," etc. -these are from the examples given below). The array **MSGLIST** is used to store individual tokens of the user's response. This information is processed and passed on to the arrays: **SEL, RN, REQ,** and **REJREQ.** These hold the selections, rejections, requests, and "negative" requests (explained in the section on user interface below), if any, that are found in **MSGLIST.** The information in these arrays is passed on to the network in the CM2 during the **respond_to_user()** phase of the loop. The remaining array of import is LASTDISP which holds the nodes to be displayed to the user in the **respond_to_user()** phase of the loop.

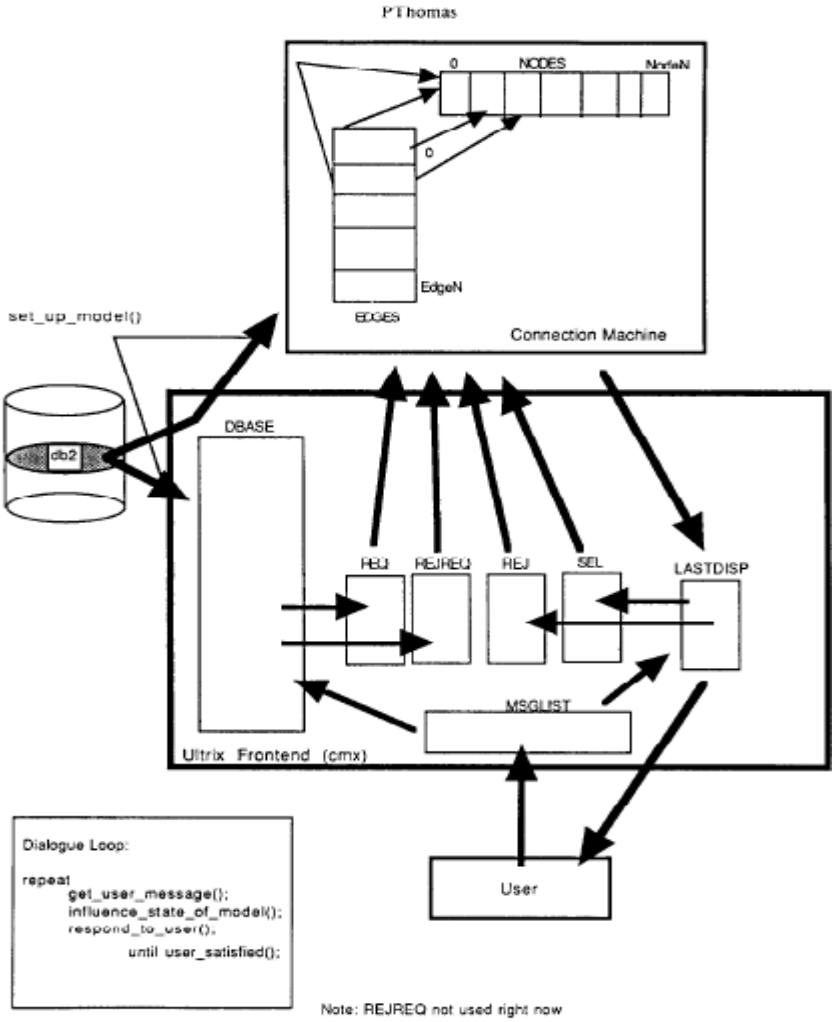

Fig. 2. Overview of PTHOMAS architecture. The bold arrows indicate major flows in the entire dialogue loop.

5.3. The database

The database used to implement and test the working of **PThomas** has been borrowed from Oddy (1975, 1977b). This "toy" database consists of 15 references from the *Communications of the ACM,* Vol. 16, 1973. The database is a network of 86 document (n = 15), author (n = 17), and term (n = 55) nodes with 325 edges among these nodes. While this database is too small to be useful in full scale testing of the system (timing, testing with real users, and the like) its advantages were: (a) it was small enough to be able to check the network status in its entirety during the development phase (this was particularly so when it came to testing the parallel operations on data in the CM); and (b) since the same database had been used by Oddy (1977b) and Ofori-Dwumfuo (1982a), it served as a means of evaluating the "correctness" of **PThomas'** working.

5.4. The user interface

The user interface comes into play during the **get_user_message()** and **respond-touser()** stage of the dialogue loop. Currently, **PThomas** has exactly the same form of interface as **Thomas,** though we do intend to improve this in the near future. Like its predecessor, **PThomas** prompts the user for input; the user in turn is shown some output and asked to evaluate it in some sense. The interface is a separate issue from the parallelism and has not received our attention yet.

5.5. The network in the Connection Machine and parallel operations

It was mentioned before that **PThomas** was implemented in C*, an enhancement of the C programming language. The reader is referred to the Thinking Machines' C* *User's Guide* and *Release Notes* (1988) for details about the language. Given the size of the test database, **PThomas** requires the attachment of only one module of the CM2. This is a 64 x 128 (8192) grid of processors.

C* allows the programmer to define a set of related data items in each of a set of processor memories. This can be seen as an extension of the **struct** construct in C or a **record** in Pascal to the processors in the Connection Machine. Such a grouping is called a **domain,** which really specifies the layout of the memory of a single processor. It is possible to have more than one **domain** in a program-a case in point is **PThomas,** as we shall see below. This means that different CM processors will have different memory layouts. It also means that different processes will get carried out in different processors, because parallel C* code is always local to a specific **domain.** Flexibility in parallelism is possible on the CM by broadcasting an instruction and executing it only on selected processors. These processors execute this instruction on the contents of their individual memories. Interdomain communication is possible and is expressed by means of pointers in C*.

5.5.1. PThomas' CM processor layout. The current version of **PThomas** employs two domains, or sets of processors in the CM2. The first one is for the nodes of the network and the second for the edges. The entire collection of node processors is called NODES, and the set of CM2 processors comprising the edges of the network is called EDGES.* In C*, the specification for the nodes of the network is as follows (see also Fig. 3):

```
domain nodes {
int id;
int cg, inhib, sel, exp, gudd, acptd, redisp;
int comp, wav;
int cursel, assigned;
int invf;
int cif;
float ratio;
}; domain nodes NODES[MAXNODES];
```

This is unlike the layout specified in Oddy and Balakrishnan (1988). There we had specified one domain with a scheme for discriminating nodes in the network from edges. The change is mainly due to the removal of certain limitations in C as well as the migration of PThomas from a Connection Machine, Model 1, to a Model 2.

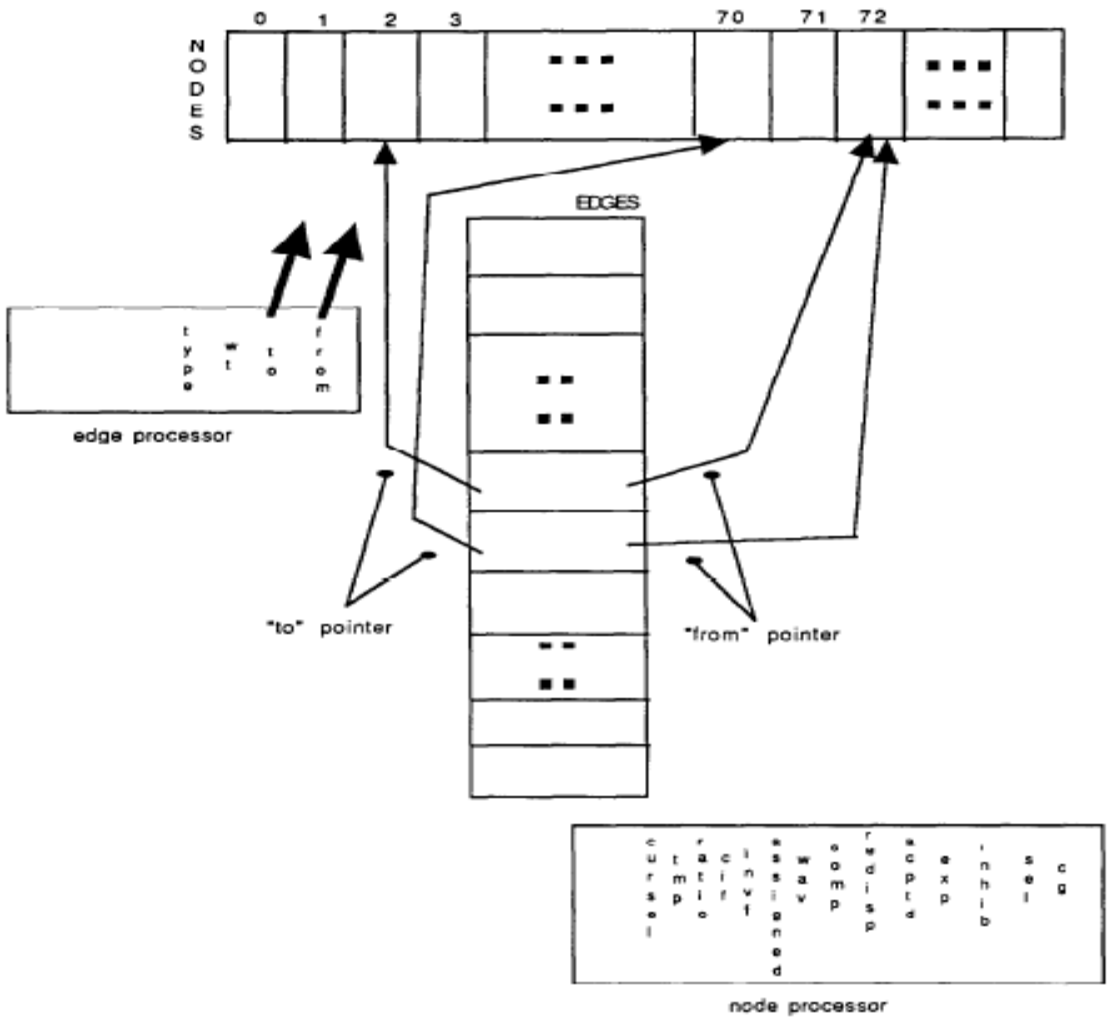

Fig. 3. Illustration of CM EDGES and NODES "arrays" and the edges for the term "proof." It is connected to the term "mathematics" and to ref. 3. Also shown is the processor layout for one edge and one node in the network.

and for the edges:

```
domain edge {
domain nodes *from;
domain nodes *to;
int wt;
int type;
}; domain edge EDGES[MAXEDGES];
```

In plain English, every node (processor) of the network has an identifying number **(id)** and a set of switches (the line beginning "int **cg .** . . ") used to represent the different sets of nodes that a Thomas-like program uses. The variables **invf** and **cif** hold the connectivity of each node in the entire network and the ***context graph,*** respectively. **invf** is calculated once (during **set_up_model()),** whereas **cif** is calculated in every iteration of the dialogue loop, **ratio** is the ratio of **cif** to **invf.** This is used in determining the node with the highest, lowest, or closest-to-average "involvement" during the phase when **PThomas** has to decide which node to pick for display. **camp** and **wav** together are used to determine the connected components of the *context graph* and in determining paths between the

components. The variable **curse1** requires separate mention. In order to maintain the original modularization of **Thomas,** subsets of nodes needed to be retained in jumping from one routine (usually involving parallel operations on nodes) to another. This variable acts as a switch that is set when control is transferred from one procedure to the next **assigned** is used in a similar fashion when determining connected components.

The layout of the **EDGES** processors is less cluttered. Each processor contains two pointers to the nodes at either end of the edge, the **from** and the **to** pointers. Associated with each link is a weight **(wt)** which for the time being has been set to one, as in the original **Thomas.** The variable **type,** is unused at present, but is intended to hold information about the type of edge the processor represents (term-term, document-author, etc.).

Figure 3 depicts the scenario graphically. Also shown are the edges for the term "proof" in the network **(id = 72).** It is connected to two other nodes-document node 3 and the term "mathematics''-in the network **(id = 2** and **id = 70).** Thus, for this node, there are two **from** pointers in **EDGES** that point to it. In exactly this way, there are many situations where there are more than one **from** pointer pointing to a single node processor. While this may seem problematic in general parallel operations with regard to collisions (i.e., many values sent to one storage location), this is not the case with **PThomas,** because operations done in the **edge** domain are either matters of selection (i.e., no collisions) or of setting a Boolean value in **nodes** to which the **edges** point. The latter employs the *parallel* OR operator (I=), which ensures that the recipient variable ORs all the values it receives in what is called an "as if serial" fashion.

A note here will be helpful regarding *purallel reduction operators* on the CM. A. C* expression of the form:

processor_addr ➡ parallel-var op= parallel_value

means that the values on the right-hand side will be sent to processors addressed by the left hand side, and combined, using **op,** with the existing value of **parallel_var.** When multiple values are directed to the same processor, they are combined, as if they had arrived serially, one at a time. However, on the CM, the implementation of such reduction operations is not that naive. They are therefore performed more efficiently than in linear time, by making use of tree-structured and hypercube communication routes between processors.

5.5.2. Parallel operations in PThomas. Having specified the layout of the CM2 processors for **PThomas,** it seems appropriate to give a flavor of the sorts of processing that goes on. Some of these are parallel, and some are not. We shall here consider some parallel operations, providing some background, if need be, and follow that with an illustration of these rather specific processes in the context of a dialogue cycle.

5.5.2.1. *Computing the involvement measures.* During the **set_up_model()** phase, one of the processes is to calculate the connectivity of all nodes in the global network, which is stored in the parallel variable **invf.** The connectivity of a node is simply its out-degree (i.e., the number of edges emanating from it). A pseudo-code expression of the algorithm used is as follows (assuming all the **invf** values are zero):

```
for each edge (in parallel)
        add 1 to invf in its source (from) node
```

The C* code is:

[domain edge] . (from ➡ invf + = (poly) 1;) ;

Every edge processor broadcasts a 1 (the notation for the *parallel* constant 1 is **(poly) 1)** to the **invf** variable that the **from** pointer is pointing to, which it collects (the +=) in an "as if serial" fashion. This example also highlights inter-domain (as also inter-processor) communication, since what is affected is the **invf** variable in all the **nodes.**

The calculation of the connectivity of nodes within the *context graph* (stored in the parallel variable **cif)** is in like vein. However, the restriction is that the nodes at the ends of the edges in the network have to be in the *context graph.* The pseudo-code is as follows (assuming all **cifs** are zero):

for each edge (in parallel)
if both its source node and its destination node are in the context graph
then add 1 to cif in its source node

The following C* code achieves this:

```
[domain edge]·{if ((from→cg) && (to→cg)) from→cif += (poly) 1;};
```

The parallel variable **cg** is used as a Boolean switch to indicate whether a node is in the *context graph* or not. **&&** is the logical and operator in C.

5.5.2.2. *Computing the highest CIF/INVF ratio.* In **respond_to_user( )**, depending upon the current state of **performance, PThomas** may have to identify the node with the highest **cif/invf** value (i.e., highest involvement) among all currently selected nodes in the *context graph* and return its nodeid. The pseudo-code is:

```
for each node which is both in the context graph and currently selected {
        in parallel: ratio = cif/invf;
        max = maximum of all these ratios;
        x = id of any node whose ratio = max
}
```

Here, the variables **max** and **x** are mono (i.e., nonparallel variables). The result is placed in **x**. The **C*** code is:

```
[domain nodes]·{if (cg && cursel){
                        ratio = (float)cif/(float)invf;
                        max>?=ratio;
                        if (ratio==max) x,=id;
                                        }
                }
```

max gets the maximum ()?=) of the ratios in the set of processors selected by the if clause. Since there may be more than one node with the maximum value of **ratio, x** is assigned the nodeid of an arbitrary node whose ratio equals **max** (,=means "take any").

5.5.2.3. *Determining the connected components in the context graph.* This process is invoked when **PThomas** attempts to unify the *context graph* in the **influence-state of_ model ( )** part of the dialogue loop. The function is **connected_comps ( )** and it returns the number of connected components in the *context graph.* Again, partly due to the fact that **PThomas** was developed with the same modularities as **Thomas,** the process is too intricate to be specified in actual code. The procedure is this:

Let all the nodes in the *context graph* be unassigned. Select an arbitrary node in the *context graph.* Make this assigned and assign it to component 1. Then (in parallel) find nodes in the *context graph* that are linked to those in component 1, but are as yet unassigned. Assign these to component 1 and mark them assigned. Repeat these steps until no further nodes qualify. If there are still unassigned nodes, increment the component number and repeat the whole process.

5.5.2.4. *Joining up components of a fragmented context graph.* If **PThomas** determines that the *context graph* is fragmented (i.e., it finds that **numcomps** is greater than one), it attempts to join the fragments by including nodes that meet certain requirements. These are that they are not *inhibited* nodes and not terms with high numbers of postings.

(With the test database there is one token of this type: the term "information storage and retrieval.") The program goes through all the components of the *context graph* in a (serially) pairwise fashion for such nodes meeting these criteria that "connect" the two components. The detection and addition and addition to the *context graph* of such nodes is parallel, however. To make the distinction between what is parallel and what is not, consider the following pseudocode (simplified drastically from **PThomas** code):

```
for i=l to numcomps- 1 (serially)
for j =i+ 1 to numcomps (serially)
in parallel: (find all nodes meeting criteria;
                    add them to context graph;
```

Components of the *context graph* are examined in a sequential fashion. However, each examination could yield (in parallel) several candidate nodes, all of which will be added to the *context graph.*

5.5.3. Parallelism and the user image-An illustration. Here we shall describe the process of adjustment in one iteration of a typical **PThomas** session, highlighting where the parallel operations discussed in the paper come into play. Table 1 shows **PThomas'** user image at a particular point of a user session. This snapshot of the user image is taken immediately after the program has made the necessary adjustments to the user image based solely on the last user input. Given this state of the user image, **PThomas** now has to:

```
determine the connectedness of the context graph

    IF the context graph is fragmented
        discard any 'useless' components
        try and join up the remaining components
        IF successful THEN bring into the context graph appropriate
          document nodes
    ENDIF

respond to the user with this new context graph
```

Table 1. PThomas' user image in this example

Given this state of affairs, it has to "influence" this model and then "respond" to the user.

Nodeid *context graph*	**Name**
09	General performance analysis of key-to-address transformation methods
12	The reallocation of hash coded tables
13	A note on when to chain overflow items in a direct access table
14	Reducing the retrieval time of scatter storage techniques
33	address calculation
58	hashing analysis
65	key-to-address transformation
71	open hashing
74	random access
75	reallocation
Note: the edges of this context graph are shown in Fig. 4a.	
Inhibited	
57	hashing
62	information storage & retrieval
Selected	
75	reallocation
Accepted	
09	General performance analysis of key-to-address transformation methods (also in context graph)

In the first step, **PThomas** examines the context graph (the nodes and associated edges) to determine the connectedness. The function **connected_comps ( ) (see 5.5.2.3)** returns a value of 3, which is held in the global variable **numcomps.** Figure 4a shows some of the values of the parallel variables at the end of this step. For each of the components, the node with the **wav** value of 0 is the one that was arbitrarily chosen. Nodes one hop away are gathered/assigned in parallel. In assigning the nodes of the context graph for Component 2, for example, the nodes **key-to-address transformation** and **hashing analysis** are assigned in parallel. The fact that these nodes have the same **wav** values in that component is indicative of such a parallel assignment.

Having discovered that the context ***graph*** is fragmented, **PThomas goes** into the function **discard_useless_components( ).** A discardable component is defined to be a component whose set cardinality is less than or equal to a parameter **small** (currently set at 2) ***and*** none of whose nodes have been explicitly requested or selected by the user. Determining the cardinality is a parallel reduction operation + =, similar to the one used to computing involvement measures (section 5.5.2.1). Also, for the particular component being evaluated, looking up which nodes have been explicitly requested and those explicitly selected is a parallel operation.

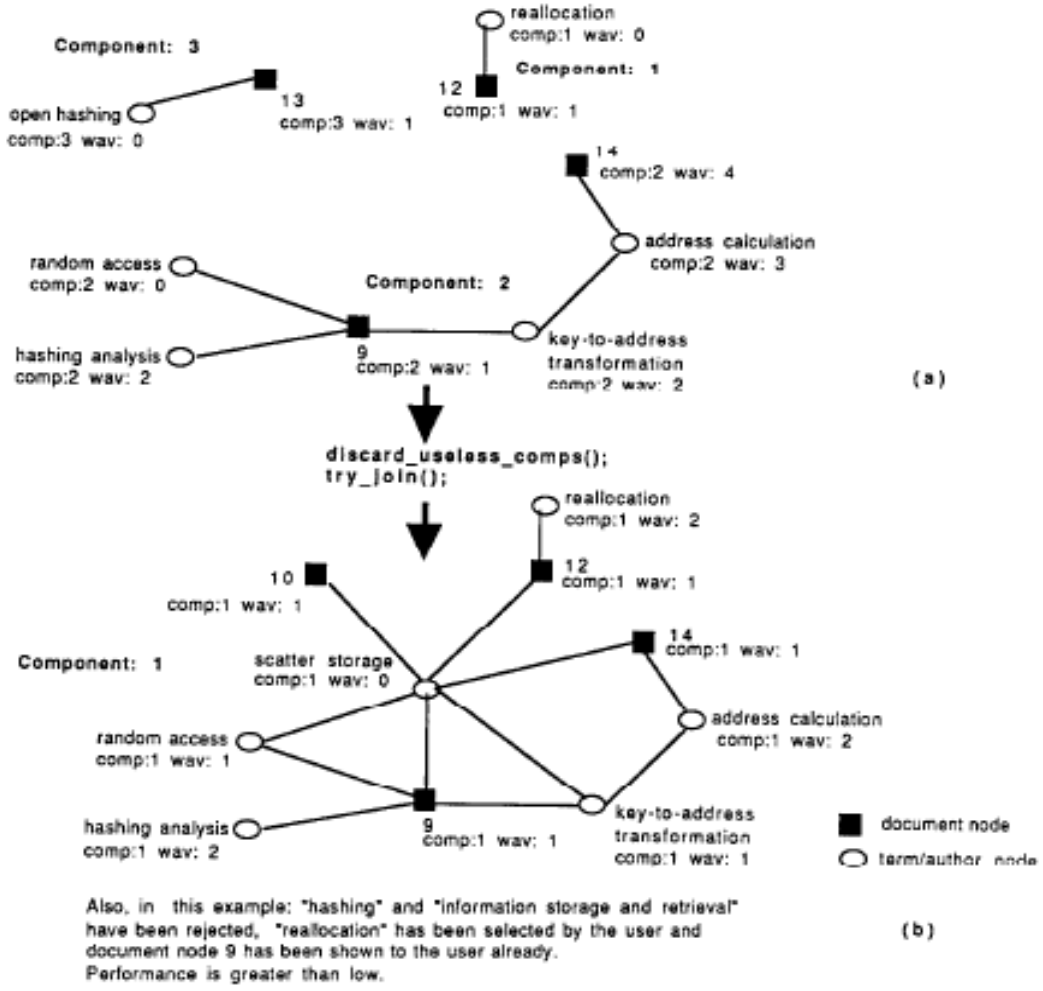

Fig. 4. The transformation of the context graph. Looking at the initial context graph, PThomas determines that it is fragmented and finds that component 3 can be discarded. **Try-join ( )** seeks candidate nodes to join the two remaining components. The node it finds is "scatter storage" which is brought into the context graph. PThomas also brings into the context graph document nodes linked to this term (reference 10). Given a now fully connected context graph and a performance greater than low, PThomas picks an "unseen" reference with the highest involvement factor. This is reference 12.

In this case, while both Component 1 and Component 2 are "small," only Component 2 is a candidate for discarding. The node **reallocation** prevents Component 1 from meeting the criteria because it has been selected by the user (in an earlier part of the dialogue). Removing components from the *context graph* also involves renumbering the components that remain. This too is a parallel operation. All nodes belonging to a component with a **camp** value higher than the **camp** value of the component just discarded decrement their **camp** variable by one, in parallel.

Having discarded Component 3, **PThomas** performs a **try_join( ).** This procedure looks for nodes that can connect any two components of the *context graph.* In this case, the node **scatter storage** is one such node **(hashing** would be too, but for the fact that since the user rejected it previously, it is an **inhibited** node). If **try-join** finds nodes that join the components, it brings these nodes into the *context graph* and all document nodes associated with them as well. In this case, document 10 is brought into the *context graph* (Fig. 4b). Once this is done, **PThomas** again determines the connectedness of the context *graph.* At this stage, **connected_comps()** returns the value 1. It may also be noted in Fig. 4b that the **camp** and **wau** values for the individual nodes of the new *context graph* are different from those in Fig. 4a.

It is now time for **PThomas** to decide what to show to the user. The manner in which this is done is inherited from the original algorithm:

```
IF the context graph is non-empty
        IF (numcomps==l) AND there are unseen documents
                display the most involved of these
        ENDIF
ENDIF
```

Only that portion of the algorithm relevant to this example has been specified here. Another factor important in this decision process is the value of **performance** which reflects how well the interaction has been going thus far. In this case, the value is above (a parameter) low.

The involvement ratios of these as yet unseen documents of the *context graph* are calculated in parallel, as described in 5.5.2.1. These are: 0.142, 0.400, and 0.200 for document nodes 10, 12, and 14, respectively. Document node 9 has an involvement ratio of 0.571, but is not a candidate because it has already been shown to the user. Finding the document node with the maximum involvement ratio is a simple application of the)?= parallel maximum operator (Section 5.5.2.2).

Now that **PThomas** has determined a document node to display, it finds all its associated nodes, composes a display, and shows it to the user. The user is then asked to respond to this display and the cycle repeats itself.

5.6. Evaluation of PThomas

It was mentioned above that one of the primary reasons for choosing a database of such small size was to be able to check the network status in its entirety during the development phase. Feldman points out the differences between programming a connectionist network and traditional computer programming (Feldman el *al.*, 1988, p. 176):

> To follow the progress of sequential computation, one usually needs to look at only a few critical variables and the instruction counter to verify that the program is indeed doing as one expects. This is in contrast to connectionist networks where the sequence of events and their results usually cannot be specified or predicted.

The same may well be said of network databases such as **PThomas.** As it turned out, there is no simple way to detect that the program is doing what it ought to be doing. For example, a minor bug that does not bring into the *context graph* two or three nodes may not be reflected in the references displayed in response to the current user image in test situations. Detecting such flaws in highly parallel code is sometimes very difficult and tedious.

The primary goal we had in mind was to get a working version of **Thomas** on the Connection Machine. Given the problems with testing associative/highly parallel networks, our criterion for testing the "correct" working of **PThomas'** graph processing was: given a state of the user image, the program should not only respond to the user in

the same manner that **Thomas** did, but that it should also generate a sequence of equivalent data structures in the CM processor memories. Rigorous testing, made possible in large part by the choice of database, has proven that this indeed is the case. The outcome is that the higher level processing in **PThomas** (as for example in 5.5.3) shows exact similarity to that of **Thomas.**

6. CONCLUDING REMARKS

We have given some account of our resumption of work on the approach to information retrieval embodied in **Thomas.** In this endeavor, our purposes are two-fold: We wish to investigate the promise offered by new, highly parallel computer architectures as tools for speeding the development of such approaches to information retrieval; and we should like to gain greater understanding of methods of introducing adaptability into information retrieval systems. This paper is but a first step towards each goal.

In pursuit of the first goal, we have completed a prototype version of the system, using the Connection Machine, and have gained some knowledge of the critical problems that need to be resolved if full-scale application is to be achieved. We have identified two substantial problems:

1. Connection Machine size limitations. Our machine has 32 K processors, but a (quite small) database representing 10,000 document abstracts could generate a network with a combined total of about one million nodes and edges. Now, the Connection Machine can be configured with many more *virtual* processors than actual ones, by dividing the processor memories. (There is a time penalty, because each actual processor will simulate several virtual processors serially.) With some fairly obvious storage economies (changing the representation of logical variables to bit fields), we could fit O (100) nodes or edges into each processor. One might also expect that future models of such machines will be much larger. However, we forsee practical problems with very large bibliographic databases for sometime to come, and feel obliged to tackle the network partitioning problem. That is, we need to find a way of dividing the network into sections that can be loaded into the Connection Machine, in the expectation that **PThomas'** processing can continue satisfactorily for an appreciable length of time before needing to swap partitions. Ofori-Dwumfuo (1982a) performed a clustering experiment which indicated, *for a small test collection,* that the network can be partitioned and processing restricted to only those partitions containing nodes explicitly named by the user, without serious degradation of performance. Further work needs to be done with this type of technique.

2. Mapping networks onto the Connection Machine. CM architecture permits communication between any pair of processors, and to some extent this can be done in parallel. However, the CM does not have a direct wire between every pair of processors, and messages are therefore routed through a hypercube. With our chosen network representation, and any others of which we are aware, some messages concurrently traversing an arbitrarily connected network will intersect at junctions in the hypercube, and will therefore be queued and processed serially. Until we have experience with larger, more realistic databases, we will not know whether this will be a serious impediment. (We do not believe that Belew's (1989) comment about the inappropriateness of the CM's SIMD architecture to the MIMD nature of connectionist models applies to **PThomas.)**

We have moved towards our second goal by taking a fresh look at some of the properties of **Thomas,** and by comparing it with other related ideas. The design of **Thomas** was an *ad hoc* response to a consideration of the problem of expressing information needs to a retrieval system. We have found that it is in many ways consonant with other network models, but is also quite distinctive in its behavior. In general, feedback is used frequently, and adaptation takes place at the level of the individual dialogue. We can identify three major levels at which adaptation can take place in an information retrieval system: (a) within the dialogue, when the system learns what will get a positive response from the user in his or her present situation; (b) at the user level, where the system learns how an individual's vocabulary, and viewpoint correlate with document representations; and (c) at the group, or societal level, where such factors as conventional terminology, trends in research interests, and schools of thought are of importance. It seems to us that different learning, or adaptation devices are needed at these different levels, and we hope to learn more about this issue by continuing the development of Thomas-like systems.

Acknowledgement

The authors would like to acknowledge the support of the Northeast Parallel Architectures Center located at Syracuse University.

REFERENCES

Belew, R.K. (1986). *Adaptive information retrieval: Machine learning in associative networks.* Doctoral dissertation, Ann Arbor, MI: University of Michigan. (U.M.I. #8702684.)

Belew, R.K. (1989). Adaptive information retrieval: Using a connectionist representation to retrieve and learn about documents. In N.J. Belkin & C.J. van Rijsbergen (Ed.), *Proceedings of the Twelfth Annual International ACMSIGIR Conference on Research and Development in Information Retrieval* (pp. 11–20). Special issue of SIGIR Forum, 23 (1–2). New York: Association for Computing Machinery.

Bush, V. (1945). As we may think. *Atlantic Monthly, 176*, 101–108.

Carroll, D.M., Pogue, C.A., & Willett, P. (1988). Bibliographic pattern matching using the ICL distributed array processor. *Journal of the ASIS, 39*(6), 390–399.

Cohen, P. & Kjeldsen, R. (1987). Information retrieval by constrained spreading activation in semantic networks. *Information Processing and Management, 23*(4), 255–268.

Collins, A.M. & Loftus, E.F. (1975). A spreading activation theory of semantic processing. *Psychological Review, 82*(6), 407–428.

Croft, W.B., Lucia, T.J., & Cohen, P.R. (1988). Retrieving documents by plausible inference: A preliminary study. In Y. Chiaramella (Ed.), *Proceedings of the 11th International Conference on Research & Development in Information Retrieval* (pp. 481–494). New York: Association for Computing Machinery.

Croft, W.B. & Thompson, R.H. (1987). I^3R: A new approach to the design of document retrieval systems. *Journal of the ASIS, 38*(6), 389–404.

Dillon, M. & Desper, J. (1980). Automatic relevance feedback in boolean retrieval systems. *Journal of Documentation, 42*, 197–208.

Doyle, L.B. (1961). Semantic road maps for literature searches. *Journal of the ACM, 8*, 553–578.

Feldman, J.A., Fanty, M.A., & Goddard, N.H. (1988). Computing with structured connectionist networks. *Communications for the ACM, 31*(2), 170–187.

Hollnagel, E. (1979). The relationship between intention, meaning, and action. *Proceedings of Informatics 5, Aslib*, 135–147.

Hypertext (1989). Perspectives on Hypertext (several articles). *Journal of the ASIS, 38*(3), 158–221.

Kwok, K.L. (1989). A neural network for probabilistic information retrieval. In N.J. Belkin & C.J. van Rijsbergen (Ed.), *Proceedings of the Twelfth Annual International ACMSIGIR Conference on Research and Development in Information Retrieval* (pp. 21–30). Special issue of *SIGIR Forum, 23*(1–2), New York: Association for Computing Machinery.

Oddy, R.N. (1975). *Reference retrieval based on user induced dynamic clustering.* Unpublished doctoral dissertation, University of Newcastle upon Tyne (U.K.).

Oddy, R.N. (1977a). Information retrieval through man-machine dialogue. *Journal of Documentation, 33*(1), 1–14.

Oddy, R.N. (1977b). Retrieving references by dialogue rather than by query formulation. *Journal of Informatics, 1*(1), 37–53.

Oddy, R.N. & Balakrishnan, B. (1988). Adaptive information retrieval using a fine-grained parallel computer. *Proceedings of the RIAO Conference*, Boston, MA.

Ofori-Dwumfuo, G.O. (1982a). *Document retrieval based on a cognitive model of dialogue.* Unpublished doctoral dissertation, University of Aston in Birmingham (U.K.).

Ofori-Dwumfuo, G.O. (1982b). Reference retrieval without user query formulation. *Journal of Information Science, 4*, 105–110.

Ofori-Dwumfuo, G.O. (1984). Using a cognitive model of dialogue for reference retrieval. *Journal of Information Science, 9*, 19–28.

Palay, A.J. & Fox, M.S. (1981). Browsing through databases. In R.N. Oddy, S.E. Robertson, C.J. van Rijsbergen, & P.W. Williams (Eds.), *Information Retrieval Research*, (pp. 310–324). London: Butterworths.

Pogue, C.A. & Willett, P. (1989). Use of text signatures for document retrieval in a highly parallel environment. *Parallel Computing, 4*, 259–268.

Preece, S.E. (1981). *A spreading activation network model for information retrieval.* Doctoral dissertation, University of Illinois. (U.M.I. #8203555.)

Quillian, M.R. (1968). Semantic memory. In M. Minsky (Ed.), *Semantic Information Processing* (pp. 227–270). Cambridge, MA: The MIT Press.

Robertson, S.E. & Sparck Jones, K. (1976). Relevance weighting of search terms. *Journal of the ASIS, 27*, 129–146.

Rasmussen, E.M. & Willett, P. (1989). Efficiency of hierarchic agglomerative clustering using the ICL distributed array processor. *Journal of Documentation, 45*(1), 1–24.

Rumelhart, D.E., McClelland, J.L., & The PDP Research Group (1986). *Parallel Distributed Processing.* Cambridge, MA: The MIT Press.

Salton, G. (1971). Relevance feedback and the optimization of retrieval effectiveness. In G. Salton (Ed.), *The SMART Retrieval System* (Chapter 15), Englewood Cliffs, NJ: Prentice Hall.

Salton, G., Fox, E.A., & Voorhees, E. (1985). Advanced feedback methods in information retrieval. *Journal of the ASIS*, *36*, 200–210.

Salton, G., & Buckley, C. (1988a). Parallel text search methods. *Communications of the ACM*, *31*(2), 202–215.

Salton, G. & Buckley, C. (1988b). On the use of spreading activation methods in automatic information retrieval. In Y. Chiaramella (Ed.), *Proceedings of the 11th International Conference on Research & Development in Information Retrieval* (pp. 147–160). New York: Association for Computing Machinery.

Stone, H.S. (1987). Parallel querying of large databases: A case study. *Computer*, *20*(10), 11–21.

Stanfill, C. & Kahle, B. (1986). Parallel free-text search on the Connection Machine system. *Communications of the ACM*, *29*(12), 1229–1238.

Stanfill, C., Thau, R., & Waltz, D. (1989). A parallel indexed algorithm for information retrieval. In N.J. Belkin & C.J. van Rijsbergen (Ed.), *Proceedings of the Twelfth Annual International ACMSIGIR Conference on Research and Development in Information Retrieval* (pp. 88–97). Special issue of *SIGIR Forum*, *23* (1–2), New York: Association for Computing Machinery.

Thinking Machines Corporation. (1988). *Connection Machine programming in C*: Version 5.0.* Cambridge, MA: Thinking Machines Corporation.

Vernimb, C. (1977). Automatic query adjustment in document retrieval. *Information Processing & Management*, *13*, 339–353.