A VARIABLE SUPPLY MODEL FOR DISTRIBUTING DEDUCTIONS

Vineet Singh
Michael R. Genesereth
Computer Science Department
Stanford University
Stanford, CA 94305

ABSTRACT

Multiple processors can be used to speed up a backward-chaining deduction by distributing or-parallel deductions. However, the actual speedup obtained is strongly dependent on the amount of communication required for the task allocation strategy. A Variable Supply Model (VSM) is presented for multiple processors with replicated databases on a broadcast network. The term *model* refers to the set of procedures and messages required to perform the computation. VSM allows an infinite class of strategies with varying amounts of communication. The utility of VSM lies in the easy and powerful way it provides for selecting a strategy that works satisfactorily given certain communication constraints. All strategies in VSM use a dynamic task supply protocol (ESP) that works better than other supply protocols described in the literature.[1]

I INTRODUCTION

Parallelism has been identified as a key to future high-performance reasoning machines - the fifth generation of computer systems (Motooka, 1984). Multiple processors must be made to cooperate to speed up a computation. Clearly, there is a need to identify parallel components of reasoning computations and there is a need to build multiprocessor hardware. In addition, there is a need to map parallel computations onto multiple processors, keeping in mind the constraints of communication. This last need is being largely ignored at the present time and may become the bottleneck in achieving high performance from multiple processors. The purpose of this paper is to address the last need (i.e., to propose methods for task allocation that work well in the presence of communication constraints).

Task allocation in a multiple processor system strongly affects the overall speedup obtained for a parallel computation (Conway, 1967). Also, communication cost can be a significant part of the cost of the computation. Therefore, a good task allocation strategy is needed and can be obtained only if both processing costs and communication costs are kept in mind.

The type of computation being considered in this paper is backward-chaining deduction (Barr, 1982) with no side-effects, and the parallelism employed is or-parallelism (Conery, 1983, Lindstrom, 1984, Ciepielewski, 1983). In addition, all the processors can do backward-chaining deductions[2] and are connected by a broadcast network.[3] Moreover, the database is replicated at each processor.[4] The goal is to complete the deduction in as short a time as possible.[5] The task allocation strategies presented in this paper do not assume any knowledge of the domain of application (i.e., the database of facts and rules may be used only syntactically). The task allocation strategies can, however, use extra information to improve their performance.

Previous approaches to parallel task allocation (Lawler, 1982) will not work well in this domain because of assumptions that are not reasonable here. Most techniques can be eliminated as inapplicable because they assume that all tasks to be allocated are known beforehand along with their processing requirements (Mayr, 1981). Another large class of techniques can be eliminated because communication cost is not considered or it is inaccurately modeled (Lageweg, 1981). More likely contenders will be considered later in the paper.

The importance of this paper lies primarily in providing two kinds of mechanisms to control communication cost. First, clever ways are found to reduce communication cost outright. ESP, a communication network protocol for transferring tasks among processors is more efficient than the *announcement-bid-award* protocol of Contract Net (Davis, 1983) and Enterprise (Malone, 1983). Second, mechanisms are described to trade off communication cost and parallelism. The Variable Supply Model (VSM) allows flexible usage of the inter-processor throughput in this respect. The term *model* refers to the set of procedures and messages required to perform the computation. *Throughput* means the amount of data (in terms of messages) per unit time that can be sent on the communication network. VSM uses ESP as its communication protocol.

This paper is organized as follows. Section II explains how to view a backward-chaining deduction as a tree of or-parallel tasks. Section III contains a description of the multiple processor architecture. Section IV then describes VSM and how it can be used to control communication. Section V describes ESP and how it can be used as an efficient task supply protocol for all the strategies in VSM. Sections VI and VII contrast the results of using the two extremes of VSM - a supply-driven strategy and a demand-driven strategy. Useful extensions to VSM and ESP and how they might fit into the current framework are described in Section VIII. That section also contains directions for future work and a summary.

II BACK-CHAINING AS OR-PARALLEL COMPUTATION

And-or trees (Barr, 1981) can be used to represent the problem-reduction in a backward-chaining deduction (Barr, 1982). In this paper, only or-parallelism is used; no and-parallelism is exploited. Therefore, an *or* tree to represent a backward-chaining deduction, as described below, is of more interest than an *and-or* tree. Backward-chaining is

---

[1] This work was supported in part by ARPA contract N00039-83-C-0136.

[2] They may, however, work at different speeds.

[3] The multiple processor scenario applies to both multiple processors connected by a broadcast local area network (like the CSMA-CD Ethernet (Metcalfe, 1976)) or multiple processors connected by a broadcast network on a single chip as suggested by Ullman (Ullman, 1984).

[4] There are two reasons for choosing this multiprocessor architecture. First, it is a practical way to utilize personal workstations when their owners are not using them (Malone, 1983)). Second, this paper can be a stepping stone to studying multiprocessors with more complex interconnection structures.

[5] It is well known that optimal task allocation even for relatively simple problems is NP-complete (Mayr, 1981). Therefore, no attempt will be made to get an optimal solution.

used in the context of rule-based systems in which the data base consists of a dynamic set of First Order Predicate Calculus (FOPC) propositions and the rule base consists of a static set of FOPC rules. In this paper, even the data base is kept unchanged as the deduction proceeds. All of the propositions in the data base are literals (i.e., either atomic propositions or negations of atomic propositions). All propositions in the rule base are required to be written in one of the forms shown below, where alpha, beta, and $alpha_1$,...,$alpha_n$ are all literals.[6]

(*if* alpha beta)

(*if* (*and* $alpha_1$...$alpha_n$) beta)

Similarly, a goal to be proved must also be a literal or a conjunction of literals. In the rest of the paper, the term data base will be used to refer to the union of the rule base and the data base as described above.

Each node in the *or* tree can be represented as a tuple of two sets: a set of goals and a set of bindings. The top-level node's set of goals contains one literal and its set of bindings is empty. If the set of goals of a node in the tree is non-empty, its children can be obtained in the following three ways: (1) Take one of the goals in the goal set and backward-chain with all possible rules. A child is created for each of the rules that can be used to backward-chain from the parent. A child's goal set is obtained from the parent's goal set by removing the goal that was backward-chained on, adding the antecedents of the rule applied, and applying the unifier from the backward-chaining to the resulting goal set. The set of bindings of the child is obtained by adding the unification bindings to the set of bindings of the parent. (2) Unify a goal from the goal set of a parent node with a proposition in the data base. A proposition in the data base is treated as a rule in (1) with no antecedents. (3) If (1) and (2) cannot be used, no child can be created and this chain of backward-chaining ends in failure.

Figure 1 presents an example of an *or* tree. A leaf represents a positive result if its set of goals is empty.

Any node in the *or* deduction tree may be referred to equivalently as a task. A *unit deduction* is the expansion of a node into its children. The result of a deduction is the binding list that satisfies the proposition in question, if only one positive result is desired, or it is the list of binding lists that satisfy the proposition, if all positive results are desired.

Notice that each of the subtasks of a task can be solved completely independently. No communication is required among the subtasks. Also, notice that the grain of the tasks in the deduction tree can range all the way from 1 unit backward-chaining deduction to an arbitrarily large size. Moreover, backward-chaining deductions are side-effect free.[7]

III THE MULTIPLE PROCESSOR ARCHITECTURE

The goal of this paper is not to describe multiple processor architectures in any great detail. All we require is that the multiple processor architecture satisfy certain properties described below.

(1) Many sequential processors are connected to a broadcast network; (2) The memory at each processor is large enough to store a complete copy of the database; (3) Message-passing is the only form of inter-processor communication; (4) A message may be either point-to-point or broadcast; and (5) Each processor has a unique processor number. An example of this kind of architecture is multiple Symbolics 3600 processors connected to a Chaosnet (Moon, 1982).

The local time at each processor is used for time-stamping messages. It is impossible to have all processor clocks completely synchronized but a close synchronization is desirable (and can be obtained by a standard distributed clock synchronization algorithm (Marzullo, 1984)).

[6]The notation for propositions is taken from (Genesereth, 1983).

[7]They are side-effect free at least in their pure form. One could have side-effects, for example, by allowing caching, but that is not considered here.

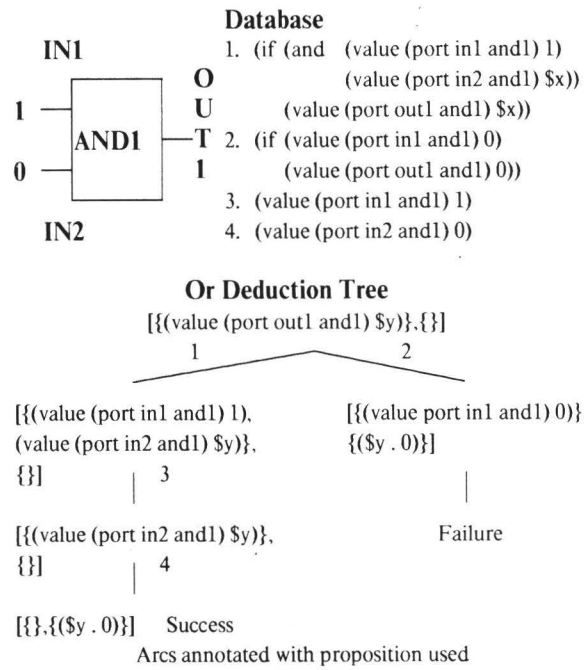

Figure 1: An Example *Or* Tree

IV VARIABLE SUPPLY MODEL (VSM)

VSM allows a class of strategies with varying amounts of communication. It will be shown later that the strategy with the highest communication cost will work best when inter-processor communication throughput is very plentiful relative to the message traffic that needs to take place. Also, the strategy with the lowest communication cost will work best when the inter-processor communication throughput is very low. The utility of VSM lies in providing a unifying framework for an infinite set of strategies with varying amounts of communication, in the ease of selection of these strategies, and in the demonstrated usefulness of the two extreme strategies. The use of intermediate strategies for intermediate communication conditions seems likely but is left unexplored in this paper.

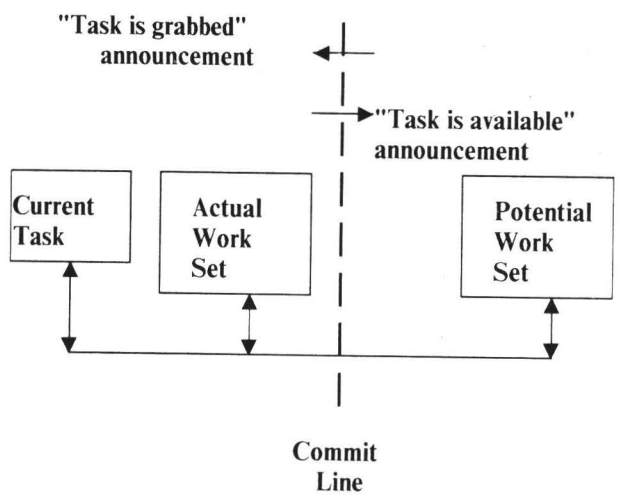

Figure 2: The Variable Supply Model

VSM lays down a specific internal organization of the set of tasks known to each processor. This organization is shown in figure 2. **Current Task** is the task being worked on by the processor, **Actual Work Set** is the set of tasks that the processor is committed to executing, and **Potential Work Set** is the set of tasks that no processor is known to be committed to perform. The lines with arrows show the directions of transfer of tasks. Any tasks that flow from left to right across the hypothetical **commit line** must be announced to other processors as being available so that they may place the task in their **potential work sets**. Similarly, any tasks that flow from right to left across the **commit line** must be announced as being grabbed by this processor so that other processors may not also grab the same task.

Ideally, the **potential work sets** of all the processors are the same at all times and each task is grabbed by one and only one processor. However, because of non-zero message delays, more than one processor may attempt to grab a task. A protocol understood by all the processors is required so that multiple grabs are resolved, results are properly handled, and tasks are killed when they are no longer needed. An efficient communication protocol (ESP) to handle these problems is described in the next section.

In VSM, the mechanism for controlling the amount of communication lies in the fact that the tasks in the **actual work set** and the **potential work set** can be arbitrarily balanced. As will be seen later, if all newly generated tasks are placed in the **potential work set** and grabbed only when absolutely needed, one obtains the supply-driven strategy with the highest communication cost. Similarly, if all newly generated tasks are placed in the **actual work set** and only placed in the **potential work set** when absolutely needed, one obtains the demand-driven strategy with the lowest communication cost. This will be explained in greater detail in sections VI and VII.

For all the strategies in VSM, if a new **current task** is needed, it is picked out of the **actual work set** if it is non-empty. Otherwise, the **current task** is picked from the **potential work set**.

V EFFICIENT SUPPLY PROTOCOL (ESP)

ESP is the network communication protocol that is used by VSM to transfer tasks among processors. It may also be referred to as a task supply protocol.

First, this section presents a detailed description of ESP. Next, the domain of application of ESP, which is larger than the present application alone, is described. Finally, some previous work is contrasted with the work presented here.

A. Detailed Description of ESP

This description consists of two parts: Tasks and Messages.

1. Tasks

Each task has a globally unique name. Attached pieces of information are:

1. Task Description: (*PROP-SET B-SET ALL?*), where *PROP-SET* is the set of propositions to be proved, *B-SET* is the set of bindings obtained so far, and *ALL?* is a boolean variable that means all positive results must be found (if true) or only one positive result must be found (if false)[8].

2. Parent Task

3. Pending Subtasks

4. Results: This is the set of results received so far for this task.

5. Grabbed Timestamp: This is the timestamp[9] at which the task was grabbed. This can be revised if a future "grab" request for the same task "wins" out over a previous "grab" request. In a comparison between two "grab" requests, the one with the lower timestamp "wins".

2. Messages

Each message is described here with the syntax *Message-Type*(*Arguments*). The following four message types are required:

1. **New**(*Task-Name Task-Description*): This is the message used to make the "Task is available" broadcast mentioned in section IV. When a processor receives a **new** message, the processor sets up the appropriate book-keeping information for the task name and puts the task name in the **potential work set**.

2. **Grabbed**(*Task-Name Timestamp*): This is the message used to make the "Task is grabbed" announcement mentioned in section IV. When a processor selects a certain task as the next **current task** for itself, it broadcasts a **grabbed** message for that task. When a **grabbed** message is received for a task, it is removed from the **potential work set** of the processor where the message is received and the book-keeping information for the task is revised (if required). If the task for which the **grabbed** message is received is the current task or even if the task is awaiting completion of its subtasks, then it will have to be aborted if the timestamp in the new **grabbed** message is lower than the previous timestamp for the task name. Aborting a task also means aborting any pending subtasks for the task. A remote task is aborted by sending a **kill** message as explained below.

3. **Kill**(*Task-Name*): A **kill** message is sent to abort tasks remotely. When a **kill** message is received, the task name is removed from the **potential work set**. In case the task in question is the current task, it is aborted and another current task is chosen. Also, if the task is awaiting completion of some of its subtasks, the subtasks are recursively aborted.

4. **Done**(*Task-Name Binding-List*): This is sent when the answer for a remotely originated task is obtained. When a **done** message is received, the result is reported to the parent of the task in question. The result is added to the results already obtained for the parent task name. If there are no pending subtasks for the parent task name, then the combined result for the parent task is reported to the processor that originated the parent task.[10] Again, a **done** message may have to be used if the originator of the parent task was a remote processor. The entire computation is

[8] In the current implementation, *ALL?* is always true.

[9] A timestamp is composed of the time in its higher order bits and a fixed number of bits for a unique processor number in its lower order bits. A timestamp with a lower (greater) time than another timestamp is always lower (greater) than the other. When the times are equal, the timestamp with the lower numbered processor is lower.

[10] For the case in which a single answer is required for the top-level deduction, a positive answer can be sent to the processor that originated the top-level deduction and the rest of the computation can be terminated. This is not done in the current implementation, however.

complete when all needed results (one or all) for the top-level task are reported.

3. Heuristics

Several heuristics, as described below, are used to reduce computation and communication.

When it is time for a processor to select a new **current task** out of the **potential work set**, locally generated tasks are preferred; this saves on sending a **done** message for that task when the task is completed.

Backward-chaining within a processor is done in a depth-first fashion. This reduces the search (compared to breadth-first) if only one positive result is desired.[11]

To reduce the number of grabs for the same task, the following heuristic is used. Grabs by processors are given some arbitrary but pre-specified priority order. By watching messages on the broadcast network, processors get hints about the status of other processors. For example, if a processor sends out a **grabbed** message, other processors know that the processor is busy. When a new task is made available, a processor will defer grabbing to a higher-priority processor if both are free.

When deciding which tasks to move to the **potential work set**, the most costly is picked. Also, when deciding which task to grab out of the **potential work set**, the most costly is chosen. This is done so that a free processor grabs the most costly task first out of the surplus tasks available from all the processors. The effect is that, in general, processors tend to remain busy longer between grabs and this reduces the message traffic.

If no information is available about the cost of deductions, the least one can go by is that deductions higher up in the deduction tree are, on the average, more costly. Also, if the delays are not very high, all processors can be heuristically assumed to be at the same level in the tree. Therefore, the order of announcement of new tasks on the broadcast network can be heuristically assumed to be in the more costly to less costly order.

B. Domain of Application of ESP

More than one processor can start executing a task at the same time. Therefore, either the tasks should be side-effect free as in the backward chaining case or repetitions of side-effects should be acceptable. Examples of acceptable repetitions of side-effects include 1) side-effects designed only to increase efficiency but not to affect the correctness of the computation (e.g., caching of propositions in the backward chaining case) and 2) idempotent computations.

As mentioned before, ESP can handle a realistic failure set. Details of this can be found in a technical report written by the authors (Singh, 1984).

C. Comparisons with Previous Work

Contract Net (Davis, 1983) was one of the first efforts to address the problem of dynamically distributing tasks among processors. Its supply protocol uses what is called an *Announcement-Bid-Award* sequence. Enterprise (Malone, 1983) uses a similar supply protocol but with significant specializations. In the discussion below, ABASP stands for the *Announcement-Bid-Award* type of protocol used in the Contract Net and Enterprise.

ESP allows task execution to begin as soon as a bid is submitted (in a **grabbed** message). This can increase processor utilization compared to ABASP in which task execution begins only when an award message is received.

Also, no award message is required as in ABASP. Additionally, there is the potential of drastically reducing the number of **grabbed** messages, the equivalent of *bid* messages in the ABASP case. The number of **grabbed** messages may, in the best case, be one per new task announced. This will happen if the first **grabbed** message is always sufficiently early to inhibit any other processor from making an attempt to "grab" the same task. Also, recall that a heuristic to reduce the number of **grabbed** messages was presented earlier.

Note, however, that there is one source of extra messages in the supply-driven strategy using ESP. Multiple executions of the same task are possible. In theory, these multiple executions can create multiple sub-tasks (with extra **new** messages) in parallel that have to be killed off by extra **kill** messages. In practice, experimental results indicate that this is not a problem when delays are reasonably short and throughput is not a constraint. Short delays lead to a quick killing of replicated executions before extra sub-tasks can be generated and announced from the short-lived replicated executions. The throughput constraint case will create a problem for Enterprise and Contract Net as well but they do not propose any specific solutions. This paper, however, presents VSM to deal with that case.

Shapiro's Bagel (Shapiro, 1983) also deals with dynamic distribution of tasks in a multiprocessor system. However, Shapiro deals with *systolic* problems where programmer determined mappings are possible. Similar assumptions in the present case would mean that one would know ahead of time how many subtasks were going to be generated from each task. This paper makes no such assumption.

VI A SUPPLY-DRIVEN ALLOCATION STRATEGY

Imagine for a moment that the broadcast network can allow an infinite message throughput. This assumption is, of course, not practical but it serves to illustrate the extreme of very plentiful network throughput. A supply-driven strategy, in which all newly generated tasks are announced to all processors, might be appropriate in precisely such a situation.

In the supply-driven strategy, all processors place all newly generated tasks, except one task that is chosen as the **current task**, in the **potential work set**. Of course, all tasks placed in the **potential work set** must be announced with **new** messages. A processor is allowed to grab a task from the **potential work set** only when it runs out of internally generated tasks. Since all surplus tasks are announced as being available, this strategy needs the highest network throughput of all strategies included in VSM.

A. Experimental Results

All results reported here were obtained by implementing VSM and ESP on a simulation of the multiple processor architecture described in section III.[12] The unit of time for the results reported is the time taken to do a unit backward-chaining deduction.

Figure 3 gives some information about the *or* deduction tree that was experimented with. The database of each processor contains the behavioral and structural description of a piece of digital hardware - a 4-bit adder. It also contains the values of its inputs. The top-level proposition is a fact to be proved about one of the outputs of the adder. It turns out that the top-level proposition has no positive answer and, therefore, the entire deduction tree is searched in trying to prove the proposition.

The time taken for 1 processor to do the adder example is 652 unit deduction time units. Figure 4 shows the results obtained for the adder example with 1 to 10 processors. For now, the reader's attention is directed to the curves labeled *Supply-Driven, Infinite Throughput.*

---

[11] This pure depth-first strategy will have to be modified if there are infinite depth paths. In the current implementation, only depth-first is used within a processor.

[12] The simulation was performed on a sequential processor (Symbolics 3600). A preliminary version was implemented with multiple (real) 3600's communicating over a Chaosnet to obtain the parameters for a realistic simulation. More details in (Singh, 1984).

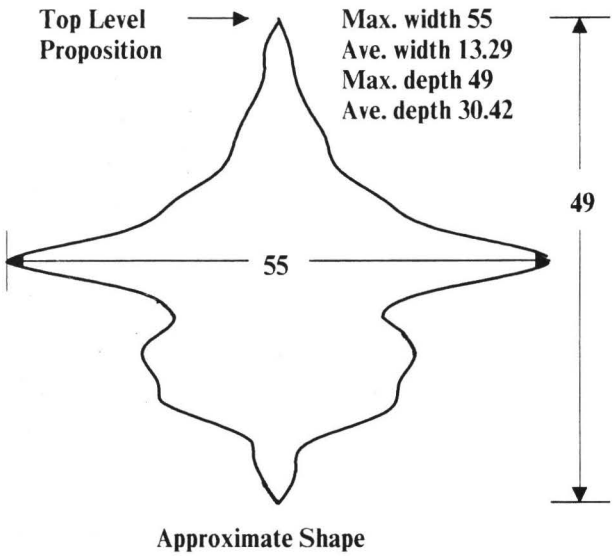

Figure 3: Or Deduction Tree for Adder Example

Notice from the figure that initially the speedup is almost linear but it becomes less so as the number of processors increases; the speedup obtained by using ten processors (7.01) is still substantial. Some of the less than linear speedup is due to the inter-processor message delay but some is certainly due to the fact that the deduction tree for the example does not permit very large speedup due to its shape (see figure 3). The deduction tree is not very wide at some depths and, therefore, all processors cannot be kept busy at all times by any possible supply protocol. A wider deduction tree for a bigger example can certainly be expected to lead to larger speedups.

The message traffic does not vary much when the number of processors is varied from 1 to 10. The total number of messages sent increases from 165 for 1 processor to 213 for 10 processors.

Note that although the speedup curve for the supply-driven strategy for infinite throughput looks somewhat like a logarithmic curve, it is a mistake to associate this with Minsky's hypothesis (of log(# of processors) speedup) (Minsky, 1970).[13]

The results mentioned so far were obtained with an allowed throughput of infinity. It is certainly possible that the throughput is not enough to send all the messages that are queued at some time in the next unit deduction time. One might expect that with a throughput bottleneck the performance of the supply-driven strategy will degenerate. This is exactly what is observed in the simulation (as shown in the curves labeled *Supply-Driven, Bottleneck Throughput*) when a fairly severe throughput limitation of 2 is imposed. The delay is one as before. The speedup for 10 processors is only 2.80.

B. Conclusions

The supply-driven strategy works well when the throughput is not a bottleneck. However, the performance can be quite bad if the throughput does become a bottleneck.

It is important to note here that most of the time, it was not really necessary to announce all surplus tasks when all processors were already busy. The next section offers an alternative that can reduce the amount of communication and thereby decrease overall task completion time for the throughput bottleneck case.

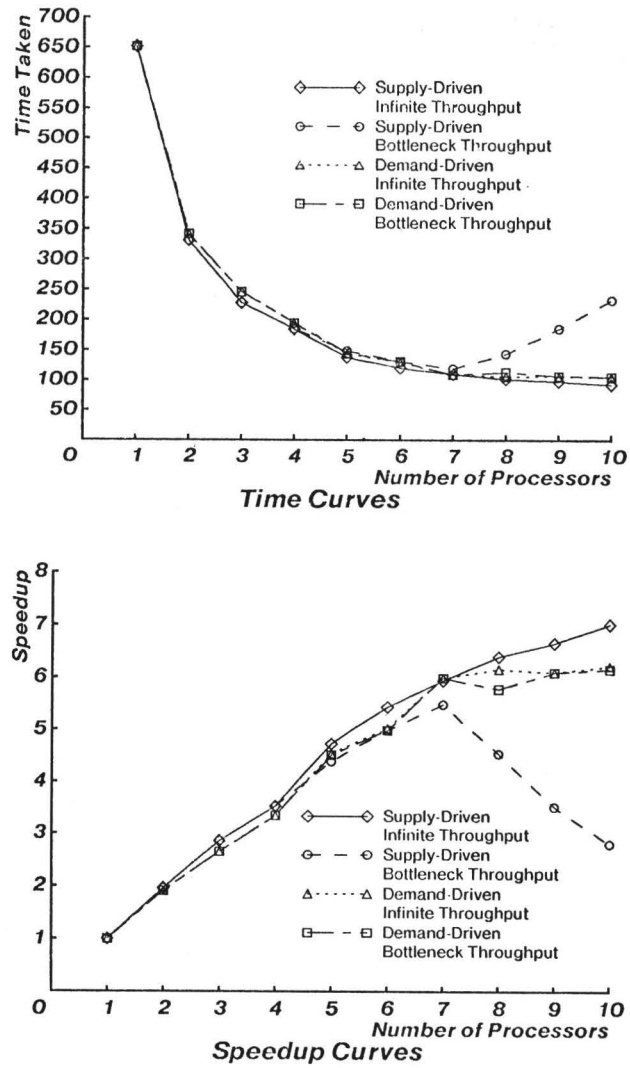

Figure 4: Data

VII A DEMAND-DRIVEN ALLOCATION STRATEGY

The supply-driven strategy always supplies any surplus tasks and, therefore, uses up a lot of throughput. The other extreme is to supply surplus tasks only when all previously announced tasks have been grabbed. This is the demand-driven strategy. As will be shown later, this can drastically reduce message traffic and lead to faster completion of tasks when the available throughput is a bottleneck.

All surplus tasks are not supplied and, therefore, not placed in the **potential work set**. The surplus tasks that are not supplied are put in the **actual work set**. More precisely, when a processor generates more

[13] If communication cost is ignored as in Minsky's hypothesis, the speedup curves for the supply-driven strategy in the infinite throughput case will be asymptotically linear, as long as the size of the *or* deduction tree is large enough compared to the number of processors. Communication cost certainly reduces speedup but there is no proof that speedups will be only logarithmic in the general case.

than one task, it keeps one as its next current task, supplies a task to the **potential work set** if the **potential work set** is empty, and keeps the remaining tasks in its **actual work set**. With this strategy, the tasks are supplied only when they are demanded (i.e., when the **potential work set** gets empty).

A. Comparison with Supply-Driven Allocation Strategy

Not only does the demand-driven strategy use fewer messages than the supply-driven strategy, as explained above, but in some cases the number of messages can be provably an infinitesimal fraction of the supply-driven case. One such case is when the deduction tree is a balanced, binary tree and the ratio of the number of tasks to the number of processors goes asymptotically to infinity. More details are provided in (Singh, 1984). In addition, experimental data presented later will show a dramatic reduction in messages in a practical case.

Another attractive property of the demand-driven strategy is that it requires less storage. Most tasks are kept locally whereas in the supply-driven case, all processors keep a copy of tasks not being worked on.

There is, however, a disadvantage with using the demand-driven strategy. There may be, in general, a higher delay in transferring tasks among processors. This can happen when more than one processor becomes free at the same time; one processor can grab the single task in the **potential work set** but the others have to wait for some surplus task to be transferred to the **potential work set**. An example in which the behavior of the demand-driven strategy is, in fact, unboundedly worse is where the total rate of generation of tasks is equal to the total rate of consumption of tasks and the rate of generation and consumption is not uniform over all tasks. A high delay in such a situation would further compound the problem.

B. Experimental Results

The curves labeled *Demand-Driven, Infinite Throughput* in figure 4 illustrate the performance of the demand-driven strategy for the adder example for unit delay and infinite throughput.

As can be seen by comparing the curves labeled *Demand-Driven, Infinite Throughput* and *Supply-Driven, Infinite Throughput*, the time taken for 10 processors with the demand-driven strategy (105) is slightly more that the time taken with the supply-driven strategy (93). The extra time taken is due to the extra delay in supplying tasks. However, the number of messages sent with the demand-driven strategy (90) is much lower than the number of messages sent with the supply-driven strategy (213). Therefore, it should be expected that the demand-driven strategy will not degrade as much in the presence of the same throughput limitation imposed on the supply-driven strategy. This is, in fact, the case as illustrated by the curves labeled *Demand-Driven, Bottleneck Throughput*. The delay is one and the maximum throughput is 2.

For 10 processors, the time taken for the demand-driven strategy only increases from 105 to 106 by changing the maximum throughput from infinity to 2. In the bottleneck case, the demand-driven strategy (with a time of 106) completely outperforms the supply-driven strategy (with a time of 233).

C. Conclusions

The demand-driven strategy can perform better than the supply-driven strategy in the presence of throughput constraints because it requires fewer messages.

It is possible that the tradeoff between delay in supplying tasks (high for the demand-driven case) and the message traffic requirements (high in the supply-driven case) may not have been optimally resolved in either of the extremes. An increase in delay may also be viewed as a reduction in parallelism. VSM allows any intermediate strategy to be selected with great ease and it seems quite probable that intermediate strategies will be found useful for intermediate communication situations. However, no results are available at the present time about these intermediate strategies. Moreover, no results are available on automating the selection of such a strategy.

VIII CONCLUSIONS

This section contains a description of possible extensions, future work and a summary of the paper.

A. Extensions

The variable supply idea is actually more general than what its application so far in VSM, in conjunction with ESP, might suggest. The extensions described in this section retain the essential goal of VSM, the ability to vary the supply of new tasks given that communication constraints may exist.

1. Extension to Different Task Domains

In cases where side-effects are not acceptable, one can use a different supply protocol with VSM that does not allow multiple instances of a task to start executing. For example, one could use the announcement-bid-award supply protocol (of Contract Net or Enterprise).

2. Extension to Different Processor Interconnection Structures

It is possible to apply VSM to cases where the interconnection structure is not a single broadcast network. Essentially, broadcasts are replaced by *limited broadcasts*. The subset of processors to which a *limited broadcast* is directed is determined solely by the originating processor of the task. More details can be found in (Singh, 1984).

3. Using Additional Information

Costs of Deductions

So far costs of deductions have not been used because it is not easy to make accurate estimates. However, if one does have a means to obtain reasonable estimates, several uses are possible.

An application of cost estimates can be to stop distribution of certain propositions if their costs are too low to justify the overhead of distribution to other processors.

Also, estimates can be used to break ties between processors with different speeds in a more efficient way than by just comparing the timestamps at which grabs take place. Timestamps based on completion time estimates will be more profitable.

Another application of estimates can be to do load balancing.

Probabilities of Proving Propositions

These probabilities, along with costs of proving those propositions, can be used to determine the best order of doing the tasks (Barnett, 1984, Simon, 1975). Further, Rosenschein and Singh (Rosenschein, 1983) place an upper bound on the amount of work that may be done to achieve the optimal order determined as above. It may be possible to generalize that upper bound result for an arbitrary number of tasks. On the other hand, the result may be more immediately useful in doing many pairwise redistributions in the hope of getting most of the power of a complete redistribution.

B. Future Work

A major effort of future work (Singh, 1985) will be to remove the two major bottlenecks present in the current method of distribution of deductions: the replicated database and the shared communication network. The architecture to be used will consist of large numbers of processors connected with local connections to neighbors (as in a hex-connected plane, for example). Each processor will have a limited amount of local memory that can only store a small part of the entire database.

And parallelism will also be taken advantage of in a limited way. Taking full advantage of *and* parallelism is a very difficult problem.

C. Summary

Controlling communication cost was seen to be crucial for the successful use of the multiprocessor used. ESP allows communication cost to be reduced in comparison to other task supply protocols. VSM allows an easy and powerful mechanism to choose strategies that work best under different communication constraints. The two extreme strategies were shown to be useful by theoretical and experimental results. In addition, VSM and ESP can be adapted for use with other tasks, interconnection structures, and additional information about tasks.

ACKNOWLEDGMENTS

We wish to thank Ernst Mayr for many stimulating discussions and all others that commented on earlier drafts of this paper.

REFERENCES

Barnett, J.A. How Much is Control Knowledge Worth?: A Primitive Example. *Artificial Intelligence*, January 1984, *22(1)*, 77-89.

Barr, Avron and Edward A. Feigenbaum (Eds.). Search. In *The Handbook of Artificial Intelligence*, : William Kauffman, Inc., Los Altos, California, 1981.

Barr, Avron and Edward A. Feigenbaum (Eds.). Automatic Deduction. In *The Handbook of Artificial Intelligence*, : William Kauffman, Inc., Los Altos, California, 1982.

Ciepielewski, Andrzej and Haridi, Sief. *A Formal Model for Or-Parallel Execution of Logic Programs*, in *Proceedings of the IFIP Congress*, pages 299-305, IFIP, 1983.

Conery, John Simpson. *The And/Or Process Model for Parallel Interpretation of Logic Programs.* PhD thesis, University of California, Irvine, 1983.

Conway, R.W., W.L. Maxwell, and L.W. Miller. *Theory of Scheduling.* : Addison-Wesley, Reading, Massachusetts 1967.

Davis, R. and R.G. Smith. Negotiation as a Metaphor for Distributed Problem Solving. *Artificial Intelligence*, January 1983, *20(1)*, . Also available as MIT AI memo # 624.

Genesereth, Michael R. *A Meta-Level Representation System.* Technical Report HPP 83-28, Heuristic Programming Project, Computer Science Department, Stanford University, 1983.

Lageweg, B. J., E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. *Computer Aided Complexity Classification of Deterministic Scheduling Problems.* Technical Report BW 138/81, Mathematisch Centrum, Amsterdam, 1981.

Lawler, E.L., J.K. Lenstra, and A.H.G. Rinooy Kan. Recent Developments in Deterministic Scheduling. In *Deterministic and Stochastic Scheduling*, : Reidel, Dordrecht, 1982. Also available as tech. report BW 146/81 from Mathematisch Centrum, Amsterdam.

Lindstrom, Gary and Panangaden, Prakash. *Stream-Based Execution of Logic Programs*, in *IEEE Logic Programming Conference*, pages 168-176, IEEE, February, 1984.

Malone, T.W., R.E. Fikes, and M.T. Howard. *Enterprise: A Market-like Task Scheduler for Distributed Computing Environments.* Working Paper, Cognitive and Instructional Sciences Group, Xerox Palo Alto Research Center, Palo Alto, California, October 1983.

Marzullo, Keith. *Maintaining the Time in a Distributed System - An Example of a Loosely-Coupled Distributed Service.* PhD thesis, Stanford University, February, 1984.

Mayr, E. W. *Well Structured Parallel Programs Are Not Easier to Schedule.* Report No. STAN-CS-81-880, Stanford University, September 1981.

Metcalfe, R.M. and D.R. Boggs. Ethernet: Distributed Packet Switching for Local Computer Networks. *Communications of the ACM*, July 1976, *19(7)*, 395-404.

Minsky, M. Form and Content in Computer Science. *J. ACM*, 1970, *17*, 197-215.

Moon, David A. Chaosnet. Symbolics, Inc., 1982. Printed by permission of Massachusetts Institute of Technology.

Moto-oka, Tohru. Fifth Generation Computer Systems: A Japanese Project. *Computer*, March 1984, , 6-13.

Rosenschein, Jeffrey S. and Vineet Singh. *The Utility of Meta-level Effort.* Report No. HPP-83-20, Heuristic Programming Project, Stanford University, March 1983.

Shapiro, Ehud. *The Bagel: A Systolic Concurrent Prolog Machine.* Technical Report TM-0031, ICOT, Japan, November 1983.

Simon, Herbert A., and Joseph B. Kadane. Optimal Problem-Solving Search: All-or-None Solutions. *Artificial Intelligence*, 1975, *6*, 235-247.

Singh, Vineet and Michael R. Genesereth. *A Variable Supply Model for Distributing Deductions.* Technical Report HPP-84-14, Heuristic Programming Project, Computer Science Department, Stanford University, May 1984.

Singh, Vineet. *Distributing Deduction to Multiple Processors.* PhD thesis, Stanford University, December, 1985.

Ullman, Jeffrey D. *Some Thoughts about Supercomputer Organization*, in *Proceedings of COMPCON*, pages 424-432, IEEE Computer Society, February, 1984.