An Improved Cost Function for Static Partitioning of Parallel Circuit Simulations Using a Conservative Synchronization Protocol

Kevin L. Kapp, Thomas C. Hartrum, and Tom S. Wailes
Department of Electrical and Computer Engineering
School of Engineering
Air Force Institute of Technology

Abstract

*Distributing computation among multiple processors is one approach to reducing simulation time for large VLSI circuit designs. However, parallel simulation introduces the problem of how to partition the logic gates and system behaviors of the circuit among the available processors in order to obtain maximum speedup. A complicating factor that is often ignored is the effect of the time-synchronization protocol (*conservative *[1] or* optimistic *[2]). Inherent in the partitioning problem is the question of how to effectively measure the relative quality of a partition. This paper describes an objective cost function for measuring the relative quality of a task partition that includes a synchronization factor for a conservative* NULL*-message protocol. A graph-based partitioning tool based on this cost function is used to perform the static task allocation for parallel simulation of a structural VHDL circuit. Results for two 1000 - 4000 gate circuits demonstrate that the additional consideration of the synchronization protocol in the cost function generates partitions that exhibit improved speedup.*

1 Introduction

As the size of VLSI circuit designs become very large, involving thousands of gates, simulation of the design becomes a critical part of the validation and design analysis process. Unfortunately, the time it takes to simulate a large circuit design is often prohibitive. The use of parallel processors offers the potential to alleviate this problem. However, if the circuit is not properly divided among the processors, this potential will not be realized. Finding the best possible partitioning of the circuit in order to obtain the fastest simulation time is the goal of many ongoing research efforts [3][4][5].

For discrete event simulation, the partitioning problem is complicated by the requirement to synchronize the local simulation times of the processors in order to avoid causality errors. One approach to this problem, the *conservative* approach [1], guarantees that causality errors will not occur by forcing a processor to block until it is certain not to receive a message from another processor with a simulation time in its "past." In order to avoid deadlock, *NULL* messages are sent between processors under various conditions.

This paper examines a graph-based approach to the static task partitioning problem for parallel discrete event circuit simulations [6]. Structural circuit simulations modeled using the VHISIC (Very High Speed Integrated Circuit) Hardware Description Language (VHDL) are used as the application domain. A model of the relative cost of a partition which accounts for the additional overhead of the conservative *NULL*-message synchronization protocol is developed. This cost model is used to evaluate candidate partitions in an iterative improvement strategy that converges to a final partition. The resulting partition is shown to be an improvement over other partitioning algorithms that do not account for *NULL* message traffic.

Data is also provided to show that as the number of *LPs* is increased, the inter-processor communications due to event messages (referred to as *real* messages) is increasingly dominated by the inter-processor communications due to *NULL* messages from the conservative synchronization protocol.

The remainder of this paper is as follows. First, previous and related work is reviewed. Then, in Section 3 the simulation model and conservative algorithm used in this work is discussed. Section 4 presents the cost model developed that includes a conservative protocol term and Section 5 defines the iterative improvement algorithm that uses this cost model. Results for two medium size (1000 - 4000 gates) circuits are presented in Sections 6 and 7, while some conclusions and needs for future research are discussed in Sections 8 and 9.

2 Previous and Related Work

In the context of parallel programming applications, the mapping problem is defined as the binding of the logical components of the parallel application program to the physical resources of the target parallel system such that some desired performance criterion is optimized [7] (i.e. minimizing total execution time). The mapping problem typically arises when the number of processes (*i. e.* tasks) required by the parallel application is greater than the number of available processors.

Optimal solutions to the general mapping problem have been shown to be NP-complete and no polyno-

mial time algorithm for their solution is known to exist. As a result, sub-optimal solutions are often pursued using various heuristic methods [7][8]. A common approach is to somehow generate a "good" starting partition, then try to improve it by iteratively moving tasks from one processor to another and evaluating the move via a *cost function.*

The circuit simulation can be modeled as a task allocation problem involving the allocation of several inter-dependent tasks of a single program among the processors in a distributed or parallel system in order to minimize the completion time of a single application program. Due to the nature of physical circuits, most researchers have pursued a static partitioning approach since interdependencies in the task structure can be statically defined a priori and do not change with time [9].

The two key objectives of most strategies that have been proposed for the general parallel program mapping problem are achieving a balanced computation load among all of the processors, and minimizing interprocessor communication. The former deals with making efficient use of all of the processor resources to avoid idle processors due to lack of work, while the latter deals with reducing non-productive overheads such as message setup and transfer times. However, usually only the "real" messages corresponding to actual signal changes in the circuit are included in the analysis. Overhead messages, such as the $NULL$ messages in the conservative model, are typically ignored.

In general, two approaches to the graph partitioning problem are followed. The first approach involves starting with no partition and assigning nodes of the graph to processors following some heuristic rules. The second approach starts with some partition and iteratively tries to improve it. These frequently use one of the first methods to establish a "good" starting partition.

2.1 Initial Partitioning Algorithms

The simplest of these is *random partitioning* whereby nodes are randomly assigned to processors, usually with the constraint of maintaining an equal (balanced) number of nodes on each processor. For complex graphs with dynamic behavior, it has been argued that this may give as good of performance as any other approach and at a much lower computational cost of the partitioning itself [10] [11][12].

One attempt to reduce inter-processor communication, *Simple Data Partitioning (SDP*, weights each node in the graph with the degree of that vertex (the total number of input and output arcs), then assigns nodes to processors successively until the total of the vertix weights is approximately to the average. This tends to group nodes with lots of communication channels on the same processor and reduce the communications between processors [13].

The *Depth-First Breadth-Next (DFBN)* algorithm follows a basic depth-first search of the circuit dependency graph, beginning with the source nodes. Nodes on the same path are assigned to the same processor. This tends to reduce inter-processor communications by combining adjacent nodes that communicate with each other [8].

2.2 Partition Improvement Algorithms

A frequently-cited baseline iterative-improvement algorithm is the *Kernighan-Lin* algorithm [14]. The graph is initially partitioned in an arbitrary but balanced manner. Then nodes are iteratively swapped between processors if the swap reduces the resulting cutset of the graph.

Several researchers have pursued a *simulated annealing* approach, where again an initial partition, often random, is made and nodes swapped between processors if the swap reduces a cost function. However, moves that *don't* reduce the cost are allowed on a random basis where the probability of such a move decreases over time. The cost function is usually some combination of the number of nodes on each processor and the interconnections between processors [13] [15] [16] [17]. Of these, only the Boukerche and Tropper [17] directly address the cost of Chandy-Misra $NULL$ message overhead in their cost model.

3 The Simulation Model

In the problem domain of structural VHDL circuit simulations, there is a direct relationship between the circuit model and the static task allocation problem. The VHDL circuits used in this research are structurally defined using only simple VHDL processes (e.g., simple logic gates). Each circuit is composed of two distinct entities: behaviors and signals. Behaviors are executable VHDL routines representing logic gates, source signals, or other simple VHDL processes. Signals correspond to interconnections between behaviors, corresponding to wires in the physical system.

Parallelism is introduced by partitioning the circuit behaviors (*i. e.* logic gates, source signals, etc.) among the available processors. Each circuit is modeled as a directed graph in which the behaviors correspond to vertices, and the inter-behavior dependencies (signal flows) correspond to directed arcs. Partitioning is accomplished by grouping the behaviors into logical processes (LPs), and assigning each LP to a processor. Behaviors are combined to create a single LP for each processor. Thus each LP is owned by a single processor, and each LP (and therefore each processor) owns a disjoint set of behaviors. Signal changes are shared among LPs by sending event messages over *channels.*

Note that a signal may correspond to more than one inter-behavior dependency (*i. e.* arc). For example, Figure 1 shows a circuit diagram for an edge-triggered D flip-flop and its corresponding directed graph. Both behaviors 4 and 7 are dependent upon the output signal of behavior 5. Thus there are 2 output arcs from behavior 5 – one each to behaviors 4 and 7.

3.1 Simulation Execution Model

The parallel VHDL simulator used in this research has been designed to operate in a single program multiple data (SPMD) mode in which each LP executes an identical copy of the simulation on a disjoint subset of the data (behaviors) [10]. When a behavior is executed, its output signal is placed on an active record list. Signals are removed from the active record list

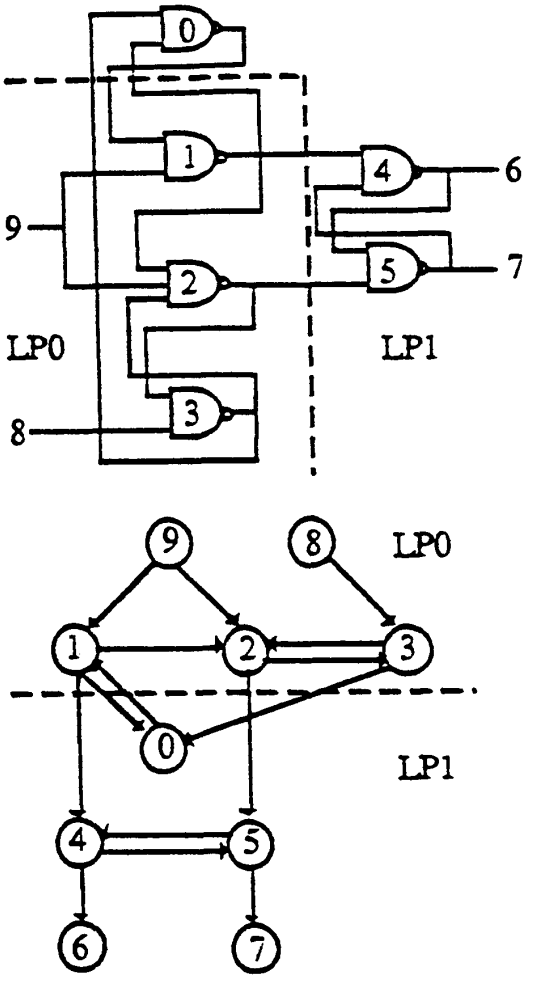

Figure 1: Edge-Triggered D Flip-Flop.

in timestamp order and compared with the previous value of that signal to determine if an actual signal change has occurred. If it has, all behaviors dependent upon the changed signal are subsequently scheduled for execution by placing them on the behavior queue.

In the parallel implementation each *LP* maintains its own active record list and behavior queue as well as its own simulation clock. The clock maintains that *LP*'s local virtual time (LVT), and indicates how far along in the simulation the *LP* has proceeded. When a behavior is scheduled for execution, it is checked against a map to determine its owning *LP*. If the owning *LP* is the local one, then the behavior is added to the behavior queue. If the behavior is owned by an *LP* on a remote processor, an *event message* is sent to the owning *LP* specifying the signal and the time at which it should change. Before removing the next signal from the active record list, all incoming event messages are read and the corresponding signal changes added to the local active record list.

3.2 Synchronization

The simulator used in this research utilizes a *conservative* parallel discrete event simulation synchronization mechanism based on the Chandy-Misra *NULL*-message protocol [1][10]. Deadlock is avoided by sending time-stamped *NULL* messages which do not represent any signal flow in the VHDL model, but pass along the earliest time at which the sending *LP* could send a real message. These *NULL* messages add directly to the inter-processor communications overhead.

*LP*s are connected by directed communication links, or *channels*, each of which may contain many inter-behavior arcs. Each channel between two *LP*s has a time associated with it which represents the time-stamp of the most recent message transmitted over that channel. Messages over a given channel must be sent with monotonically increasing time-stamps. An *LP*'s safe-time, t_{safe}, represents the maximum simulation time to which the *LP* can execute with the guarantee that no messages will be received with earlier time-stamps. The safe-time is determined as the minimum time of all input channels, and is updated each time the *LP* receives a message.

If the earliest signal value on the active record list has a time-stamp greater than the safe time (*i.e.* cannot safely be executed) and there are no event messages from other *LP*s waiting in the input queue, the *LP* must block and wait for an incoming message to advance the safe time. *NULL* messages are used to advance the times associated with each channel (thereby advancing t_{safe}) when there are no "real" event messages available to be sent.

3.3 *NULL* Messages

NULL messages serve as guarantees to the receiving *LP*s that no real events prior to t_{null} will be sent by the sending LP_i. This allows the receiving *LP*s to advance the clock associated with their input channel from LP_i to time t_{null}, which may also allow them to update their safe times, thus avoiding cyclical waiting and preventing deadlock.

In our implementation, *NULL* messages are sent under the following circumstances [10][6]:

- When LP_i sends an event message over one of its output channels with a time-stamp of t_i, it sends a *NULL* message over all other output channels with a time-stamp of t_i to let all downstream *LP*s know that they will not receive any further event messages from LP_i with a timestamp less than t_i. (No *NULL* message is sent over a channel if any outgoing message with timestamp t_i has already been sent over that channel).
- If an *LP* cannot safely process the next signal on the active record list, prior to blocking it sends a *NULL* message over each of its output channels with a time-stamp indicating the earliest that a real event message could be sent. This is calculated to be the earlier of (1) the time of the signal at the top of the active record list or (2) the safe time (which is the earliest an event message from another *LP* could arrive) plus the minimum propogation delay through the *LP*'s portion of the circuit.
- Deadlock is prevented as follows. If an *LP* receives a *NULL* message on one of its input channels while blocked, the corresponding channel time (and possibly t_{safe}) will be updated and the *LP* unblocked. Since the *NULL* is not real, the *LP* will reblock, again sending out *NULL*s based on the updated t_{safe}.

4 Partitioning Cost Model

4.1 Partitioning Requirements

This section analyzes those parameters of the parallel circuit simulation affecting execution time and combines them into a set of cost factors important to the partitioning problem. Most reported partitioning strategies are based on maximizing load balance and minimizing communications between processors. Our model also includes these, addressed in Section 4.2 as the *load imbalance factor* H_b and in Section 4.3 as the *communications factor* H_c. In addition, however, our model takes into consideration the effects of the conservative synchronization *NULL* messages with the *NULL message communications factor* H_n, developed in Section 4.4. In Section 4.5 the cost factors are combined to form an objective function for measuring the relative cost of a partition.

4.2 Load Balancing

Load balancing is the degree to which all available processors are assigned an equal share of the computational load. In parallel VHDL simulations, the computational load consists of processing signal changes, scheduling the affected behaviors, and executing the affected behaviors (which in turn may cause further signal changes). Consider the VHDL simulation to consist of a set B of N behaviors b_i. Since the behaviors here are simple VHDL processes, the load of all behaviors can be considered equal. Let W represent the computational cost of behavior b_i $(1 \leq i \leq N)$. In the parallel partitioning over n logical processors, set B is divided into n mutually exclusive subsets B_j $(1 \leq j \leq n)$, containing N_j behaviors ($\sum N_j = N$ and $\cup B_j = B$ and $\cap B_j = \{\ \}$) where each set B_j is assigned to logical process LP_j. The computational load L_j for LP_j can then be calculated as:

$$L_j = \sum_{b_i \in B_j} W = WN_j \tag{1}$$

The load imbalance factor, H_b, is then defined to be:

$$H_b = \frac{L_{max} - L_{avg}}{L_{avg}} \tag{2}$$

where

$$L_{max} = \max_{1 \leq j \leq n} L_j = W \max_{1 \leq j \leq n} N_j \tag{3}$$

and

$$L_{avg} = \frac{1}{n} \sum_{j=1}^{n} L_j = \frac{WN}{n} \tag{4}$$

The load imbalance factor then becomes:

$$H_b = \frac{W \max_j N_j - \frac{WN}{n}}{\frac{WN}{n}} = \frac{n(\max_j N_j) - N}{N} \tag{5}$$

4.3 Real Message Overhead

Inter-process communications can be represented by a *communications weight matrix* C, where each element c_{ij} represents the total cost of directed communications from LP_i to LP_j. If we assume the transmission cost between any two *LP*s to be constant, then $c_{i,j}$ represents the number of messages between those *LP*s. However, since there is no *a priori* information available regarding signal activity, some means of estimating the relative frequency of communication between *LP*s is needed. Since each channel between any two *LP*s can correspond to many inter-behavior arcs (wires), c_{ij} is defined as the number of such inter-behavior dependencies (*i.e.* arcs on the dependency graph) from LP_i to LP_j.

For n logical processes, C is an n x n matrix where the main diagonal entries represent the cost of each *LP*'s communications with itself. Since self-communication does not contribute to the parallel simulation communications overhead, all main diagonal entries are set to zero. The communications factor H_c, representing the overall total inter-process communications costs, can be calculated as the sum of all the entries in the communications weight matrix.

$$H_c = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \tag{6}$$

4.4 *NULL* Message Overhead

The conservative synchronization protocol discussed in Section 3.2 adds additional communications overhead in the form of *NULL* messages. Experience with the VHDL simulations used in this research has shown that the ratio of *NULL* messages to real messages grows with the number of processors, a fact that cannot be ignored when making partitioning decisions. The equation for the inter-processor communications cost factor H_c only accounts for the real message traffic corresponding to actual signal change events in the simulation. Due to the potential dominance of the interprocessor communications by *NULL* messages, an additional cost factor is needed to take this sizable overhead into account.

Analysis of the rules for sending *NULL* messages given in Section 3.3 indicates that the number of *NULL* messages transmitted is heavily dependent on the total number of channels in the *LP* interconnectivity graph, denoted by n_{chan}. This is equal to the number of non-zero entries in the communications weight matrix. With n *LP*s, each *LP* can be connected to at most $n-1$ other *LP*s. Also, assuming that the *LP*s are fully connected, then there are at least $n-1$ channels. Thus, the value of n_{chan} is bounded by:

$$n - 1 \leq n_{chan} \leq n(n-1) \tag{7}$$

Clearly n_{chan} is an indication of the cost of sending *NULL* messages. Reducing the number of inter-*LP* channels should correspondingly reduce the *NULL* message overhead. Note that the effects of "lookahead" could also be considered, based on finding the minimum path through each *LP* as a limit on how soon an "arriving" signal change could generate an output message. Unfortunately, determining the minimum path through each partition of the circuit for each round of our iterative algorithm (Section 5) is too computationally intensive to be practical for large circuits. Thus for partitioning purposes lookahead is ignored, even though it *is* used in the simulation itself. The resulting value of the *NULL* message communications factor H_n is then given by:

$$H_n = n_{chan} \tag{8}$$

4.5 Objective Cost Function

Intuitively, there is a natural conflict between the partitioning goals of minimizing the inter-process communications costs and assigning each LP an equal share of the computation load. The primary goal of any partitioning algorithm is to strike a balance between these two conflicting goals that results in maximizing useful parallel computation. This typically results in an objective cost function of the form [15][13]:

$$H = H_c + \alpha H_b \tag{9}$$

Simulation results have shown that it is possible to modify the partition such that this cost function predicts an improvement in simulation performance, only to find that actual simulation performance is worse due to the effects of *NULL* message overhead [6]. To account for this deficiency, the objective cost function proposed in this research uses all of the cost factors discussed in Section 4.1. Specifically, the objective cost function H is defined as:

$$H = H_n H_c + \alpha H_b \tag{10}$$

where α is a proportionality coefficient which controls the relative influence of the communications and load balancing portions of the equation. Currently this must be determined empirically for each circuit modeled.

5 Partitioning Algorithms

A graph-partitioning tool was developed that can read in a VHDL circuit graph and partition it by one of several algorithms, resulting in a mapping of behaviors to *LP*s that is subsequently used in the parallel simulation itself. Four different partitioning algorithms were investigated in this research. The first is a simple random partition in which graph nodes are randomly assigned to *LP*s while keeping the number on each *LP* balanced [10]. The second approach uses a simple depth-first (SDF) partitioning of the behavior graph [6]. The third technique performs a simple breadth-first (SBF) partitioning [6]. None of these three heuristics utilizes the cost model, but were included for comparison purposes as well as for establishing an initial partition for the fourth approach. There an iterative improvement algorithm moves behaviors between *LP*s and uses the cost function to determine whether the move is an improvement or would cause performance to degrade, in which case the move is rejected [6]. The search space is limited by considering only "border" behaviors (those that have more connections to another *LP* than to behaviors in the same *LP*) as candidates to move. The number of iterations is limited to a user-specified maximum or until no more improvement in the cost function is possible.

In order to evaluate the cost function, values of the parameters are calculated from the graph on each iteration. However, the value of the weighting coefficient α had to be provided for each circuit tested. The coefficient value was selected to provide the desired relative weightings of the communcations and load imbalance portions of the objective cost function. The first step was to examine the expected magnitudes of the communications cost sub-function ($H_n H_c$) and the load imbalance sub-function (H_b) for a typical example (e.g. the wallace tree multiplier). Because of the methodology used to represent the cost factors, the communications cost sub-function typically produces a much larger value than the load imbalance sub-function. Therefore, after comparing the resulting expected values against the actual curves a value was chosen.

6 Experimental Environment

The parallel VHDL simulations of an 8-bit wallace tree multiplier circuit and a 16-bit x 16-bit associative memory array were used to validate the objective cost function developed in this paper. The wallace tree multiplier consists of 1,050 behaviors and 1,770 inter-behavior arcs. The associative memory array consists

of 4,243 behaviors and 9,312 inter-behavior arcs. Four different algorithms were used to partition each circuit. Specifically, each circuit was simulated with a random partition [10], a simple depth-first (SDF) partition [6], a simple breadth-first (SBF) partition [6], and a partition created using an iterative improvement algorithm [6].

Performance measurements were taken on an 8-node Intel iPSC/2 hypercube. All speedup calculations use a single-LP partition as the performance baseline. Each circuit was simulated with 2, 3, 4, 5, 6, 7, and 8 LPs for each of the four partition types. Only one logical process (LP) was assigned to each physical processor in order to eliminates side effects that would be caused from context switching and message passing between LPs on the same processor. Simulation output was turned off during the collection of timing data to minimize the influence of I/O bottlenecks of the target hypercube platform.

7 Experimental Results

Figure 2 compares the partition statistics values of inter-LP arcs and LP output lines with the corresponding component of the inter-processor communications (*i.e.*, real and null messages transmitted, respectively). Curves are shown for all four partition types. The top portion of Figure 2 clearly shows the relationship between the number of inter-behavior dependencies which cross the LP boundaries and the number of real event messages transmitted between LPs during the simulation.

Similarly, the bottom portion of Figure 2 clearly shows the relationship between the number of LP-level dependencies and the number of null messages transmitted during the simulation. It should be noted that the reduction in real message traffic and an increase in the average lookahead values contributed to the reduction in null message traffic experienced by the deliberate partitioning strategies of SDF, SBF, and border annealing. However, analysis of the simulation results show that these effects are negligible due to the dominating influence of the number of LP output lines on the number of null messages [6].

Comparing the relative magnitudes of the real message and null message curves of Figure 2, it is clear that the inter-processor message traffic is dominated by null messages for more than 2-4 LPs. It is also interesting to note that the slope of the real message curve tends to decrease as the number of LPs is increased, while the slope of the null message curve tends to increase. This indicates that the dominance of the inter-processor message traffic by null messages is likely to worsen as additional LPs are added. Similar curves were obtained with the associative memory circuit [6]. This dominating influence of the null messages highlights the importance of accounting for null messages when comparing the relative costs of various partitions.

The primary objective in measuring the cost of a partition using a cost function is to have an objective means of comparing the relative quality of different partitions. Thus the partition based on the cost function should show improved performance over those not

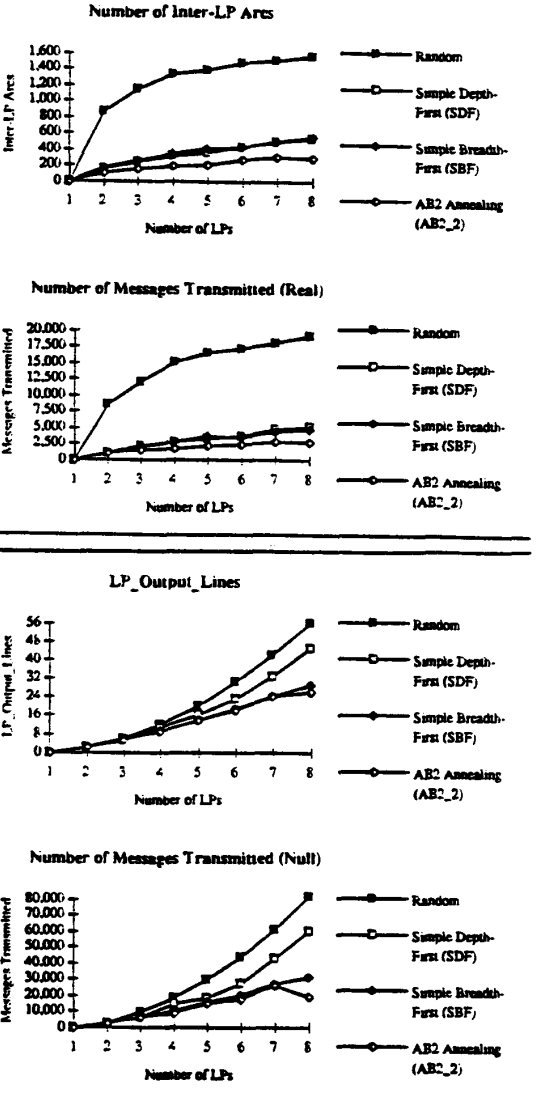

Figure 2: Wallace Tree Message Traffic Analysis.

using it. Figure 3 shows the actual speedup curves for the wallace tree multiplier circuit for all four partition

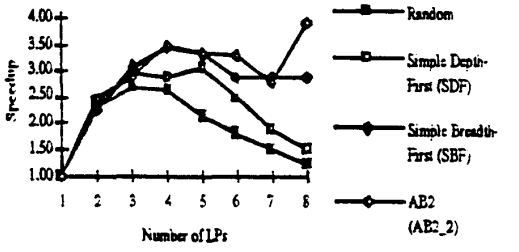

Figure 3: Wallace Tree Speedup Results.

types. Similarly, Figure 4 shows an identical set of graphs for the associative memory array circuit.

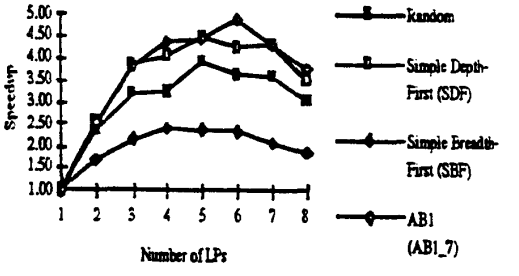

Figure 4: Associative Memory Speedup Results.

8 Analysis and Conclusions

The following conclusions can be made about the objective partition cost function proposed in this research:

- *The partition cost function must account for more than load imbalance and the number of inter-behavior arcs between LPs.* Data analysis shows that as the number of *LPs* is increased, the *NULL* message traffic due to the conservative synchronization protocol begins to dominate the inter-processor communications overhead. Reducing the real message traffic by reducing the inter-behavior arcs which cross LP boundaries serves to increase the dominance of the *NULL* message overhead. Not accounting for the *NULL* message overhead will give an inaccurate picture of the quality of a given partition. Therefore, *NULL message synchronization overhead must be accounted for in the partition cost.*

- *NULL message overhead is proportional to the number of channels in the LP connectivity graph.* Results have shown that the number of null messages required to maintain synchronization is directly proportional to the number of channels in the LP connectivity graph (referred to as "*LP* output lines"). By decreasing the connectivity between *LPs* using a deliberate partitioning scheme, it is possible to reduce the null message overhead and improve simulation performance.

- *Further reductions in real message overhead will have negligible impact on simulation performance.* The deliberate partitioning algorithms used in this research have made significant reductions in the amount of real message communications overhead compared to a random partitioning of the circuit behaviors. The null message overhead is also reduced, but by a much smaller margin. As a result, the inter-processor communications overhead is dominated by the null message traffic for a relatively small number of LPs. The problem is exacerbated as the number of LPs is increased. Further reductions in real message traffic without significant reductions in null message traffic will have a negligible impact on the total inter-processor message traffic.

- As shown in Figures 3 and 4, for the two large circuits analyzed, the AB2 algorithm, which used the cost function proposed in this paper, showed improved speedup over one through eight processors when compared to the other partitioning approaches.

9 Summary and Future Work

This paper describes an improved cost function for partitioning VHDL circuit simulations across parallel processors. The improvement is due to accounting for synchronization protocol *NULL* messages in the cost function. Tests on two large circuits show that the use of this cost function with a simple iterative improvement algorithm produces a partition with better speedup than other partitioning algorithms.

There are several areas for continued research. Although we have included the effect of *NULL* messages in our model, it is not clear that we have done so in the best way. Other forms of including that information are being considered. Also, the Chandy-Misra protocol can benefit by using *lookahead* to predict the time of the next message [18]. While our simulator uses this approach, we have not included the effect of lookahead in our cost model due to computational efficiency. However, we feel that such information is critical to providing a partition that provides the best lookahead. Finally, our cost model has only been tested in a simple iterative loop. We are planning to expand the tool to use a simulated annealing loop in hopes of getting an even better partition.

Acknowledgements

We would like to thank our sponsor, the ARPA QUEST project, for their support. We also wish to

acknowledge several prior students without whose help the current research could not have taken place: Mike Proicou [19], Ron Comeau [20], and Tom Breeden [10]. We wish to thank Eric Christensen, U. S. Army, for his initial version of the graph partitioning tool. Finally, we would like to thank John Hines of the U. S. Air Force Avionics Laboratory for providing the original VHDL compiler as the basis for our parallel version.

References

[1] K. Chandy and J. Misra. "Asynchronous distributed simulation via a sequence of parallel computations," *Communications of the ACM*, vol. 24, pp. 198–206, April 1981.

[2] D. R. Jefferson, "Virtual time," *ACM Transactions on Programming Languages and Systems*, vol. 7, pp. 404–425, July 1985.

[3] H. Carter, R.Vemuri, P. A. Wilsey, J.Aylor, R. Waxman, and T. Hartrum, "High speed acceleration of vhdl simulation, synthesis, and atpg: Overview of the quest project," in *Spring 1991 VHDL Users' Group*, pp. 85–90, April 1991.

[4] C. Sporrer and H. Bauer, "Corolla partitioning for distributed logic simulation of vlsi-circuits," in *7th Workshop on Parallel and Distributed Simulation (PADS93)*, (San Diego, CA), pp. 85–92, July 1993.

[5] R. D. Chamberlain and C. D. Henderson, "Evaluating the use of pre-simulation in vlsi circuit partitioning," in *8th Workshop on Parallel and Distributed Simulation (PADS94)*, (Edinburgh, Scotland), pp. 139–146, Jul 1994.

[6] K. L. Kapp, "Partitioning structural vhdl simulations for parallel execution on hypercubes," Master's thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1993. AFIT/GCE/ENG/93D-7.

[7] K. Schwan *et al.*, "Mapping parallel applications to a hypercube," in *Proceedings of the Second Conference on Hypercube Multiprocessors*, pp. 141–151, 1987.

[8] S. Manoharan and P. Thanisch, "Assigning dependency graphs onto processor networks," *Parallel Computing*, vol. 17, pp. 63–73, 1991.

[9] P. Sadayappan and F. Ercal, "Nearest-neighbor mapping of finite element graphs onto processor meshes," *IEEE Transactions on Computers*, vol. C-36, pp. 1408–1424, December 1987.

[10] T. A. Breeden, "Parallel simulation of structural vhdl circuits on intel hypercubes," Master's thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1992. AFIT/GCE/ENG/92D-1.

[11] S. P. Smith, M. R. Mercer, and B. Underwood, "An analysis of several approaches to circuit partitioning for parallel logic simulation," in *International Conference on Computer Design*, pp. 664–6676, 1987.

[12] S. A. Kravitz and B. D. Ackland, "Static vs dynamic partitioning of circuits for a mos timing simulator on a message-based multiprocessor." in *Distributed Simulation*, pp. 138–1406, 1988.

[13] J. M. Conrad and D. P. Agrawal, "A graph partitioning based load balancing strategy for a distributed memory machine," in *Proceedings of the 1992 International Conference on Parallel Processing, Vol II*, pp. 74–81, August 1992.

[14] B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, vol. 49, no. 1, pp. 291–307, 1970.

[15] T. Bultan and C. Aykanat, "A new mapping heuristic based on mean field annealing," *Journal of Parallel and Distributed Computing*, vol. 16, pp. 292–305, December 1992.

[16] E. E. Witte, R. D. Chamberlain, and M. A. Franklin, "Parallel simulated annealing using speculative computation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, pp. 483–494, October 1991.

[17] A. Boukerche and C. Tropper, "A static partitioning and mapping algorithm for conservative parallel simulations," in *8th Workshop on Parallel and Distributed Simulation (PADS94)*, (Edinburgh, Scotland), pp. 164–172, Jul 1994.

[18] R. M. Fujimoto, "Parallel discrete event simulation," in *Proceedings of the 1989 Winter Simulation Conference*, pp. 1–34, 1989.

[19] M. C. Proicou, "A distributed kernal for simulation of the vhsic hardware description language," Master's thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1989. AFIT/GCS/ENG/89D-14.

[20] R. C. Comeau, "Parallel implementation of vhdl simulations on the intel ipsc/2 hypercube," Master's thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1991. AFIT/GCS/ENG/91D-03.