Parallel Processing Letters

Performance Evaluation of a Parallel Decoupled Data Driven Multiprocessor

MITCHELL THORNTON
Department of Computer Science and Engineering, Southern Methodist University
P. O. Box 750122, Dallas, Texas 75275, U. S. A.

Received (received date)
Revised (revised date)
Communicated by (Name of Editor)

ABSTRACT

The Decoupled Data-Driven (D^3) architecture has shown promising results from performance evaluations based upon deterministic simulations. This paper provides performance evaluations of the D^3 architecture through the formulation and analysis of a stochastic model. The D^3 architecture is a hybrid control/dataflow approach that takes advantage of inherent parallelism present in a program by dynamically scheduling program threads based on data availability and it also takes advantage of locality through the use of conventional processing elements that execute the program threads. The model is validated by comparing the deterministic and stochastic model responses. After model validation, various input parameters are varied such as the number of available processing elements and average threadlength, then the performance of the architecture is evaluated. The stochastic model is based upon a closed queueing network and utilizes the concepts of available parallelism and virtual queues in order to be reduced to a Markovian system. Experiments with varying computation engine threadlengths and communication latencies indicate a high degree of tolerance with respect to exploited parallelism.

Keywords: Performance Modeling, Data-Driven Execution, Markov Chain, Deterministic Model

1 Introduction

The dataflow model of computation promises to exploit parallelism through asynchronous instruction execution on the basis of operand availability. Several different architectural approaches have been proposed and evaluated [1] [2]. The dataflow model of execution has been criticized for unequal load balancing and complicated resource management issues in general. In addition, scheduling issues have prompted some criticism of this computation model. Furthermore, the dataflow model of execution is unable to benefit from the many advancements in the design of contemporary control-flow based machines that are based upon the exploitation of locality such as pipelining, caching, and delayed accesses and branching.

In order to benefit from both the exploitation of locality that many control-flow machines utilize and the use of inherent parallelism found in dataflow machines, there has been some interest in hybrid control-flow/dataflow architectures recently [3] [4]. Behavioral simulations have been performed and promising results have been obtained for the Decoupled Data-Driven (D^3)architecture [5]. This paper will address the development and results obtained of a queuing network model of the D^3 architecture. This model is verified by comparing its results to those obtained using the behavioral architectural model. After verification, various aspects of the architecture may be easily varied by changing certain parameters of the queuing network allowing the system designer to experiment with architectural modifications without resorting to changing the behavioral model.

The use of analytical models for estimating the performance of various computer architectures and their resource requirements has been studied by other researchers. In [6] FORTRAN industrial applications were were compiled using the Polaris compiler and key parameters were extracted. These parameters were used to construct symbolic functions that described function complexity with respect to data and runtime resources and to extrapolate on how these resource scale. A stochastic modeling approach was used to characterize dataflow graphs for the reliability analysis of interconnection and computer networks is described in [7]. A hybrid computer architecture that is extremely similar to the D^3 approach described here is described in [8] and both deterministic and analytic models are used to characterize its' performance.

First, a brief description of the architecture is presented so that the appropriateness of the model may be described and the limitations can be pointed out. Next, assumptions are presented in order to simplify the model so that it can be analyzed in a practical manner. The notions of 'available parallelism' and 'virtual acknowledgment queue' are presented and incorporated into the stochastic model. After the model is formulated, it is verified by comparing modeling results with those obtained by computer simulation. Performance analysis measures are then obtained by varying parameters such as available parallelism, processor communication latencies, and average lengths of the control-flow code templates (hereafter referred to as 'threadlength').

2 The D^3 Architecture

The D^3 machine is based upon the hybrid combination of von Neumann style (control-flow) processors and data-driven synchronization. The synchronization of a von Neumann machine is based purely upon the program counter (PC) that points to the next instruction to be executed in the control-flow of the program. The D^3 architecture also uses a PC within a particular thread, but the threads themselves are scheduled according to operand availability. This allows the benefits of locality to be utilized within each thread and the exploitation of inherent parallelism in a particular program due to data-driven scheduling of the threads. An overall view of a single node in the D^3 architecture is given in Figure 1.

Figure 1 shows how the scheduling of threads and the thread computation occur

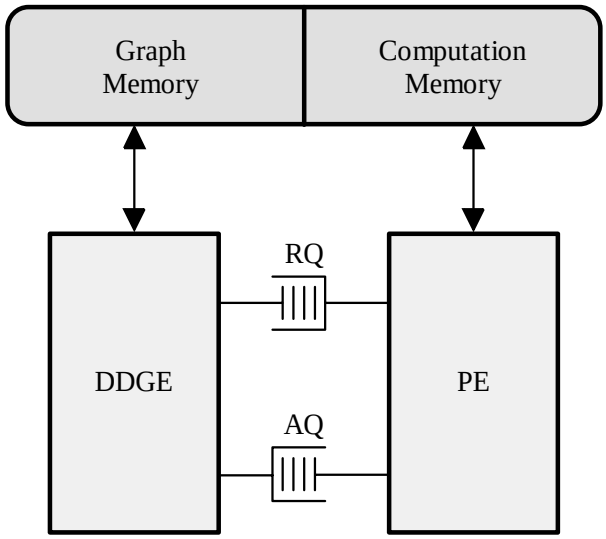

Fig. 1: Diagram of a Single Node in the D^3 Architecture

concurrently by using the data-driven graph engine (DDGE) to perform the synchronization of thread execution and using the processor engine (PE) to perform the control-flow execution of threads. This allows for increased tolerance to scheduling overhead and communication latency. The DDGE and PE are connected by two queues, the ready queue (RQ) and the acknowledgment queue (AQ). As soon as a thread (or actor) has completed execution, the computational element (CE) within the PE is freed and a pointer to the particular code template along with a context value is entered into the AQ. The freed CE is then allocated a new thread and execution begins.

The objects in the AQ are serviced by individual graph elements (GEs) within the DDGE. The GE examines the pointer in the AQ and determines which actor (or actors) require the operand just computed. These actors may be referred to as "consumers" of the actor that was retrieved from the AQ and is currently being processed by a particular GE. If the consumers only require one operand they are called monadic actors and pointers to their respective code templates are placed into the RQ indicating computation may begin. If the consumers require a second operand, they are not scheduled for execution but must "wait" for their remaining operand. This "waiting" is accomplished by using graph memory that contains a "status" value for each actor in the program. Dyadic actors have a status value of 2 and monadic actors have a status value of 1. As soon as a producer arrives in the AQ, the GE processes this actor by decrementing the status values of all its consumers. Whenever a status value reaches 0, this actor is deemed executable and is scheduled by placing a pointer to its code template into the RQ.

It is important to note that servicing by the DDGE is always constant. The

only way for non-constant service to occur is if the number of servers (GEs) is less than the number of actors in the AQ. When this occurs, the actors must wait in the queue.

In contrast, the service time of the PE is variable. In the model for the D^3 architecture, it is assumed that all control-flow instructions execute in unity time, hence the service time is equal to the threadlength.

The programs for the D^3 architecture are composed of two main entities; the graph (synchronization) portion and the computational portion. The computation portion consists of segments of code composed of conventional, control-flow instructions such as ADD, LOAD, STORE, etc. The graph portion contains the status value and the consumers. Collectively, the graph portion and the computation portion are called an actor. Thus a D^3 program (or D^3 graph) is a partially ordered conventional program with a data-dependency graph superimposed on it.

3 Formulation of the Stochastic Model

In order to model the D^3 architecture both the hardware and the particular data-driven graph to be executed must be accounted for. As discussed above the architecture is conveniently represented as a computation engine and a graph execution engine connected by two queues. The two engines are modeled as multi-servers for the two queues.

The RQ server model is very straight forward and obeys typical server constraints. The queue objects (i.e., the actors) in the RQ are pointers to conventional control-flow code templates of varying threadlength. The service time of the RQ objects is equal to the number of operations in each template since it is assumed that all control-flow instructions require unity execution time. The threadlength is modeled as being exponentially distributed with some mean value. Thus, the RQ is modeled as a multi-server queue with an exponential service time. The particular data-driven graph that is being modeled is used to compute sample statistics such as average threadlength and the percent of dyadic actors. The average threadlength is then used as the mean service time for the RQ.

The AQ is modeled as two parallel queues referred to as "AQ" and "VAQ". The VAQ (virtual AQ) is not present in the architecture. Its purpose is to simulate the action of dyadic actors. Dyadic actors require two operands before they can execute. This means that when an RQ server is finished, the consumer of that actor may require another operand before it can be scheduled for execution, hence the consumer is not immediately scheduled for execution. The consumer must "wait" for its other operand before it can be executed. In the architecture, this "waiting" is accomplished by decrementing the status variable of an actor for its respective context. In order to comply with a closed queuing network model for the D^3 architecture, the VAQ is used to service the dyadic actors. By observing simulation results for various D^3 graphs, it seemed appropriate to model the VAQ service time as being exponentially distributed with a mean service time determined from the particular program being executed. The other queue, AQ, is a multi-server queue with a constant service period. This directly represents the current state of the

architecture. Communication latencies among the GEs may be varied by changing the service time of the AQ. When an actor has been served by the RQ, it is placed in either the AQ or VAQ depending upon the outcome of a uniform random number generator and the input parameter that states the percentage of dyadic actors. This input parameter is essentially a routing probability in the network for arrival at the VAQ.

It should be noted that in reality all actors are not either monadic or dyadic. It is possible for actor to require an arbitrary number of operands before it can be scheduled for execution. However, such actors can always be transformed into a set of smaller actors that are all monadic or dyadic in nature.

In order to formulate the model as a Markovian system, the notion of available parallelism is used. In other performance analysis studies, this parameter has been computed in a variety of ways [9] [10]. In this paper a very simple representation for available parallelism is used. Encouraging results are obtained by setting the available parallelism input equal to the total number of actors in a given context for the particular D^3 graph under consideration. The available parallelism parameter is used during initialization of the model to determine the total number of actors present in the network.

Since the total number of actors is fixed, the network is conveniently modeled as a Markov chain with the states determined by the number of actors being served in each queue. These states are denoted by:

$$(n_1, n_2, n_3)$$

Where

1. n_1 is the number of actors being served in the RQ
2. n_2 is the number of actors being served in the AQ
3. n_3 is the number of actors being served in the VAQ
4. N_{tot} is the total number of actors in the system, or alternatively, the input parameter, available parallelism
5. p_{dya} is the percentage of dyadic actors for a particular D^3 graph

Clearly,

$$N_{tot} = n_1 + n_2 + n_3$$

Using this relationship and the definitions above, the following equation can be written that represents the relationship for the Markov chain state transitions:

$$N_{tot} = n_1 + n_2 + p_{dya}(n_3 - n_2)$$

Figure 2 shows a schematic representation of the stochastic model. The number of states in the Markov chain depend upon the value of N_{tot}. A general state of the Markov chain is shown in figure 3. The values of m, k, and j are in general

greater than one. This is especially true since an infinite number of servers are assumed to be present in the model. In reality any physical realization of the D^3 architecture will necessarily contain a finite number of servers, however when the available parallelism is less than the number of servers at each queue, the behavior of the queues is identical to queues with infinite servers. Allowing infinite servers in a model of the D^3 architecture allows the inherent parallelism present in the D^3 graph to fully exploited and thus a convenient measure of this performance characteristic is obtained. For this reason, both the stochastic model and the simulation model offer this capability.

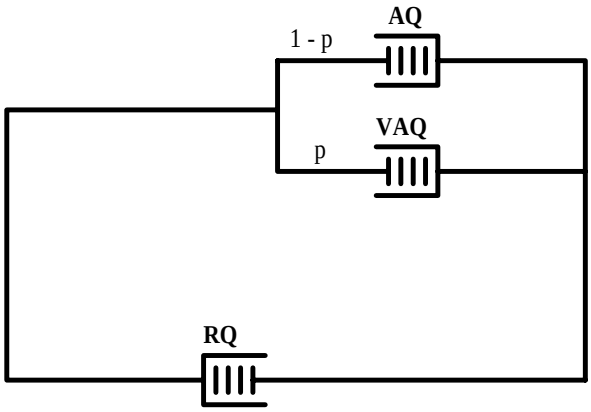

Fig. 2: Diagram of the Closed Queuing Network Model for the D^3 Architecture

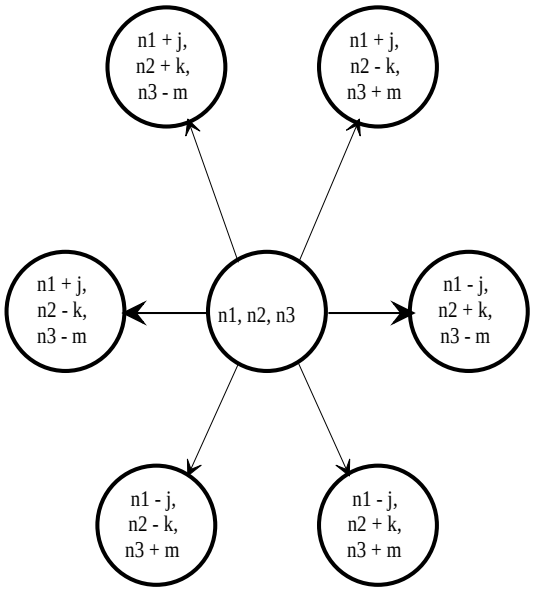

Fig. 3: General State for the Markov Chain model

Table 1: Comparison of Simulation and Stochastic Model Results

D^3 Graph	Simulator Results		Stochastic Model Results	
	Avg. CE	Avg. GE	Avg. CE	Avg. GE
DOT	2.7	0.54	2.7	0.72
TRAP	5.7	0.21	7.5	0.68
FIBONACCI	2.0	0.27	1.6	0.16
JACOBI	2.4	0.39	2.4	0.29

Since the RQ has an exponential service rate, the arrival rates of the AQ and VAQ are approximately Poisson and the queues are of the following type:

1. RQ - M/M/c
2. AQ - M/D/c
3. VAQ - M/M/c

4 Model Results

The results of the model were obtained by using the simulation language, SIMSCRIPT [11]. After the model was developed, it was verified by comparing the results obtained to those obtained by the simulator. There are several different D^3 graphs executing on the simulator and 4 of these were chosen to validate the stochastic model with. Table 1 compares results obtained from the simulator and the stochastic model as implemented with SIMSCRIPT.

These results indicate an amount of agreement between the stochastic model and the computer simulation of the architecture as quantified by an average agreement of required computation CEs being 0.55 and an average agreement of required GEs being 0.23. Based on these results, experiments were performed with various changes in the input parameters such as available parallelism and communication latency to analyze the performance of the D^3 architecture. The largest deviation in results between the simulator and the stochastic model occurred for the graph that executes the Trapezoid rule for an integrand that is a third degree polynomial. This is most likely due to the model parameter of average thread length (i.e., mean RQ server time). This particular graph has several actors with threadlengths around 40 and several with threadlengths less than ten, hence the average value is around 25, but in reality there are no actors with a thread length of this size. So, the exponential distribution of service time for the RQ is a very approximate assumption for this particular example. A more accurate stochastic model could be formulated for the Trapezoid rule experiment by using an empirically measured distribution from the deterministic model; however, this would defeat the purpose of attempting to obtain a generalized model for all program examples. Furthermore, when parameters such as available parallelism are extrapolated, it is not clear that the empirically measured distributions would scale proportionally. For these reasons,

the results reported here are based on using exponential distributions with average threadlengths.

To determine how well the architecture exploits the inherent parallelism in a particular D^3 graph, the parameters from one of the examples above was held constant and the available parallelism parameter was varied. This is equivalent to assuming the same communication latencies (queue arrival rates) and threadlengths (RQ mean service times) as in the original example, but assuming that more or less parallelism is available in the D^3 graph being executed. The plot in figure 4 illustrates how the resource utilization versus the available parallelism varies for the DOT example. The results indicate that resource utilization grows linearly with respect to available parallelism with a nearly unity slope for the maximum number of CE's used. The ideal case would be a slope of 1.0 for this curve indicating 100% exploitation of available parallelism. The stochastic model seemed to be more optimistic with this parameter than the simulator did. Typically, the simulator would indicate a slope of approximately 0.6 to 0.7 while the stochastic model would predict results from 0.7 to 0.9. This is most likely due to the absence of actors requiring more than two operands before being scheduled for execution and the exponential service time distributions.

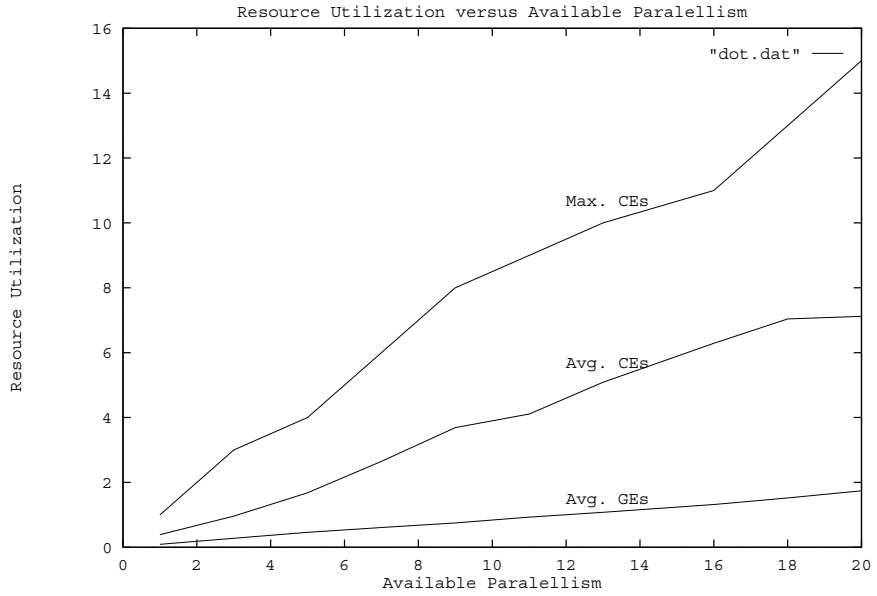

Fig. 4: Available Parallelism versus Resource Utilization for the DOT example

Another interesting result was obtained by varying the average thread length (the mean RQ service time) and observing the effect on resource utilization. These results are displayed in the plot shown in figure 5. This experiment represents trading off the two virtues of increased performance due to locality in the PE and increased performance due to inherent parallelism in the data-driven graph. The results for the FIBONACCI generator example indicated that increasing the

threadlength by 100% degraded parallelism by only 33%. This result is encouraging because it means that greater performance can be achieved by exploiting locality with a minimal resulting degradation due to decreased parallelism. In the limit, the average threadlength would approach the time required to compute the entire program as an unrolled loop. The data in this portion of the graph could give interesting information about the amount of locality being exploited and the amount of inherent parallelism that could be utilized.

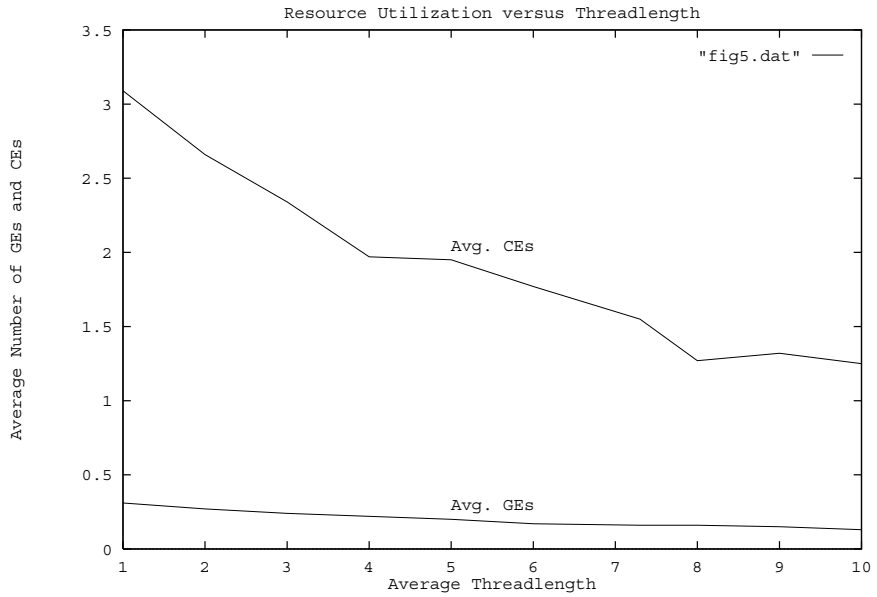

Fig. 5: Average parallelism versus Average Threadlength for the FIBONACCI Example

To study the effects of communication latency on the D^3 architecture, the constant service time of the AQ is varied for a constant available parallelism and mean RQ service time. The results of this experiment are given in figure 6. It is desirable for this curve to be as flat as possible indicating a high degree of tolerance to increased communication latencies. It should be noted that another way of combating the effects of increased communication latencies is to increase the threadlength of the computation code templates thereby gaining more computation per scheduling activity. For the purposes of the stochastic model, the threadlength remained constant for this experiment.

The results in figure 6 the best obtained so far over our suite of example programs. The average number of CEs remained nearly constant when the communication latency was increased by a factor of 10. The communication latency resulted in an increased average number of executing GEs. This is precisely why the architecture is designed in a "decoupled" manner. This result clearly shows how the D^3 architecture tolerates communication latencies by letting the DDGE handle the degradation and allowing for practically no computational degradation.

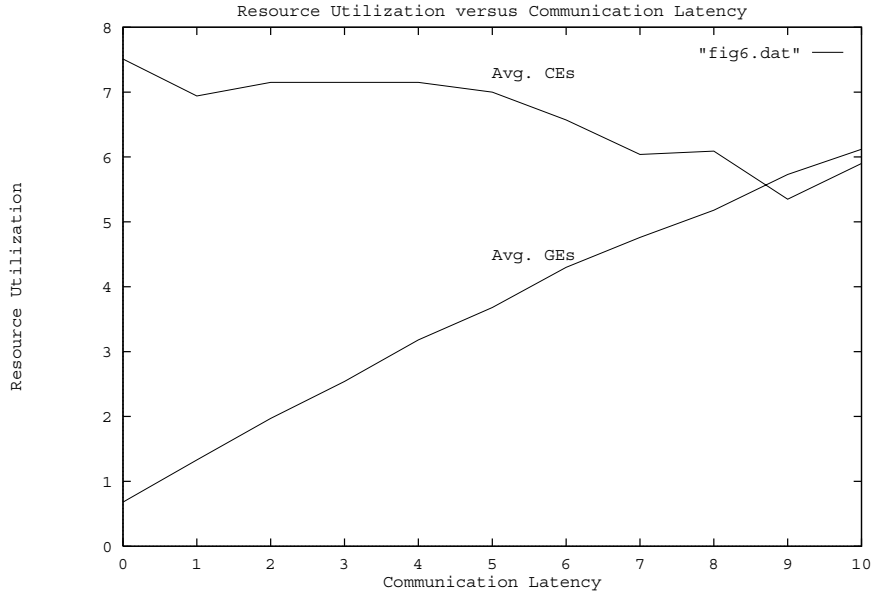

Fig. 6: Communication Latency versus Resource Utilization TRAPEZOID Rule Example

5 Conclusion

This paper has described the formulation and implementation of a stochastic model for the D^3 architecture. The model was constructed as a closed network of queues. Currently, the network model is very simple but encouraging results have been obtained nevertheless. Results were obtained from this model and were shown to have a close agreement with results obtained from a computer simulation of the architecture. Various performance analysis measures were generated using the stochastic model. The results indicated that the D^3 architecture has a high degree of tolerance to communication latency and is also able to achieve a good amount of exploitation of available parallelism.

1. Arvind and Gostelow, K.P., The U-Interpreter, *IEEE Computer*, **pp. 42-49**, February, 1882.
2. Dennis, J.D., Data flow Supercomputers,*IEEE Computer*, **C-29(11):48-56**, November, 1980.
3. Ianucci, R.A., Toward a Dataflow/von Neumann hybrid architecture, *Proceedings of the* 15^{th} *Annual International Symposium on Computer Architecture*
4. Evripidou, P. and Gaudiot, J.L., A Decoupled Graph/Computation Data-Driven Architecture with Variable-Resolution Actors, *1990 International Conference on Parallel Processing.*
5. Evripidou, P., Thornton,M., and Gaudiot, J.L., A Decoupled Data-Driven Machine with Variable-length Thread Support, *Tech. Rep. 93-CSE-29*, SMU, 1993.
6. Armstrong, B. and Eigenmann, R., Performance Forecasting: A Methodology for Characterizing Large Computational Applications, *Proceedings of the International Conference on Parallel Processing*, pp. 518-525, August 1998.
7. Chen, D. J. and Kavi, K. M., Stochastic Dataflow Graph Models for the Reliability

Analysis of Interconnection and Computer Networks, *Journal of Information Science and Engineering, (Institute of Information Science)*, **7(2):253-278**, June 1991.

8. Kavi, K. M., Giorgi, R. and Arul, J., Comparing Execution Performance of Scheduled Dataflow with RISC Processors, *Proceedings of the* 13^{th} *International Symposium of Computer Architecture Parallel and Distributed Computing Conference*, August 2000.
9. Ghosal, D. and Bhuyan, L.N. Performance Evaluation of a Dataflow Architecture, *IEEE Trans. on Comp.* **vol. 39 no. 5**, May 1990, pp.615-627.
10. Houghton, R.C., Jr., Performance Evaluation of Task Graphs on Parallel Architectures **Ph.D. Dissertation, Duke University**, *Department of Computer Science*, 1991.
11. Russell, E.C., **SIMSCRIPT II.5 Programming language**, *4-th Edition, CACI Products Company, LaJolla, CA*, 1987.