Using a Jini based Desktop Grid for Test Vector Compaction and a Refined Economic Model*

Tezaswi Raja
Rutgers University, Dept. of ECE
Piscataway, NJ 08854, USA
tezaswir@caip.rutgers.edu

Manish Parashar
Rutgers University, Dept. of ECE
Piscataway, NJ 08854, USA
parashar@caip.rutgers.edu

Abstract

Testing of Very Large Scale Integrated(VLSI) circuits is done by designing huge random sets of vectors of which only a few are useful. The process of filtering these good vectors from the overall set is called vector compaction. *As the integrated circuits become denser, this problem is becoming a major bottleneck and the computation time could run into days for a single chip. In this paper we demonstrate how a distributed Grid architecture can be used for the speed-up of the problem. The architecture is based on a Jini Desktop Grid. Economic models become a prime issue in such a scenario. The current economic models for minimizing cost or time are ad-hoc and entirely under the control of the broker middle-ware architecture, leaving the end user with little choice. In this paper we present a revised economic model that gives more choice to the user in terms of time and cost before execution. We match the high-level application layer to the available resources by utilizing a system of composite performance modeling of the available resources. We demonstrate the performance of this new architecture on some VLSI benchmark circuits.*

1 Introduction

The testing of VLSI circuits is done by assuming that it has a fault at a particular line and designing a *test vector* such that when this vector is applied at the primary inputs of the circuit, the output differs at the primary outputs for a faulty and fault-free circuit. Thus the fault can be detected. The list of faults that are critical and have to be tested are grouped and called the *fault list*. Techniques for devising the test vectors for the given fault list are called *Automatic Test Pattern Generators(ATPG)* [3]. Now *sequential circuits* contain memory elements in them as well as feedback loops. In testing these circuits the order of the vectors in the vector set is also critical. Hence, devising a complete test vector set for these circuits though ATPGs can be intractable. To test a sequential circuit, a random test vector set is simulated at the inputs of the circuit usign a *fault simulator*. The fault simulator assumes that a fault is present in the circuit and gives different responses for a good and faulty machine at all lines in the circuit. As this difference trickles to the primary outputs the fault is said to be detected and the fault simulator proceeds to test for the next fault in the fault list. The ratio of faults detected to the total faults in the fault list is called the *fault coverage*. Since the vector set is random there are many redundant vectors that do not contribute to the fault coverage and can be dispensed with. Typically the vector sets start at 20000 vectors and need to be compacted to about 300 for a fairly large circuit. Pomeranz and Reddy discuss some efficient techniques for compacting the test vector set [9]. In this paper we focus on the *vector compaction by omission* which is presented in Section 2.1.

1.1 Computational Desktop Grids

Grid programming has emerged as a new paradigm for solving large-scale problems in science, engineering and commerce [1]. The vision of the Grid is to deliver transparent computational power to applications distributed geographically. They also provide:

- *Secure execution environment* requiring a single authentication for multiple resources in the grid.
- *Virtual Organizations* of heterogeneous systems that pledge certain resources under their own terms and conditions.

*The research presented in this paper is supported in part by NSF via grant numbers ACI 9984357 (CAREERS), EIA 0103674 (NGS) and EIA-0120934 (ITR), and by DOE ASCI/ASAP (Caltech) via grant numbers PC295251 and 1052856.

0-7803-8430-X/04/$20.00 ©2004 IEEE

- *Seamless Transparent Access* to resources giving the impression to the user that grid were a single system and not a collection of heterogeneous geographically distributed system, which it is.

Among the Grid architectures, *Desktop Grids* allow users to exploit the idle cycles on pervasive desktop PCs to increase the available computing power by orders of magnitude. The basic idea is to harness the computing power of desktops to the fullest extent when they are not in use. Successful examples of this architecture are the Entropia Virtual Desktop Grid, Seti@Home and Condor [7,8].

1.2 Economic Models

A key issue in Desktop Grids is the distribution of cost. *Who pays for these services?.* The natural answer would be the end users who use the services. This leads to the related question: *How do we price the customer for using the services?.* The resources are heterogeneous in terms of architecture, power, configuration and availability. They are owned and managed by different organizations with different access policies and cost models that vary with time, users and priorities.

The price of a service can be expressed as:

- *Resource value = func(Strength, Resource cost, Service overhead, demand, preferences)*

Economic models are very important while deploying the Desktop Grid architecture to end users. Resource owners and end users have different goals and objectives, and for a normal user that can be cumbersome to deal with. Hence the accepted practice is to use *Grid Schedulers* or *Brokers* within the middleware that negotiate with resources on behalf of the user and to come up with a solution that is beneficial to both the user and the resource owner [4]. These brokers perform negotiations as well as *task scheduling*,i.e. the mapping of tasks to appropriate resources.

The goal of the broker is to make the actual resource transparent to the user, giving the feeling that the user is talking only to the broker. To enable this, the Grid needs to maintain these extra services including:

- *Information about the resources* such as speed, reliability etc.
- *Effective Resource Exploitation* based on the resource capacities and reliabilities.
- *Economic models* linking the usage of various resources in terms of actual cost to the end user.

The current Grid architecture consists of a *Grid fabric*(the hardware, networking and protocols) as the base and a higher level of Grid services such as the Globus Toolkit. Above this are the application toolkits that utilize the Grid services.

Now the deadline and budget constrained economic models are very rigid and deprive the user of the essential choices available to the user. This is the motivation for our refined model described in Section 4.

Problem Statement The goal of this paper is twofold. First is to solve the problem of vector compaction using a desktop Grid architecture based on the Jini framework. The second is to propose a refined economic model that is flexible and suitable for such scenarios.

In Section 2 we describe the background on the compaction technique and also on the reasons for choosing the Jini framework. In Section 3 we describe our architecture for the Desktop Grid. In Section 4 we describe the new economic model and the implementation is described in Section 5. Results are presented in section 6 and we conclude in Section 7 with some pointers to future work.

2 Background

In this section we give a brief background on the vector compaction procedures and particularly the omission technique which was used in this project. Then we discuss the *javaspaces* framework which becomes the backbone of this project. We also discuss the current economic models and their shortcomings.

2.1 Vector Compaction by Omission

Consider a circuit whose test vector set contains n vectors v_1, v_2, $v_3 \ldots$ v_n which give a fault coverage of f_1. Now the vector compaction by omission can be described as the following algorithm [9].

1. Fault simulate the entire test vector set of n vectors and analyze the fault coverage f_1
2. Set $i = 1$
3. Remove the vector v_i from the test vector set, simulate again and analyze fault coverage f_2
4. If $f_2 \leq f_1$ then the vector v_i is essential and cannot be deleted
   else delete vector v_i from the set

5. If $i = n$ then STOP

6. else increment i and go to step 3

Hence we see that the above algorithm takes n^2 time where n is the test vector length. The main advantage of this technique is its accuracy but the disadvantage is the exponential time it consumes. Hence for large circuits with long test vector lengths this technique becomes intractable and one has to resort to less efficient techniques.

This is the motivation behind this project and we would like to device a way that can provide the computing resources necessary by making the system distributed. The inherent parallelism is obvious and hence this is an example of an *embarrassingly parallel* system that needs extensive computing resources.

2.2 Jini and Javaspaces: An Overview

Jini technology, developed by Sun microsystems, is a runtime infrastructure that assists in building and deploying truly distributed systems that are organized as a federation of services. It provides a set of APIs, mechanisms and network protocols that enable addition discovery, access and removal of services. Jini presents a service based model of distributed computing; a Jini service joins a federation to share its services with clients while a Jini client joins a federation to gain access to these services. Discovery of clients is facilitated by a *look-up* service. The look-up service primarily maintains a mapping between each Jini service and its attributes. Whenever a Jini enabled device advertises its service, the look-up server adds its information to the map. A Jini client can request the look-up service for a list of Jini servers that match the design attributes. A typical interaction in a Jini based distributed system is as follows. Jini service providers first locate the look-up service. This is done by using the Jini *discovery protocol.* Once it has located the look-up service, the service provider uses the *join* protocol to become a part of the federation and register itself with the look-up service. When a Jini client connects to the system, it requests a service by sending the desired list of attributes to the look-up server. The look-up server does an attribute check and returns a list of matching services.It is then up to the Jini client to select a specific Jini client and directly request service.

Javaspaces is Java's implementation of tuple spaces and is provided as a Jini service. A Javaspace is a shared, network accessible repository of Java objects, and provides a programming model that views applications as a collection of processes co-operating via the flow of objects into and out of one or more spaces. The Javaspaces API leverages the Java type semantics nd provides operations to store, retrieve and look-up java objects in the space.

Motivation for choosing Javaspaces: Javaspaces provides methods for decoupling the semantics of distributed computing from the semantics of the problem domain. This separation of concerns allows the two elements to be managed and developed independently. For example, the designer does not have to worry about issues such as multi-threaded server implementation, low level synchronization or network communication protocols. All access operations to objects in the space such as read/write/take can be executed within a transaction.

This is the major advantage of Javaspaces and hence it naturally supports the *master/worker* parallel/distributed computing paradigm. In this paradigm, the application is distributed across available worker nodes using the *bag of tasks* model similar to the one used by Batheja *et al.* [2] and has the following advantages:

- The model is naturally load balanced. Load distribution in this model is worker driven and every worker takes almost equal amount of work.
- The model is naturally scalable.

This is the motivation for choosing javaspaces in our implementation.

2.3 Current Economic Models

The current models available include :

1. Commodity Market model based on supply-demand price fluctuations
2. Posted price model based on pre-decided prices
3. Bargaining model based on negotiating between competing resources
4. Tendering model based on tender publication and evaluation from different providers
5. Auction Model based on bids.

In all these models the there is one commonality with the interaction of the user with the broker [5]. The broker asks for a

1. *budget* which is the upper price limit that the broker can negotiate for the job with
2. *deadline* which is the maximum time limit that the job can take.

These quantities are to be estimated by the user prior to the execution [5, 6]. Based on these quantities the broker then negotiates with the resources and schedules the jobs such that the cost and the time deadlines are met. In more recent models the user is also allowed to specify whether he wants to optimize the computation for time or cost [5]. There are certain drawbacks for such models in a desktop Grid scenario which we will describe in Section 4.

3 Overall Architecture

The overall architecture of a Jini based Dekstop Grid is shown in Figure 1.

It builds on the basic framework proposed by Batheja and Parashar [2] as shown in Figure 2. The framework contains a worker module and a master module. The master module is an application level process on the master processor. It hosts the Javaspaces service and registers it with the Jini substrate. The master module defines the problem domain for a given application. It decomposes the application into independent tasks that are Javaspaces enabled, and places them in the space.

The worker module runs as an application level process on the worker nodes. It is a thin module and is configured at runtime using the *Remote Node Configuration Engine* i.e. worker classes, providing the solution content for the application, are loaded at runtime. This minimizes the overheads of deploying application code. Interaction between the master and the worker processes is via virtual, shared Javaspaces. A detailed description of this framework is presented in [2].

Our Desktop Grid essentially adds another layer to the Grid between the Grid service layer and the application layer. This is the choice plot layer which is responsible for the intelligent economic model and the interactions of the two layers which will be described later. The Brokers shown are the interfaces to every sub-organization or group within the virtual organization. This broker has the information of the resource capabilities of each of the resources within its own sub-organization. The brokers have a communication protocol by which they can communicate with the brokers within the sub-organization as well as the other brokers. This enables the Grid to be sub-divided into smaller collections, but also ensure the scalability by providing the interaction of brokers. In case the organization exceeds a certain number of brokers, then an extra layer can be added in the hierarchy to create a super-broker who communicates only with brokers in its sub-category which in turn have their own sub-organizations. In this way the architecture is split into a tree network where the organizations are small but the depth of the architecture can be increased with the number of resources in the organization.

4 Refined Economic model

Motivation: Let us assume that the economic model being used is a deadline-budget constrained model described above. Consider a case where the pricing is such that the user gives a deadline of 10 hours and a budget of *$400* to the broker. Assume that the choices available to the broker are as shown in Table 1.

Table 1: Choices available to the Broker

Choice	Cost	Time Taken(Hrs)
1	405	6
2	395	9
3	205	13

If the user chose to *optimize time* the broker finds the choice 2 but cannot choose the choice 1 as the budget is exceeded. If the user chooses to *optimize cost* then the broker selects choice 2 again as the other 2 choices fall out of either the deadline or the budget.

But now let us analyze the choices more carefully. The choice 1 and choice 2 are just 10$ apart but the difference in time is about 50%. Given a choice the user would definitely prefer choice 1 but since he does not see the choices himself he is punished for another 3 hours for his budget setting.

This is precisely the area that this proposed model targets. *We try to give the user more choices such that any mis-calculation in the initial budget or deadline setting can be negated.* The user chooses exactly what he wants and knows exactly for what he is paying for. The number of choices can be limited to be within a range in the time or cost for execution.

The operation of the proposed economic model is illustrated in Figure 2. The master is the user that

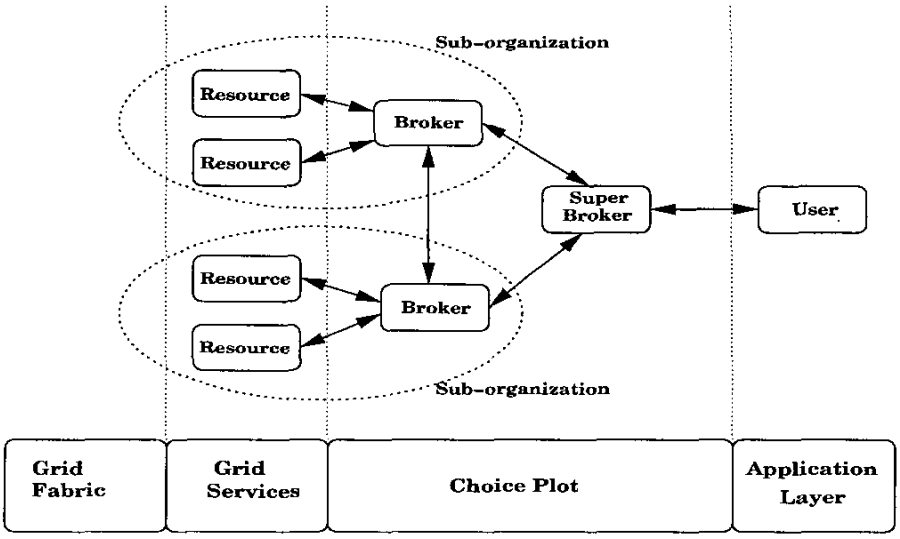

Figure 1: The proposed architecture and its position in the Grid

needs work to be done. In this model we try to complement existing economic models but try to improve them by adding more choice to the end user. The rigid deadline and cost constraints can be uneconomical and our model can be used to generate a window of choices within a specified range of time and cost.

Key interactions include:

- **Broker-Master Interaction:** The broker is listening on a port for orders. Master sends the broker information about the space and the units of work N. The Master also sends the *per unit computation time* p. The Broker then invokes its workers for quotes.
- **Broker-worker Interaction:** The broker asks the Worker for its current usage. It also asks the worker for its cost per unit time c_j, defined as the cost of running an application on machine j per unit time. c_j is based on the current usage of the worker j. If the worker is overloaded then the cost is higher for it to accept more jobs and slow down the existing ones.
- **Broker-Broker Interaction:** The broker then asks its *peer brokers* for their price quotes. The *peer brokers* in turn ask their workers for their usage and quote and final tally is delivered to the original request broker k as a *single quote* c_k.
- **Computation of the Graph:** The quotes of each broker and worker j are stored as c_j. Now we define a quantity called *Machine Factor* μ as the ratio of the running speed of different machines normalized to any one machine. The broker then decides the price choices using the following algorithm. Here the peer brokers are also considered as workers for ease of explanation.

  1. Divide work such that worker j has N_j units(equal intially).
  2. For each worker j
     time(j) = $\mu_j \times$ p $\times N_j$
     cost(j) = time(j) $\times$ (τ_j).
     Total cost = Total cost + cost(j).
  3. end For
  4. Find slowtime = Max(time(j)).
  5. (slowtime,Total cost) is one choice for the user.
  6. Transfer work units from slow to fast worker and recalculate.

  This algorithm generates all the choices within a specified range and sends them to the Master. The range can be decided by the Master or set to +/- 20% of the budget by default.
- **Choice Plot:** The choices are received from the broker by the Master and they are plotted using a barchart. The user then makes his/her choice

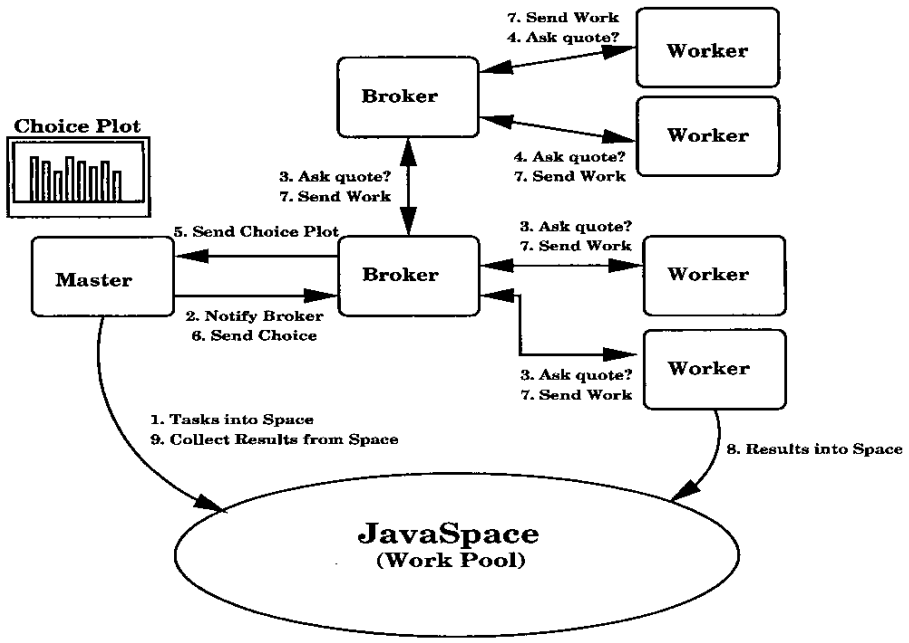

Figure 2: The architecture and Working of the proposed model

of the time/cost he wants to compute and sends it to the Broker.

- **Allotment of Work** The broker has already stored the data regarding the choices sent to the Master and their corresponding split into workers. Now as the master returns its choice the broker then allots the corresponding amount of work to each of the workers and peer brokers.
- **Work and Results** The workers then do the allotted amount of work and place the results in the JavaSpace. The Master collects the results.

The major advantage of this approach is the amount of choice that the user has, to pay for his services. It presents the user with more trade-offs ratherthan the rigid predetrmined single choice as before. This gives the broker more credibility and also makes the operation more transparent.

5 Implementation outline

The implementation of the Vector compaction using the javaspaces framework shown in Figure 2, consists of the following basic data classes aré:

Vector(): This is an object that contains a vector as a string and a vector serial number and also the total number of vectors in the set.

Task(): This is an object that contains a serial number of a vector that has to be left out in the fault simulation by the worker.

Result(): This object contains a task number on which the worker worked upon and a resultant fault coverage.

The main modules are described below.

Master module

The functions of the master module are as follows.

1. Read the vectors from vector file
2. Place each vector as *Vector* object into space
3. Place each Task as *Task* object into space
4. Notify Broker with units and per-unit-time
5. Receive Choice Plot and plot it
6. Notify broker of choice
7. Collect the objects from the *Result* class
8. Analyze the fault coverages
9. Create the compacted test vector set
10. Clean the space by removing unwanted vectors

Worker module

The functions of the worker module are as follows:

1. Listen to Broker orders
2. Send the usage and price quote to Broker
3. Get number of units from the Broker

4. Get *Task* class object from the space.
5. Read all the vectors from the space
6. Run the fault simulator on the vector set omitting vector in Task
7. Write fault coverage as *result* class object into space
8. Iterate above steps for number of work units

6 Experimental Results

6.1 Machine Factor

The machine factor is defined above as the ratio of times taken by machines to perform the same task. Since the time taken is dependent on many factors we try to average it over a certain range such that the machines are fairly comparable on basis of time.

The results of the machine factor calculations are shown in Table 2.

Table 2: Machine factors of some machines in our testbed

Machine	Machine Factor
faraday	1.0
noyce	1.12
deforest	0.63
ampere	0.70

6.2 Per unit computation time

The per unit computation time is the time given by the user to the broker as an estimate. This is specific to the type of task that is being given to the broker. In this case, we have observed that the execution time is proportional to the number of gates in the circuit as shown in the Table 3.

Table 3: Choices available to the Broker

Circuit	No. of Gates	Per unit time	Per unit time per gate
s349	196	26.36	0.129
s1196	575	77.34	0.133
s1494	680	91.46	0.134

Hence the master just computes the number of gates in the circuit after parsing it. Then

Per unit computation time = 0.134 × gates

6.3 Choice Plots

The choice plots were generated for different circuits and the results for s27 is shown in Figure 3.

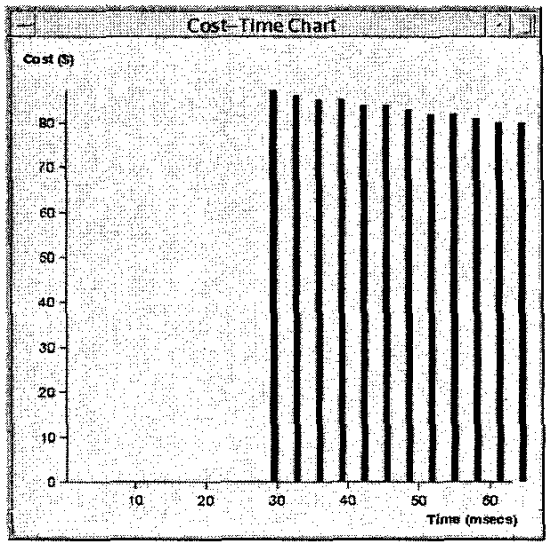

Figure 3: Typical Barchart for the circuit s27

6.4 Scalability Analysis

The model was tested on several parallel workers and the execution times are shown in Figures 5, 6, 7.

Note that the executing time for s1494 on a single machine is approximately 32 hours and by using 10 workers we have reduced the machine time to around 3 hours without loss in fault coverage.

7 Conclusion

In this paper we have presented a hierarchical, scalable architecture with an improved economic model. The model takes into account the various heterogeneous capabilities of different resources and presents the user with a choice plot containing the various cost-time trade-offs available to him. The user chooses the alternative that is best suitable for him and the brokers perform the computation within the time and cost budget. The advantage of this method lies in the fact that the ignorance of the user in setting budgets for his computation will not work to his detriment. We have implemented a testbed architecture using Jini and Javaspaces and used it for a known problem in integration circuit testing.

The limitations of this model is that we need to quantify the per-unit-computation time accurately for the broker to come up with a satisfactory choice plot. This is very applicable to units of work with almost same units of work required for each. Future work can be to implement a larger prototype and see how the model scales. The interaction time between the Master and the Broker might become significant for smaller applications.

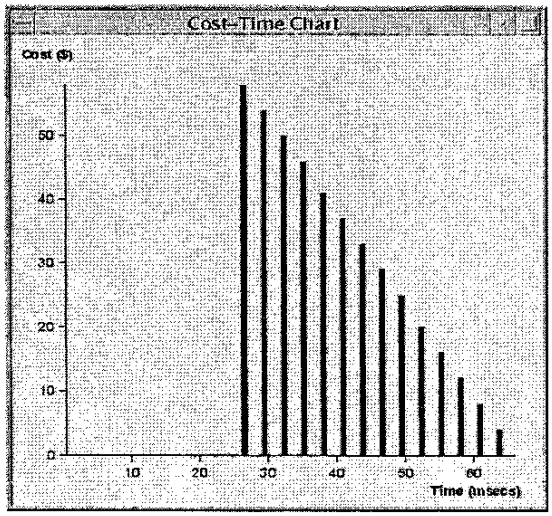

Figure 4: Typical Barchart for the circuit s349

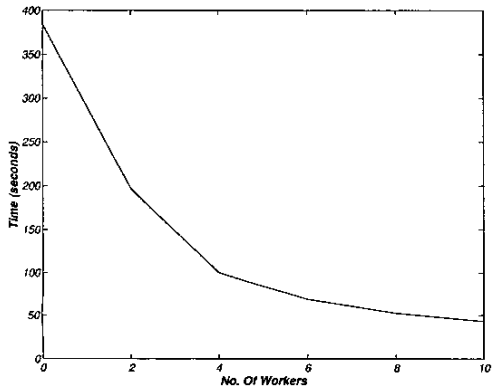

Figure 5: Execution time on multiple processes for s349

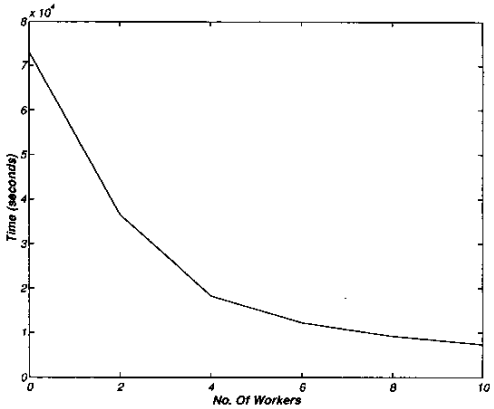

Figure 6: Execution time on multiple processes for s1196

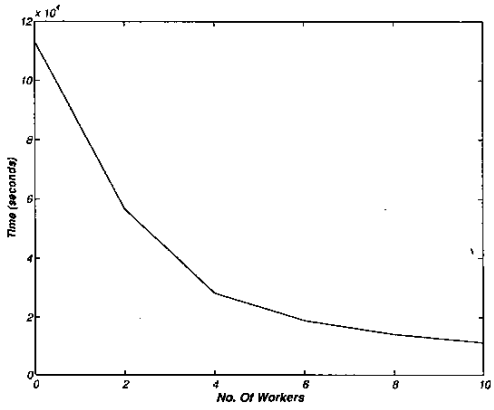

Figure 7: Execution times on multiple processes for s1494

References

[1] *The Grid: Blueprint for a Future Computing Infrastructure.* "Morgan Kauffmann Publishers, USA", 1999.

[2] J. Batheja and M. Parashar, "A framework for adaptive cluster computing using JavaSpaces," *Lecture Notes in Computer Science*, vol. 2110, pp. 647–, 2001.

[3] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed VLSI signals.* Kluwer Academic Publishers, 2000.

[4] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/g: An architecture for resource management and scheduling in a global computational grid," in *Int'l Conference and Exhibition on High Performance Computing in Asia-Pacific Region (HPC ASIA 2000)*, June 2000, pp. 283–289.

[5] R. Buyya, J. Giddy, H. Stickinger, and D. Abramson, "Economic models for management of resources in peer-topeer adn grid computing," in *SPIE Int'l Conference on Commercial Aplications of High Performance Computing*, Aug. 2001, pp. 13–25.

[6] R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Journal of Concurrency and Computation: Practice and Experience(CCPE)*, vol. 2110, May 2002.

[7] A. Chien, B. Calder, S. Elbert, and K. Bhatia, "Entropia: Architecture and performance of an enterprise desktop grid system," *Journal of Parallel and Distributed Computing*, vol. 63, pp. 597–610, 2003.

[8] D. Epema, M. Livny, R. V. Dantzig, X. Evers, and J. Pruyne, "A world-wide flock of condors: Load sharing among workstation clusters," *Future Generation of Computer Systems*, vol. 12, pp. 53–65, 1996.

[9] I. Pomeranz and S. M. Reddy, "On Static Compaction of Test sequences for Synchronous Sequential Circuits," in *Proc. of the Design Automation Conference*, June 1996, pp. 215–220.