Collaborative Distributed Computing in the Field of Digital Electronics Testing

Eero Ivask, Sergei Devadze, Raimund Ubar,

Tallinn University of Technology, Department of Computer Engineering, Raja 15, 12618 Tallinn, Estonia
{ieero, serega, raiub}@pld.ttu.ee

Abstract. Computation tasks used in digital design flow for test quality evaluation can require a lot of processor and memory resources. To speed up execution and to overcome memory restrictions, a collaborative computing approach was proposed in this paper. Web-based system architecture allows seamlessly aggregate many remote computers for one application. Efficient collaboration requires credit based priority concept, issues of task partitioning, task allocation, load balancing and model security must be handled. Experimental results show feasibility of proposed solution and gain in performance.

Keywords: collaboration, distributed computing, task partitioning, task scheduling, load balancing, priorities, fault simulation, model security

1 Introduction

Today, sizes of digital electronic circuits are still increasing dramatically according to Moore's law, which states that transistor density on integrated circuits doubles about every two years. This trend is predicted to continue at least for another decade, posing serious problems for test engineers.

To ensure fault-free products, devices must be carefully tested with appropriate test inputs during manufacturing. Widely used process in test engineering is fault simulation: it is used to evaluate the fault coverage of the test sets, it is often required for automatic test pattern generation, fault diagnosis, test compaction, built-in-self-test optimization, etc. For some computation intensive tasks, like test generation, fault simulation step is carried out many times, making simulation speed one of the key issues for overall task performance.

One solution to meet the time-to-market for larger circuits could be the improvement of the fault analysis algorithms. However, plentiful of different methods proposed during last decades leaves little room for improvement. Another way to solve the problem is to divide computation tasks and combine available computer resources. In this paper, we focus on collaborative computing. We assume that participants have computers that are not fully loaded and this load is rather uneven in time. Working is more efficient when every participant has the possibility to run his

tasks using collective resources. Effective collaboration requires credit based priorities, good task scheduling and load balancing.

Concept of infrastructure of the current solution was initially inspired from MOSCITO [1], [2] used for example in European VILAB cooperation project. Original system was implemented as client-server Java application. It allowed invoking single work tools remotely and organizing them into predefined automated workflows. Users in different locations cooperated via sharing their software tools in joint workflows. However, task partitioning and parallel execution was not supported. Major limitation for extensive Internet based use was TCP/IP socket based communication that required dedicated communication ports in firewalls on server side and also on the client side for each application and for end users.

More flexible web-based solution for remote tool usage following some key ideas of MOSCITO was proposed in [3]. Socket communication was replaced with HTTP, Java Servlets were used on server side and Applets along with Java applications were used on client side, also database was introduced. In current paper, this concept is revised and improved to support distributed computing and collaboration via effective resource sharing.

There exist also some general-purpose frameworks for distributed computing. BOINC [4] for example, built with PHP scripts and AliCE [5] that uses Java based Jini technology. Both have drawback of relying on remote procedure calls, which is restriction in firewall-protected networks.

The paper is organized as follows: web-based infrastructure is described in section 2. In section 3 priorities concept is explained. Task partitioning and task allocation are handled in sections 4 and 5. Section 6 discusses model security problems. Section 7 presents workflow with distributed computing. Experimental results are presented in section 8 and conclusions are given in section 9.

2 Distributed Environment

We have used Web-based client-server concept in our solution. There is Master server, several workstations and arbitrary number of users (Figure 1). Each user is associated with certain workstation, he is owner or he belongs to the team. Simulator application instances run on workstations. Simulator is provided with network communication abilities, it is wrapped with additional software layer – agent is created. For each simulator there is dedicated agent. Agent and simulator reside on the same computer. User accesses only Master. Users and Agents work in “polling” mode, Master is working in “answering” mode. Master and Agents must be started by administrators first. Thereafter, user can initiate a task, which is passed to Master where it is stored until a free Agent is asking for a new task. Each user has its own server-side workspace in the database. When task is complete, Agent passes results to Master where results are assembled and stored again until user will ask for results.

2.1 Implementation

Infrastructure of the system is built with Java applet/servlet technology. Communication flow between system components can be seen in Fig. 1. Tomcat is a servlet container that is used in the official reference implementation for the Java Servlet and JavaServer Pages technologies. Tomcat and servlets running on it play important role in order to access intranet resources on workstations and MySQL database. Simulator tool is implemented in C language. Wrapper agent for simulator is written in Java, which has excellent support for network programming. System components can run on different computing platforms. Simulator instances must run on their native platform. Master servlet resides usually separately from agents, they must not reside in the same local area network. Each agent can reside on different local area network. Since accessing hard drives for Java applet is restricted for security reasons by default, then GUI applet has been signed digitally, with self-signed certificate for simplicity.

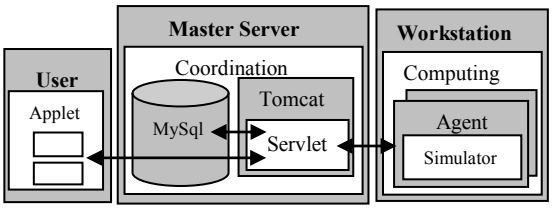

Fig. 1. System components and communication

Use of applet/servlet approach means that general communication is based on HTTP protocol. Firewall traversal is therefore no problem as only one web server port must be configured on Master server. There is no need for opening extra ports in the firewalls on the user side as it is the case with other solutions (which would be major restriction). Communication could be secured via SSL encryption, when necessary. Data passing between user and Master and between workstations is implemented following Transfer Object (TO) design practice – information is passed once as data bundle with serialized Java objects.

Data handling takes place in coordinator servlet on Master server. Problem is that web-based HTTP communication is naturally stateless. Normal HTTP session is valid only for short time, but simulation process may run much longer. We must identify users and store all their data. Users can then come back and receive their results later. Data module has three layers: presentation layer, business logic tier (data base queries, etc.) and physical database. First two layers are implemented in Java. User is accessing database via presentation layer, not directly. User tier consists of several functions to execute business layer queries. Database access is implemented using Data Access Object (DAO) design practice. Data access is using also Tomcat built-in connection pooling to speed up DB transactions.

3. Priorities

Important prerequisite for collaboration is that performance of each participant is overall enhanced; nobody should experience unfair lack of computing power, especially these participants who previously have donated most of their resources. Solution here is to introduce a priority system based on participants credit. Credit for host computer is calculated as:

$$\text{Credit} = \text{Credit} + \text{Mode} \cdot \text{HostPerf} \cdot \text{CompTime} \cdot \text{LoadFraction} . \quad (1)$$

In formula (1), Mode equals "1" when computer is donating and "-1" when consuming; HostPerf referes to performance index of the computer, it is calculated initially after executing sample calibration task; CompTime is the time host computer has devoted to the current running task; LoadFraction is number between 0 and 1, inclusively. Credit rating of the host increases while other participants are consuming its processor time, at the same time ratings of the quests will decrease the same amount. Credit numbers can go negative, it shows who is contributing and who is not. Participant with higher credit rating will have higher priority and will be served sooner.

4. Task Partitioning

There are several methods to parallelize task execution in digital test engineering: algorithm can be parallelized, circuit model can be partitioned into separate components, fault set and test pattern data can be divided. In this paper, we rely on model partitioning, but technically it is possible to improve the overall performance by adding at the same time also test pattern parallelism concept. Faults are included already in simulation model, therefore fault list itself is not partitioned. Granularity of the task is important issue to consider. The smaller the network latency, the smaller slices of the task are useful. However, Internet has relatively big network latency, therefore larger model pieces are suitable – communication between partitions should be avoided.

Our simulation method and graph based model partitioning is described in [6]. Process of fault simulation is divided into number of parallel sub-processes. For each sub-process, a partial calculation model is constructed. In the current implementation, no analysis is conducted to find the optimal selection of partition for minimizing the size of overlapped area. Instead of that, partition is selected randomly taking into account only the required task size and amount of available memory. Algorithm uses internal memory counter to keep track of the currently allocated memory and stops the construction of partial calculation model when maximally allowed amount of allocated memory is reached. During model partitioning some overlapping likely occurs, as we want to avoid interdependences. These repetitive model parts are simulated several times, resulting some speed penalty. In fault coverage point of view results will be accumulated.

5. Task Allocation

Optimal task allocation ensures that all computers would stop computing at the same time. Several aspects have to be considered. Very slow machines should get only a small fraction of total load, workstation, which has fault, should get no load at all. Distribution of the tasks should happen within short time interval. Scheduler has to be fast not to delay the execution of the tasks. Scheduling objective is to minimize overall execution time, which is actually time of the longest executing subtask. In real-life situations tasks are accumulated and then scheduled in batches [7].

There could be possibly two goals while allocating tasks to different computers: 1) maximize the speed of particular task execution for particular user 2) maximize system overall throughput from the perspective of all users. We assume that in case of collaborative networking, the latter is more important; we can use adaptive approach depending on the situation.

5.1 Scheduling Algorithm

Initially, idle agents are polling the master server. Each agent announces its tolerated load threshold, it's current load and current performance index. Agents can determine its performance index by host profiling i.e. executing a small sample task and measuring the completion time. Thereafter, Agents prepare it's profiling message for Master server. This message includes also time stamp, so Master can measure the transfer time and determine network latency index for this particular Agent (it is assumed that clocks of both hosts are synchronized). Master collects the information initially for some predetermined time, obtained coefficients are applied later – faster machines will get larger tasks respectively.

Let us assume, that tasks are already submitted to the Master. Scheduler takes the task with highest credit rating and selects an Agent with least load index in the list, given that Agent's load is smaller than allowed threshold. Thereafter, scheduler takes the task with next highest credit rating and searches for next Agent satisfying the threshold condition above. This repetitive process continues until there are no tasks in the list.

The Agent selection idea in this intuitive scheduling strategy corresponds to Best-fit algorithm described in [8] and used for memory allocation problem. When dimension of scheduling becomes larger, i.e. number of agents and tasks grows, it is possible to switch over to a First fit algorithm also described in [8] – we look for first Agent in the list who's parameters satisfy condition above. This ensures that scheduling itself is not delayed.

When higher-priority task is submitted to scheduler, and there is no idle Agent available in the system, then scheduler has to suspend or reallocate some of the current tasks if expected completion time of the priority task is greater than migration time of the current task. Candidate task for migration is the one who has the lowest priority and longest execution time. Migration means that Agent notifies Master server and sends intermediate result of simulator tool to Master, thereafter Agent terminates the simulation task. Scheduler finds new target candidate for the task under migration. It will be the Agent with the lowest credit rating and with the best

performance rating. Task must be also migrated when load on the host increases above allowable threshold, i.e. if local computing activity increases, as our goal was that participants in collaboration may not suffer. Exception is only the case, when such task belongs to the host owner. Load threshold is adjustable by each participant. Further information on load balancing can be found for example in [9], [10].

6. Circuit Model Security

In this paper, we assume that server host is trusted – circuit models must be uploaded first. Other hosts of participants can be less trusted, because to a certain extent, we can overcome model security problem automatically by partitioning the model itself - partial model is not very valuable for potential malicious agent. We may even enforce that model pieces are scheduled to the same processors during repetitive executions of the same circuit model, disabling the potential collection activity. However, identifying the right pieces of model must be handled. It is possible to compute the hash value of the model piece and save it to database. When later within limited time frame same hash value is computed for some model piece, then this task is scheduled to the same processing unit. Less limiting strategy is to ensure that not all the model pieces will be scheduled to the same processing unit. Malicious collector still will have no complete puzzle. Let us note that it is much harder to assemble the meaningful full model from pieces than just to recognize the same model piece. Computation of hash values could be performed in parallel with separate execution threads on multiprocessor computer.

7. Simulation Workflow

First, user specifies parameters and design file location (see Fig. 1). In addition, size of the simulation task can be predefined by user. Thereafter users Applet contacts with coordinating Master server and described parameters along the model are passed automatically. Task coordinator process on Master stores all requests from user(s) in the database. Simulation agents constantly poll the Master and if any subtask is scheduled by coordinator, then agents receive the appropriate parameters and design file and will start actual native simulator tool executable.

Simulator first constructs simulation model taking into account of the size limit of the subtask. While reaching the limit, it saves the breakpoint information into local file system. Simulation agent then reads the breakpoint information and passes it to the Master where it will be stored for other simulation agents. When next idle agent is polling, then it will receive the circuit file along parameters and breakpoint information. Agent starts a copy of simulator tool on the their host. Simulator first constructs simulation model again, but this time it is not starting from beginning, but restores from breakpoint up to the point when task size limit will be reached. New break point information will be again saved to local file system and later passed to Master server database by agent. Simulation agents then wait until their subtasks will be completed and report results back to Master server. Process repeats until there are

no simulation sub tasks left. Note that simulators have been started subsequently, but thereafter they run concurrently. Total starting delay is small compared to runtime. Finishing order of simulators depends. It may not necessarily be the same as starting order as simulation speed depends on the piece of the circuit model - some pieces are more complicated to simulate. When all simulators are finished, then Master assembles sub results into final result and stores in database. Results are passed to user when requested.

8. Experimental results

In experiments we measured communication overhead, memory gain by model size reduction, overall simulation speedup and scalability of the solution when processor number increases. Simulation was carried out on UltraSPARC IV+ 1500Mhz servers. Tomcat servlet engine and MySQL database were running on two cores AMD Athlon 64 6000+ 3 GHz processor with 2 Gb memory. User applet was also running on the similar Athlon machine. Circuit file loading takes less than second for the largest 3,2 Mb size circuit file B17C on the user computer. File transfer to the database and user notification takes about 6 seconds. Thereafter, simulation agent receives circuit file from Master server 3-4 seconds later. Total communication delay was approximately 12-15 seconds (8%) in average compared to local single processor solution with current sample circuits [11]. Simulation results are presented in Table 1. B17C circuit for example contains part of the Intel 386 processor, repeated three times. In simulation experiments 100K test patterns were applied to circuits.

Table 1. Distributed simulation with model partitioning.

Circuit	B17C	B21C	B22C
Number of logic gates	31008	18966	27599
Max model partitions	13	12	13
Max model build time,s	0,24	0,32	0,37
Max subtask simul. time, s	214	146	195
Model size reduction, times	4,1	2,5	2,6
Pure simulation speedup, times	3,4	2,7	3,1
Distrib. simul. speedup, times	3,2	2,5	2,9

9. Conclusions

We have presented how web-based distributed computing concept can be used collaboratively for digital circuits fault simulation. We introduced credit based priority management, addressed task scheduling and load balancing problem, handled model security issues. In experiments, web communication overhead was sufficiently small, depending on the size of the circuit and the number of test vectors simulated. In

practice, circuits and vector sets may be much larger – overhead would be consequently smaller. Further improvement could be the use of compacted models.

Model partitioning allowed reducing required memory amount up to 4 times and at the same time simulation was also speed up 3,2 times (pure simulation speedup 3,4 times) compared to single processor simulation. Further speedups can be achieved by combining model partitioning and test set partitioning in the future.

Model partitioning algorithm could be possibly improved to support finer granularity. However, for moderately sized collaborative consortium, granularity might be sufficient, especially when combined with test set partitioning, since in reality there is usually more than single circuit model in use at given time – total number of tasks to be scheduled may be larger and appropriate for number of computers available. Model partitioning is useful in sense that it offers basic circuit model security since entire model needs to be handed over to central server only.

Acknowledgement

The work has been supported by Estonian SF grants 7068, 7483, EC FP7 IST project DIAMOND, ELIKO Development Centre and European Union through the European Regional Development Fund (Research Centre CEBE).

References

1. MOSCITO, http://www.eas.iis.fhg.de/solutions/moscito
2. Schneider, A., Ivask, E., Miklos, P., Raik, J., Diener, K. H., Ubar, R., Cibáková, T., Gramatová, E.: Internet-based Collaborative Test Generation with MOSCITO. In: Design, Automation & Test in EUROPE, DATE'02 pp.221--226. Paris, France (2002)
3. Ivask, E., Raik, J., Ubar, R., Schneider, A.: WEB-Based Environment: Remote Use of Digital Electronics Test Tools. In: Virtual Enterprises and Collaborative Networks, pp. 435--442. Kluwer Academic Publishers (2004)
4. Open-Source Software for Volunteer and Grid Computing, http://boinc.berkeley.edu/
5. Teo, Y.M., Low, S.C., Tay, S.C., Gozali, J.P.: Distributed Geo-rectification of Satellite Images using Grid Computing. In: International Parallel & Distributed Processing Symposium, IEEE Computer Society, Washington (2003)
6. Devadze, S., Ubar, R., Raik, J., Jutman A.: Parallel Exact Critical Path Tracing Fault Simulation with Reduced Memory Requirements. In: Design & Technology of Integrated Systems in Nanoscal Era, DTIS '09, Cairo, Egypt (2009)
7. Hwang, K., Xu, Z.: Scalable Parallel Computing: Technology, Architecture, Programming. McGraw-Hill, San Francisco (1998)
8. Donald, E. K. Fundamental algorithms. Vol 1. Second edition. Addison-Wesly (1973)
9. Eager, D.L, Lazowska, E. D., Zahorjan: Adaptive load sharing in homogenious distributed systems, IEEE Transactions on Software Engineering, SE-12 (5), 662--675 (1986)
10. Shivaratri, N. G., Krueger, P., Singhal, M.: Load distributing for locally distributed systems. IEEE Computer 25 (12) (1992)
11. Example circuits, http://www.cad.polito.it/tools/itc99.html