ISSN 0002-306X. Изв. НАН РА и ГИУА. Сер. ТН. 2013. Т. LXVI, № 3.

UDC 681.3

COMPUTER SCIENSE AND INFORMATICS

T.H. SHAHINYAN

A METHOD FOR PARALLEL TESTING OF ELECTRONIC DESIGN AUTOMATION APPLICATIONS

Applications that are used in electronic design automation (EDA) are usually developed by hundreds of engineers and the development process lasts years. Parallel to the software development process many automated tests (AT) are developed by software quality assurance (SQA) engineers. Tests are used for regular testing of software functionality, performance and other features. The number of tests usually reaches several thousands. And running the whole test suite in an acceptable time interval is a challenging task. A method for parallelization and fastening test runs is presented.

Keywords: software testing parallelization, distributed runs, linux scripting.

Introduction. The duration of AT can last from several seconds to several hours. Short tests are preferable, because their future usage and modification is fast. But there always are tests that last long. The successive run of a whole test suite can take tens of hours or even several days. So this makes it unacceptable to use successive runs. Therefore the distributed runs are used for fastening the testing process.

The main types of distributed runs are:

1. Distribution between several computers.
2. Distribution between multiple CPUs of the same computer.
3. Conbination of 1 and 2.

Distribution between several computers is more complicated, because it is also required to consider their loading, and for that purpose commercial applications like grid engine [1] are used. In this work, the second type of distribution is presented, which doesn't depend on other applications and is developed using bash [2] scripting in the environment of Linux [3] operating system (OS). Using the Linux OS is not arbitrary. It is the main environment for developing EDA applications and it also has all the required feautures for parallelization of tasks. Modern servers can have tens and even hundreds of CPUs [4], and using distributed runs can significantly reduce the testing time.

Task Definition. Given M tests. It is required to run tests on the N CPUs of the same computer. For each test the maximum run time (RT) T_{imax} is known, where i=1,M. The maximum RT is used for killing a test, when its duration exceeds T_{imax} . This is required for not having hanging tests, which may occur for various reasons. When killing a test, all child processes should be killed recursively. After the end of each test, the next test should automatically begin. The test driver should give the following run time information:

1. The total number of tests.
2. The number of test being run, the test name, start time and maximum RT.
3. The name of the finished test, its status and duration.

Overview of the Proposed Method. During the parallel distribution of tests, it is important to sort them so that their durations appear in an ascending or descending order, so that tests with close durations should run one after the other. This is required for shortening the total RT. The ascending order is usually preferebale because if a test driver is interrupted, more tests can be run before interruption.

To organize a distributed run a BASH script is used, which accepts the following arguments:

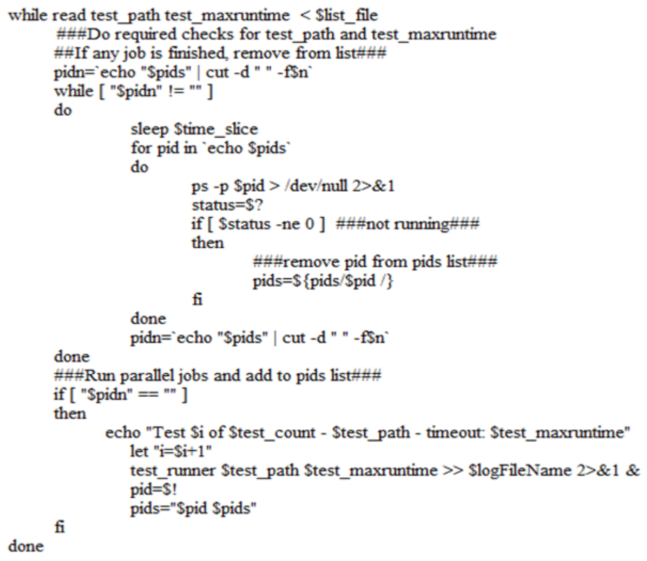
- -list <File> with the test list of the following format: test_path [maxruntime]
- [-n] <integer> number of parallel jobs to run (default is 1)
- [-maxruntime] <integer> maximum RT of ATs in seconds.

Used when a test in the list file hasn't own maxruntime (default is 3600).

The subtasks that uccur during this main task are as follows:

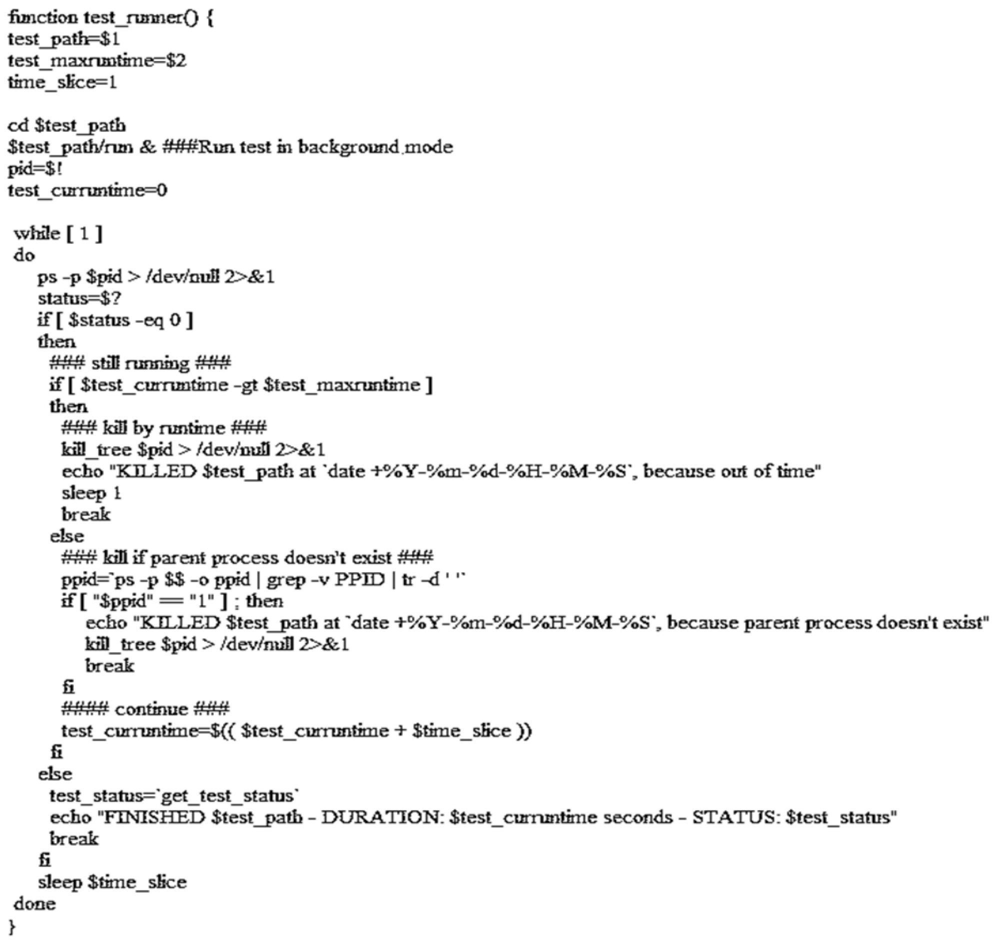
1. Creating a loop that runs all tests with N threads.
2. Having self monitoring threads.
3. Killing child processes recursively.

To organize a loop for running all tests, a BASH while a loop is used (Fig.1). To organize subtask 2, a test is run in the background mode and a job monitoring loop is created (Fig.2). For killing processes recursively a BASH function is created (Fig.3).



```
while read test_path test_maxruntime  < $list_file
     ###Do required checks for test_path and test_maxruntime
     ##If any job is finished, remove from list###
     pidn=`echo "$pids" | cut -d " " -f$n`
     while [ "$pidn" != "" ]
     do
             sleep $time_slice
             for pid in `echo $pids`
             do
                     ps -p $pid > /dev/null 2>&1
                     status=$?
                     if [ $status -ne 0 ]  ###not running###
                     then
                             ###remove pid from pids list###
                             pids=${pids/$pid /}
                     fi
             done
             pidn=`echo "$pids" | cut -d " " -f$n`
     done
     ###Run parallel jobs and add to pids list###
     if [ "$pidn" == "" ]
     then
           echo "Test $i of $test_count - $test_path - timeout: $test_maxruntime"
             let "i=$i+1"
             test_runner $test_path $test_maxruntime >> $logFileName 2>&1 &
             pid=$!
             pids="$pid $pids"
     fi
done
```

Fig. 1. A BASH loop that runs all tests with N threads on the same computer

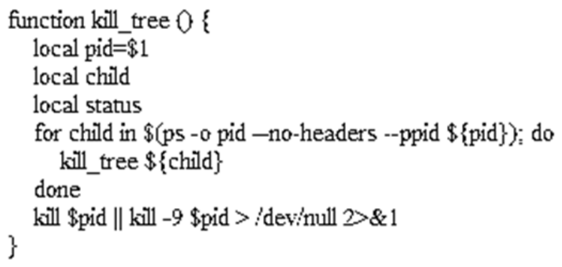


```
function test_runner() {
test_path=$1
test_maxruntime=$2
time_slice=1

cd $test_path
$test_path/run & ###Run test in background.mode
pid=$!
test_curruntime=0

 while [ 1 ]
 do
    ps -p $pid > /dev/null 2>&1
    status=$?
    if [ $status -eq 0 ]
    then
      ### still running ###
      if [ $test_curruntime -gt $test_maxruntime ]
      then
        ### kill by runtime ###
        kill_tree $pid > /dev/null 2>&1
        echo "KILLED $test_path at `date +%Y-%m-%d-%H-%M-%S`, because out of time"
        sleep 1
        break
      else
        ### kill if parent process doesn't exist ###
        ppid=`ps -p $$ -o ppid | grep -v PPID | tr -d ' '`
        if [ "$ppid" == "1" ] ; then
            echo "KILLED $test_path at `date +%Y-%m-%d-%H-%M-%S`, because parent process doesn't exist"
            kill_tree $pid > /dev/null 2>&1
            break
        fi
        #### continue ###
        test_curruntime=$(( $test_curruntime + $time_slice ))
      fi
    else
      test_status=`get_test_status`
      echo "FINISHED $test_path - DURATION: $test_curruntime seconds - STATUS: $test_status"
      break
    fi
    sleep $time_slice
 done
}
```

Fig. 2. Self monitored thread



```
function kill_tree () {
   local pid=$1
   local child
   local status
   for child in $(ps -o pid —no-headers --ppid ${pid}); do
      kill_tree ${child}
   done
   kill $pid || kill -9 $pid > /dev/null 2>&1
}
```

Fig. 3. Function for killing all child processes recursively

Determining the Optimal Number of Threads. Hyper-Threading technology [5] is a form of simultaneous multi-threading technology, where multiple applications, or multiple threads of a single application, can be run simultaneously on one processor. The Intel Xeon processor family implementation of Hyper-Threading technology enables each physical processor to appear as two logical processors to the operating system and software (Fig.4). Each logical processor maintains an independent architectural state, and can respond to interrupts independently. The two logical processors within each physical processor share the physical execution resources. Doing so allows a second, simultaneous thread to use execution resources that are not being used when only one thread is executing. The result is an increased utilization of the execution resources within each physical processor package. This improvement in CPU resource utilization yields higher processing throughput for the application. Figure 4 compares a processor with Hyper-Threading technology to a processor without Hyper-Threading technology. Hyper-Threading technology is well-suited for multi-processor systems and can further enhance their performance. On a multi-processor platform, the operating system can schedule separate threads to execute not only on each physical processor simultaneously, but on each logical processor simultaneously as well. This improves overall performance and system response because many parallel tasks can be dispatched sooner due to twice as many logical processors being available to the system. In the past, sharing memory across multiple CPUs could limit the performance scaling, due to bottlenecks in accessing shared memory. Because the caches are shared among the logical processors in a physical processor, Hyper-Threading technology can benefit applications that implement a pair of concurrent threads. The Intel Xeon family servers produce better performance when threads are twice as many as processors due to Hyper-Threading.

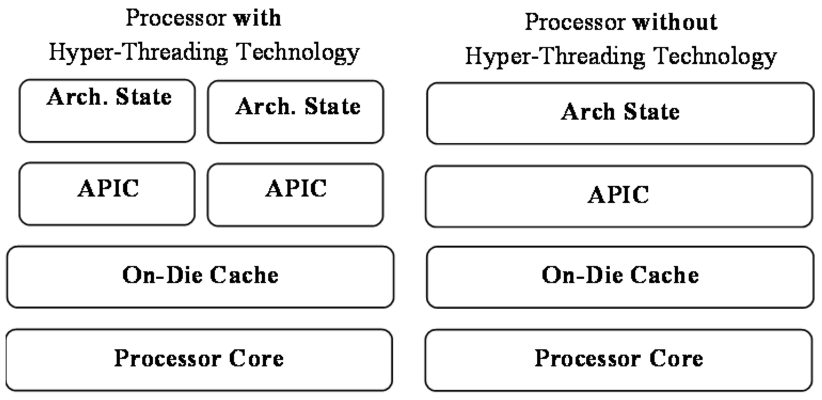


Fig. 4. Comparison of an Intel® Xeon™ Processor with Hyper-Threading Technology to an Intel Processor without Hyper-Threading Technology

Experimental results. To determine the efficiency of parallel threads on the same computer, several runs were made on an Intel Xeon server with 47GB random access memory (RAM) and16 CPUs, each 2800MGz. To use Hyper-Threading 32 tests with different durations from 107 to719 seconds were selected. The successive RT was about 152 minutes. Table 1 shows RT results for variaous threads.

Table 1

Total RT for various threads, tests have different durations

Number of Parallel Threads	Total RT (minutes)	Relative speedup
1	152	1
16	12	12.7
32	12	12.7

In both cases of 16 and 32 threads the total RT was determined by the test with maximum RT.

To determine the dependence of the total RT on the number of threads it was determined to use tests with the same duration. This was done by copying and renaming the same test. The test lasting 220 seconds was selected. Of course the test duration is not constant. It may differ depending on the server loading. The following tests were implemented to determine the speedup depending on the number of threads. The toal RT for 32 successive runs was 99 minutes (Tbl. 2). And the maximum speedup with 32 threads was 12.3. It is important to note that during the experiments, the RAM was sufficient, and dynamic memory was not used.

Table 2

Total RT for various threads, tests have same duration

Number of Parallel Threads	Total RT (minutes)	Average RT of test (seconds)	Relative Speedup
1	99	185	1
16	10	259	9.9
32	8	430	12.3

Conclusion. The testing time is an important challenge. An increase of tests count increases time challenges. A simple and efficient methodoly is proposed that can be used for distributing runs among multiple CPUs of the same server. Distribution among CPUs does not require commercial applications, thus it is cheap. The number of threads can be greater than the number of processors due to Hyper-Threading. Using a distributed run on 16 processors with 32 threads can shorten the testing time almost by 25 percents compared to the run on 16 processors with 16 threads.

REFERENCES

1. **Shankar U.** Oracle Grid Engine User Guide. Release 6.2 Update 7. E21976-02. 2012.
2. **Newham C. and Rosenblatt B.** Learning the bash Shell. Second Edition. - O'Reilly, January, 1998. – 334 p.
3. **Garrels M.** Introduction to Linux. A Hands on Guide. 1.25 Edition. - 2007. – 223 p.

4. **Merritt R.** CPU Designers Debate Multi-core Future. -EETimes Online, February 2008. http://www.eetimes.com/showArticle.jhtml?articleID=206105179
5. Building Cutting-Edge Server Applications with Hyper-Threading Technology. White Paper. Intel Corporation. 2002. – 10 p.

SEUA (POLYTECHNIC). The material is received 12.04.2013.

Տ.Հ. ՇԱՀԻՆՅԱՆ

ԷԼԵԿՏՐՈՆԱՅԻՆ ՆԱԽԱԳԾՄԱՆ ԱՎՏՈՄԱՏԱՑՄԱՆ ԿԻՐԱՌԱԿԱՆ ԾՐԱԳՐԵՐԻ ԶՈՒԳԱՀԵՌ ԹԵՍՏԱՎՈՐՄԱՆ ՄԵԹՈԴ

Ներկայացված է ավտոմատ թեստերի (ԱԹ) զուգահեռ բաշխման և թեստավորման ժամանակի կրճատման մեթոդ: Մեթոդի հիմքում ընկած է Linux օպերացիոն համակարգի (ՕՀ), ինչպես նաև ժամանակակից բազմապրոցեսորային քոմփյութերների հնարավորություններ կիրառման սկզբունքը: Առաջարկված մեթոդն իրականացված է Linux ՕՀ-ի փաթեթում առկա սկրիպտավորման լեզուներով, ինչը չի պահանջում ֆինանսական ծախսեր: Hyper-Threading տեխնոլոգիայի հիման վրա մշակված բազմապրոցեսորային քոմփյութերների կիրառումը թույլ է տալիս N պրոցեսոր ունեցող քոմփյութերի վրա բաշխել 2N անկախ ԱԹ-եր, ինչը ապահովում է թեստավորման ժամանակի կրճատում՝ համեմատած N հատ ԱԹ-երի զուգահեռ բաշխման հետ:

Առանցքային բառեր. ծրագրերի զուգահեռ թեստավորում, թեստավորման բաշխում, linux սկրիպտավորում:

Т.О. ШАГИНЯН

МЕТОД ПАРАЛЛЕЛЬНОГО ТЕСТИРОВАНИЯ ПРИКЛАДНЫХ ПРОГРАММ АВТОМАТИЗАЦИИ ЭЛЕКТРОННОГО ПРОЕКТИРОВАНИЯ

Представлен метод параллельного распределения автоматических тестов (АТ) и уменьшения времени тестирования. В основе метода лежит использование возможностей операционной системы (ОС) Linux, а также современных многопроцессорных компьютеров. Предложенный метод реализован с использованием языков скриптинга, входящих в пакет ОС Linux, что не требует финансовых затрат. Использование компьютеров, разработанных на основе технологии Hyper-Threading, позволяет параллельно распределять 2N АТ на компьютерах с N процессорами, что обеспечивает уменьшение времени тестирования по сравнению с распределением N параллельных АТ.

Ключевые слова: параллельное тестирование программ, распределение тестирования, линукс-скриптирование.