Parallel Computing for Functional Verification and Compute Farms: The Holy Matrimony

Amit Sharma
Synopsys
RMZ Infinity, Old Madras Road
Bangalore, India
0091.80.40189129,
amits@synopsys.com

Shekhar Basavanna
Synopsys,
RMZ Infinity, Old Madras Road
Bangalore, India
0091.80.40188068,
bshekhar@synopsys.com

Srinivasan Venkataramanan
CVC
Vibhu Complex, HSR Layout,
Bangalore, India
0091.80.42134156
srini@cvcblr.com

ABSTRACT

In the functional verification of complex chips, there are several phases where the requirements for the memory and runtime are far beyond the simple, single compute-server capabilities. With multi-core processors being ubiquitous nowadays, EDA tools have emerged over the last several years to provide solutions leveraging these multiple cores through parallel computing to push the limits of memory and runtime limitations of erstwhile simple computer infrastructure. At the same time, CAD teams across semiconductor organizations create Regression Infrastructures across Load Sharing Facility (LSF) setups and compute farms to use the multiple cores per machines optimally. This is done through intelligent automation which schedules various jobs across available cores to give the best possible throughput while running regressions.

Though compute and human resources are available, using them efficiently is the key. While the above approaches make use of multi-core processors in different useful ways, it becomes a challenge for CAD teams to come up with an optimal flow to leverage the Parallel Compute solutions and Compute Farm Regression Infrastructure together to increase the throughput further. One of the reasons for this is the fact that typically for the parallel compute solutions, the gains do not scale in the same order as the numbers of cores used in a specific application. For example, a simulation running on four cores might give a 2X improvement in runtime, however this is lesser than the throughput one could have obtained by scheduling four similar jobs on these four cores. This results in a lower throughput than the scenario when multiple jobs are scheduled on different cores. Another important aspect in Functional Verification is the Turn Around Time (TAT) for debug. Long running simulations take even longer when debug modes are turned on.

This paper analyzes different innovative models and comes out with a methodology for Regression Management which leverages Parallel Computation Solutions along with compute farm based solutions to give the best possible throughput. This takes into account the specific requirements for scheduling long running simulations along with shorter simulations of varying runtimes and memory overhead. The approaches are discussed and dissected for different functional verification and simulation requirements. This involves parallelism at compile time, parallelism at runtime which includes application level parallelism and design level parallelism, coverage analysis and merging parallelism. Finally, along with a presentation of the models which can be leveraged by different CAD teams, we conclude with an analysis of the modern day compute server configurations, trends and guidelines to choose for VLSI Design and Verification teams.

Categories and Subject Descriptors

New Frontiers in Verification

General Terms

Performance, Theory, Verification

Keywords

Compute farms, multi-core, simulation

1. INTRODUCTION

Given the complexity of today's chips, the number of simulations that are required to verify various functionalities in a DUT, the manifold iterations of analysis and debug, the requirements to merge and update different databases, poses a significant challenge to the verification engineers and CAD teams of various organizations. Long running simulations end up hogging resources and increase (Turn Around Time) TAT for debug and analysis. Compute farms and farm based solutions help in the utilization of available resources, however, do not provide any mechanism to cut down the time required to run the longer tests.

The EDA tools have now emerged over the last couple of years to leverage multi-core processors and parallel computing to improve simulation performance and also to push the limits of memory limitations. However, there are definite challenges in deploying parallel compute solutions professed by these EDA technologies in compute farms. The primary drawback is on the overall 'throughput'. Owing to Amdahl's law and the inherently sequential or 'dependent' nature of several applications, the gains do not scale in the same order of maginitude as the numbers of cores used. Hence, this causes a potential challenge when trying to deploy these solutions especially when compute resources are limited. Additionally, there are challenges with respect to load balancing, OS scheduling etc. when the farm based solutions try to launch multi-core jobs leading to non-optimal utilization of the farm.

In this paper, we talk about all these challenges and then delve into how we can look at each of these challenges and come up with different solutions to mitigate the same. We explore and come up with scenarios where running multi-core jobs on compute farms give the best results and can be leveraged to impact project schedules and productivity in the best possible . This paper also talks about the guidelines we followed to come up with a small application or middleware to help the compute farm software to schedule multi-core simulations better. Readers can go through this paper and come up with their own variations or algoritms to help meet their custom requirements.

2. COMPUTE FARMS IN FUNCTIONAL VERIFICATION

Verifying complex chips now require millions of simulations to be run across the project life cycle. This includes interactive, batch and simulations of different runtimes and priorities. Hence compute farms comprising of collections of machines with high speed processors or multiprocessors have become ubiquitous in design houses for launching their EDA jobs. When multiple such jobs are launched at the same time, it is important to have 'job schedulers' to help launch jobs optimally in the farms and with high throughput for atomic and batch submission of jobs.

In summary, some of the benefits that farm based solutions bring in are:

- Enables jobs processing and better processing of resources
- Software addresses load balancing so that there are minimal page faults
- Fair queuing of jobs
- QoS levels maintained
- Helps the administrator to manage priorities
- Launch jobs on appropriate machines

Thus, these software help in increasing CPU utilization rates, reducing I/O wait times, and eliminating memory paging. They enable shorter run times for jobs taking up lesser memory, as well as the concurrent execution and completion of large-memory jobs that otherwise would not complete. There are various high performance compute farms solutions available now. The most widely used amongst them are Sun Grind Engine (SGE) from SUN, and Load Sharing Facility (LSF) from Platform Computing. These does a fair job of load balancing/distribution for single core simulations besides delivering other benefits mentioned above. However, multi-core simulations throw a different set of challenges. In this paper, we explore solutions to deal with these on the LSF compute farm. The same solutions could be applied on SGE also.

3. PARALLEL COMPUTING IN FUNCTIONAL VERIFICATION

Multi-core is the trend in modern processor architectures providing performance improvement in computation and greater throughput with less power. Gains diminish on single core architectures with increase in operating frequency. Hence, multi-core processors provide an opportunity for verification tools to leverage the parallelism offered by the multi-core compute platforms to improve the overall performance. The EDA tools such as VCS have been performing this through one of the following mechanisms:

- Exploiting Design Level Parallelism (DLP): Simulate different cores of a multi-core design or different partitions of a design in parallel with other cores, leading to better simulation performance

- Exploiting Application Level Parallelism (ALP): Offloading Verification technologies or applications like waveform dumping, functional coverage collection, toggle coverage reporting, evaluation of SystemVerilog Assertions to different CPU's – reducing the overhead on the core HDL simulation

- Spawning out the compilation and linking processes of creating a simulation executable on multiple threads.

4. THE LSF SCHEDULER

The LSF Batch is a layered distributed load sharing batch system built on top of Platform LSF Base. When users run a regression, LSF follows the flow as in Figure 1 to launch the jobs [3]. LIM ((Load Information Module) and Master Load Information Module (MLIM) modules return the potential hosts on which a given job can be run, and Master Batch Daemon (MBD) module identifies the host**.** The master host puts the job in a queue and dispatches the job to an execution host after waiting for an appropriate time when an execution host with the necessary resources becomes available. When more than one host is available, the best host is chosen.

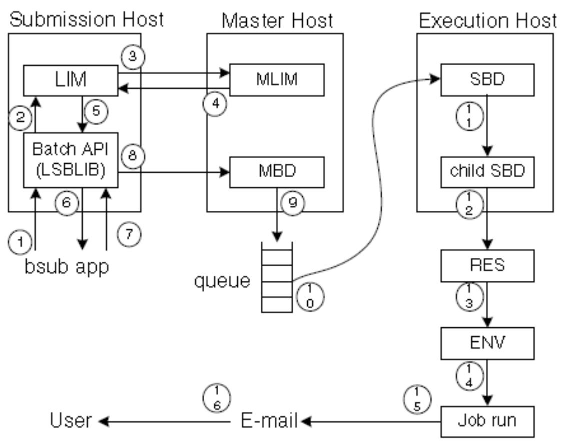

Figure 1: The LSF Batch Scheduler

5. ADOPTION CHALLENGES IN COMPUTE FARMS

With exciting multi-core technologies available, which give definite gains on long running simulations, design and verification, engineers can significantly improve the TAT on debug and other design iterations. However, as CAD teams start planning for deploying these technologies in the compute farms, they are faced with new set of questions and challenges.

5.1. Managing Throughput

Though gains offered by ALP/DLP can go up to 5-6X, typical gains are smaller in magnitude compared to the cores used. For example, if the number of cores used is two, the gains would be lesser than 2X and for three cores it might be lesser or significantly lesser than 3X, unless the 'parallelism' exhibited by the designs is of the 'embarrassingly parallel' type which it typically is not. The gains are restricted by Amdahl's law which describes an upper bound of parallel speedup over 'n' cores as given in the following equation:

$$\mathbf{T_{parallel} = \{(1\text{-}P) + P/n\}\, T_{serial} + O}$$
$$\mathbf{Scaling = T(p)/T(s) < n}$$

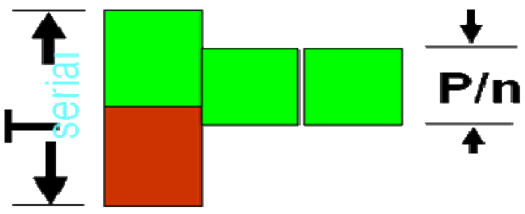

Figure 2: Amdahl's Law

Here,
T_{serial} or T(s) is the time taken for the serial simulation
$T_{parallel}$ or T(p) is the time taken for the parallel simulation
(1-P): the part of the problem which cannot be parallelized and hence is run sequentially
P: part of the problem which can be parallelized
O: time taken for synchronization etc.

Thus we can see that the '**Serial code'** limits scaling. The higher the value of (1-P), the lesser is the potential of getting high gains. Also, with even a small value of (1-P), the gains from Parallel compute will never match up to the number of cores used for the simulation.

Hence, if we just look at throughput, 'n' single core jobs submitted in an 'n' core machine will always give a higher throughput than submitting the same as a batch of multi-core simulations no matter how close the gains are to a theoretical maximum. Thus, this drives the perception in CAD teams that although they have access to superior technology, this might come at a higher cost. Though they might get good results on individual simulations, there might be non-optimal results when multiple jobs need to be turned around quickly with limited compute.

5.2. Scheduling Jobs

The compute farm software helps schedule all the single core simulations based on the dynamic state of the farm. The load balancing techniques ensure that the jobs are equally distributed across all the available machines. The requirements when it comes to multi-core simulations are different. The same algorithms that help in the job scheduling of singe core simulations might cause multi-core simulations to be run non-optimally. Here are some of the scenarios for which existing load balancing software do an inadequate job:

a) *Slot reservation/Starvation*: Multi-core jobs require more than one core to run. Hence, the Job Schedulers should be able to free up slots on a single machine as it prepares to launch multi-core jobs. Thus, there can be scenarios where there are multiple slots available on the farm, however, the sufficient slots will not be available on the individual machines to run multi-core jobs.
The reverse can be true as well and machines completely set aside for multi-core runs might be 'idle' in specific intervals. Hence, a lot of planning is required to avoid these scenarios.

b) *Conflict and collision*: There would be scenarios when single core jobs and multi-core jobs would be launched on the same machines at the same time resulting in multiple jobs running on the same core and thus degrading performance for both.
When parallel simulations are launched, they do not start running all the threads immediately. This is because, the master scheduler sitting on the main partition will take some time to start launching child threads or partitions on these different cores. As a reason, LSF will end up launching other serial/parallel simulations as well on the same cores thus overloading all.

c) *Managing Priorities*: Priorities change throughout a project cycle. For example, late in the project cycle, specific regressions have higher priority, whereas earlier in the project cycle,, specific interactive jobs have higher priority.

d) *Alarm System/Cycle Stealing*: Also, besides the scenarios mentioned in 'b' above, the threads running on some of the cores in multi-core simulations can go into 'wait' mode. This typically happens when threads are waiting for a synchronization event. The compute farm software which keeps on monitoring all these different cores sees that there is no real activity scheduled on specific cores and end up scheduling other jobs on these. This is 'cycle stealing' and is a factor that degrades performance for both the ongoing multi-core simulation and the new job that is launched.

e) *Paging*/Page Swaps: LSF by default launches jobs in the background if the running jobs do not take up the complete CPU memory.. Though this can be configured to get better results, as the default setup will overload all the jobs on a machine. Even if one manually identifies jobs that can run in parallel and serial, unless they are launched after the running jobs complete, they can overload the currently running jobs.

f) *Scheduling Across Queues:* Teams need to find out an effective mechanism to schedule specific multi-core across queues if specific queues are idle.

Because of these limitations, even a high priority regression will not be able to exploit the benefits of a multi-core simulation.

5.3. Utilization of Resources

Teams need to come up with their own analysis and criteria for deciding whether a simulation should be launched on single core or on multiple cores. The inputs for these decisions would be based on the gains available with multi-core, the number of cores used, the time taken for a simulation, the priority of the job, the compute resources and licenses available and so on.
For example, one of the questions could be,
If a'n' core job will hog 'n' licenses and 'n' cores to give a gain of 'x'%, does the improvement of x% justify the degradation in utilization?
A lot of these variables are dynamic in nature and hence the decisions on whether to launch a job as a single core job or a multi-core will change dynamically as well.

5.4 Issues with Default OS Scheduling

With multi-core processors, the OS that runs on these processors implements/provides a default scheduling, also known as OS scheduling. Many of the existing OS scheduling algorithms on multi-core systems work on the basis of "load balancing" across the cores. With a specific focus on the EDA tools used in VLSI design flow (though this fact is likely true for other applications as well), this model is not so optimal. The default OS scheduling tries to balance the jobs/threads across the available resources to ensure fair distribution of CPU time and minimize the idling of cores. This has been studied, explored and several deficiencies have been identified in other works [4,5]. The problem gets compounded as these cores are not truly "independent", they rather work in a single multi-core CPU setup, hence perform certain housekeeping processes internally from time-to-time. This can lead to some "instantaneous" load on one of the cores while the OS scheduler tries to look for a free core. Also, it is generally observed in several design verification compute processes that the peak memory/load utilization occurs during the end of the process and most importantly it tends to be non-linear.
For example, consider two compile processes getting launched on a four-core machine. Assume that comp-1 has been launched on core-1 and it is in its pre-processing stage (hence not huge load yet on core-1). Now when comp-2 arrives for scheduling, assume that the core-2 is performing some housekeeping and hence shows a large "instantaneous" load. It is likely that the default OS scheduler along with the Farm Job scheduler in this case schedules the comp-2 process on core-1 again.

LSF would launch jobs based on the slots availables and instantaneous loads. As there is no historical information of the resource requirement of the job, either because of the %CPU utilization or the memory required, it can directly impact the performance of the job. This also means that more than one 'heavy' job running on a single core.

6. DELIVERING VALUE WITH PARALLEL COMPUTE JOBS ON COMPUTE FARMS

The value of a new technology can be judged based on how it is complimentary to the existing ecosystem. If the new benefits from the parallel simulations cannot be delivered on the compute farms, its impact and adoption might not match the promise it holds. Given the challenges that have been seen, it can now be understood that the deployment of these technologies on server farms cannot be done through brute force expecting the existing farm based solutions to launch these jobs optimally. Also, at any point in time, there would always be a mix of single core jobs and multi-core jobs. And therefore, any solution has to account for this.

6.1 Parallel Compute Deployment Across a Project Life cycle

Given that the primary concern for adopting the multi-core technologies is with respect to compute resources being used up, it becomes relevant to look at the resource utilization across a project life cycle. Consider an example of compute usage, though the usage will vary across projects and organizations.

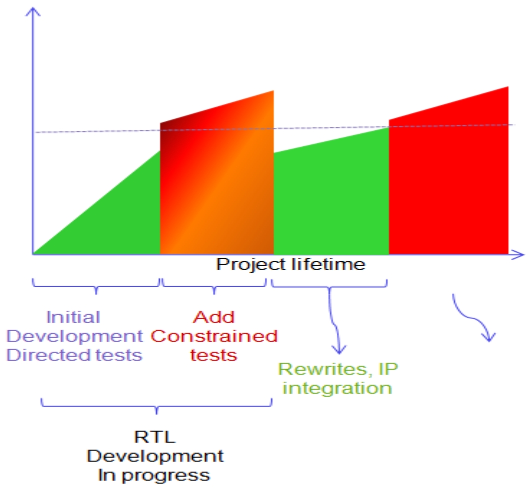

Figure 3: Example of Compute Resource Usage Across Project Life Cycle

What is apparent from the above usage representation is that the usage of these compute resources would vary significantly at different times. Hence, based on a simple integration to a license monitor or a LSF daemon for dynamic evaluation of 'free licenses' and 'available compute', CAD teams can formulate simple steps to ensure more multi-core runs are scheduled during the phases when there is more compute availability than the number of jobs being launched at specific times. This also ensures that the resources are effectively utilized at all points of time and are not left 'IDLE'.

Application of the types of parallel compute solutions across project life cycles: We can see that there also are different requirements during these cycles. With respect to parallel compute solutions in functional verification, we can map these different solutions to these different stages. During the debug phases, long running interactive jobs requiring waveform dumping and message logging has to be thrown at the servers. Hence, bumping up the priority of the jobs with 'Waveform Dumping' enabled in separate cores will improve the TAT. Similarly during the IP integration phase or the Interconnect validation stage, more 'Parallel Toggle Coverage' runs can be scheduled.

***Multi-core job scheduling for lower job rates*:**
During the scenarios where the number of jobs submitted is relatively lower, a set of simple constraints can determine how many jobs should be submitted as single core jobs or multi-core ones.

(Consider a simple scenario of having the same runtime for all the jobs)

- Submit multi-core jobs when:
  jobs * no of cores < number of active slots
- For jobs* no. of cores > slots,
  min. regression time is time taken for serial run
  Individual jobs can be run on serial core or multi-core based on the following constraints:

```
constraint all_jobs {
    no_of_serial_jobs +
max_no_of_parallel_jobs = no_of_jobs;
    no_of_serial_jobs +
max_no_of_parallel_jobs*cores_per_sim =
no_of_slots;
  }
```

Here is a simple illustration of the above explanation:

No of jobs submitted	time on serial	max no of jobs on parallel mode (2 cores)	min time (2core)	max no of jobs on parallel mode (3 cores)	min time (3 cores)	no of jobs on parallel mode (4 cores)	min time (4 cores)
50	100	50	80	50	71.42857	50	66.66667
100	100	100	80	100	71.42857	100	66.66667
150	100	150	80	125	71.42857	83	66.66667
200	100	200	80	100	71.42857	66	66.66667
250	100	150	80	75	71.42857	50	66.66667
300	100	100	80	50	71.42857	33	66.66667
350	100	50	80	25	71.42857	16	66.66667
400	100	0	100	0	100	0	100
450	200	0	200	0	200	0	200
500	200	0	200	0	200	0	200
600	200	0	200	0	200	0	200
800	200	0	200	0	200	0	200

Table 1: Parallel Mode, jobs < slots

For the experiment,

No. of machines = 100
No. of cores per machine = 4
Runtime per job = 100
Gain with 2 cores = 1.25x
Gain with 3 cores = 1.4x
Gain with 4 cores = 1.5x

Thus, for different number of multi-core jobs, as long as the requirements for cores are less than the total available, multi-core gives an improvement in individual and regression time. For multi-core requirements > 'number of active slots', but 'number of jobs' < 'number of active slots', the scheduler can schedule both serial and multi-core runs without impacting overall utilization (regression time = serial regression time, but individual run times for 'x' no of jobs will be improved).

6.2 Defining a Parallel Computation Methodology

Now, consider the scenario which is more typical when the runtime across different simulations differ. Assume there are six jobs (j1, j2, j3, j4, j5 and j6) and 4 cores are available (c1, c2, c3 and c4).

Assume runtimes are: j1 : 5, j2 : 6, j3 : 8, j4 : 11 and j5 : 25

If this test list is given to LSF, c1, c2, c3 and c4 might run j1, j2, j3 and j4 first, followed by j5.

	c1	c2	c3	c4
Jobs |	j1	j2	j3	j4
Jobs |	j5	idle for 24	idle for 22	idle for 19

core "c1" runs for a longer time (j1 + j5 => 5+25): 30 mins which is the Total Regression Time. LSF can help reorganize this in serial mode (provided we give it a list with the historical times) to run same regression in 25 units (launch j5 early).

Now let's say that j5 gives a gain of 1.6x on 2 cores. Hence, if we launch the same set of jobs directly on specific hosts as given below:

	c1	c2	c3	c4
Jobs |	j5	j5	j4(5)	j3(6)
Jobs |	j5	j5	j1(11)	j2(8)

We see that we can complete the regression in 16 time units which is an improvement over the serial run.

The above can be extrapolated to generate an efficient job matrix for a larger number of jobs across more slots for effective utilization of the farm.

Core1	Core 2	Core 3	Core 4
aj_1	aj_1	cj_1	dj_1
aj_1	aj_1	cj_2	dj_2
aj_3	aj_3	cj_3	dj_3
…	…	…	…
aj_{n1}	aj_{n1}	cj_{n2}	dj_{n3}

Here aj_{n1} (run on 2 cores), cj_{n3} , dj_{n4} are the runtimes of individual jobs $\sum aj_{n1}$ (2cores) == $\sum cj_{n2}$ == $\sum dj_{n3}$ (for ensuring max core utilization and avoiding paging effects).
Here 'n1' ,'n 2', 'n3' need not be equal
a_{j1}-a_{jn1}, c_{j1}-c_{jn3} and d_{j1}-d_{jn4} are four test lists which are created based on historical data taking into account available cores, load on the machines, run time per job and gains per job on parallel mode

Hence, using the available resources optimally through a combination of identifying specific hosts for specific jobs, parallel computer solutions can provide an improvement to utilization and TAT for individual as well as regression time.

This simplistic illustration helps deliver the following two important things:

– "Directed Scheduling" where the user is aware of the full picture and hence has more control to decide how jobs can be scheduled on specific hosts.
– Historical Data on runtimes/memory which can aid in the decision making process.

Thus, some kind of a mechanism is needed to change the default scheduling of compute farm solutions. Additionally, this has to make sure that there is enough configurability to take in different levels of user inputs. Given the fact, that there are multiple parameters which can influence how these simulations can be launched, the decisions on how to go about these should be accompanied by a set of guidelines.

6.2.1 Creating a 'Middleware' or 'LSF wrapper'

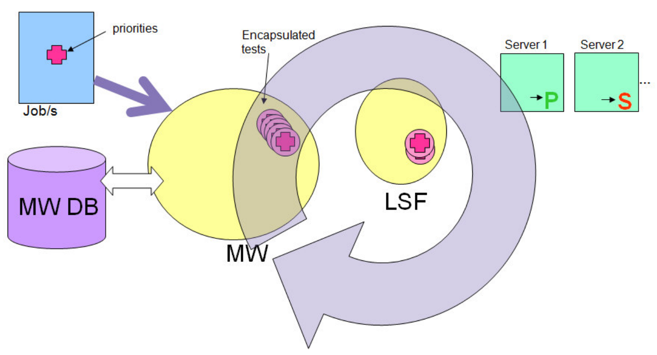

Figure 4: Middleware to Control LSF Scheduling

The 'Middleware' shown above would bias the LSF scheduling towards serial or mutlicore simulations based on the dynamic evaluation of variables and historical data lodged in the 'MiddleWare' database. The assumption is that the multi-core run times and serial mode run times are known, the gains with different cores are known and constant, the memory consumption pattern is known across single and parallel runs, and the degradation due to paging effects are captured.

For this application to give the best results, it has to manage the challenges enumerated in Section 5 above. Additionally, it has to address the following challenges:

6.2.2 Effects of Swapping/High Memory Jobs

"Swapping" occurs when chunks of RAM are paged to the disk in order to free the virtual memory. These additional read/writes that are incurred slow everything down. This typically happens when multiple large memory jobs are launched on different cores of a machine thus significantly increasing the queuing delay times encountered by other jobs. Because of multiple page swaps, there is a significant degradation in runtimes of jobs on different cores in a machine.

Hence, submitting these jobs as parallel jobs, but sequentially can give better throughput.

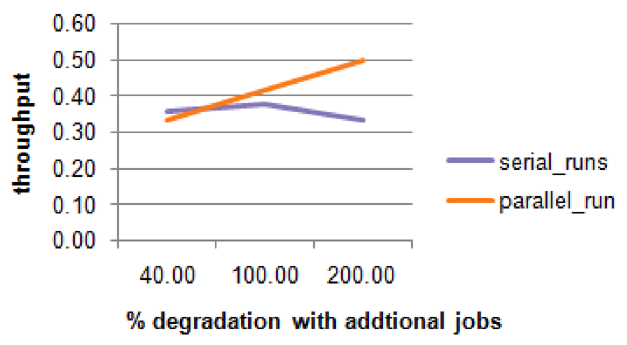

Figure 5: Effects of Paging

Figure 5 shows how multiple large memory jobs on multiple cores cause a reduction in throughput. Here the throughput of serial runs changes as more jobs are scheduled on different cores. Instead, if the same set of jobs is run sequentially on the same machine in parallel mode, the throughput with parallel mode becomes higher.

As mentioned earlier, even if one manually identify the high memory jobs that can run in parallel and serial, until they are launched after the running jobs complete, they can overload the running jobs (refer section 5.2b). Therefore, for these jobs, parallel simulations must complete first, otherwise application will end up wasting time waiting for enough cores to be available to run a parallel job. Thus there has to be check for the same.

The following table shows a typical run where T3, T4, T5 end up running on the same cores as multi-core tests T1 and T2, even though T1 and T2 occupies all the four cores of two machines with two cores each.

The overall regression time is 92.35 minutes.

Testcase	Parallel (min)	Serial(min)
T1	67.58	-
T2	68.12	
T3		73.47
T4		76.56
T5		92.06

Regression with paging effects

Now, introducing the check of ensuring that serial simulations are not run on the machines till the parallel simulations are completed gives the following results:

Testcase	Parallel (min)	Serial(min)	Comments
T1	21.28 (on vgamd269 – core 1 and 2)		Running parallel
T2	21.28 (on vgamd270 – core 1 and 2)		
T3		48 (vgamd 269 – core 1)	Running parallel
T4		48 (vgamd 269 – core 2)	
T5		48(vgamd 270 – core 1)	
T6		48 (vgamd 270 – core 2)	

Table 3: Mixed Parallel and Serial Regressions with Paging Effects Nullified

The total regression time is 48+21.28 = 69.28 mins.

Thus from the 2nd job onwards, having the application check not only for the CPU utilization but also for the job termination, provides a perceptible difference in overall regression time.

6.2.3 Honoring Priorities

In any project group, different individuals and different classes of tests would have different priorities. Hence, priorities might be set per user or per job. The application needs to keep track of the 'tokens' for submitting jobs. These tokens would take care of tracking the priorities. Through this, the application needs to come up with objective metrics to re-evaluate a 'virtual' utilization based on gains, cores usage, license usage and weightage of job into consideration. Thus jobs can be submitted in parallel mode if recomputed utilization is higher than serial utilization.

6.2.4 Slot Reservation/Pre-Emptive Strategies

The middleware would need to free up slots in a machine based on cores required by parallel mode jobs whenever there is a priority. This would involve the following:

- Re-launch low-priority jobs on different machine/or suspend them to free up slots. This can also free up memory and avoid swapping. Lower priority jobs can be terminated if the 'computed lost cycles' is low.
- Needs to be done on the fly/dynamically as otherwise machines/slots may lay unutilized. Hence dynamic re-computation and analysis of input job queue is required.
- The application should be able to disable this when required.

This is required to ensure that parallel simulations can be scheduled not necessarily only when the farm has available machines but also in scenarios of lower priority single core simulations hogging simulation slots.

6.3 Creating Middleware

The following section describes a mechanism how a viable middleware for some of our requirements can be created. Referring to Figure 1, our middleware would interact with MBD and LIM. It uses LIM functions to identify the hosts on which the jobs can be run, and submits the jobs to the queue. The primary difference is that here, MBD will not choose the host; instead, jobs will be submitted directly to the queue.

Users can invoke this application in the following two ways:

1. Single job
2. Multiple jobs (also known as batch job)

6.3.1 Job Flow

1. Get job info: Runtime of the job and its runtime memory.

2. Get the available machine list based on the memory required: Use ls_gethostinfo function to get list of the available machines; use ls_sharedresourceinfo() to get memory and CPU related information.

3. Based on available cores and memory, identify number of slots: Identify serial/parallel simulation (discussed below);

If jobs are of high priority, use lsb_switchjob() to push low-priority jobs to a different queue.

4. Generate test list: not required for single test flow

5. Start Submit: Lock the machines - ls_lockhost.; To avoid cycle stealing , use Submit jobs - lsb_submit

6.3.1.1 Identifying Serial/Parallel Simulations

To identify the number of serial and parallel simulations, the following algorithm can be used.
Define the following variables:

NoC ->	no of cores
ToS ->	no of simulations to run
ToR ->	Total runtime of all jobs
MoJ ->	Memory required for each job
SRoJ ->	Serial Runtime of a job
PRoJ2->	Parallel runtime of the job – 2 threads
PRoJ3->	Parallel runtime of the job – 3 threads
PRoJ4->	Parallel runtime of the job – 4 threads
NoS ->	number of serial simulation
NoP ->	number of parallel simulation

Based on the dynamic info (memory required and available) generated in #2, associate jobs with possible machines which run the job without introducing any swapping effect.

```
If NoC > 4*ToS -> run jobs in P-mode – 4 threads
    elseIf NoC > 3*ToS -> run jobs in P-mode – 3 threads
      elseIf NoC > 2*ToS -> run jobs in P-mode – 2 threads
        else if (NoC > ToS)
            if (single Job) -> run in serial mode.
            if (BatchJob)->IdentifyPar_Ser_Sims();
```

IndentifyPar_Ser_Sims():

Jobs that can run in parallel must meet the below condition
$(SRoJ\text{-}PRoJn) \geq (SRoJ1+\cdots+SRoJn)$

“n" is the no of cores.
The Gain observed with a parallel simulation using n threads is equivalent to running any ‘n’ jobs in serial mode. This constraint will ensure that any of the parallel simulation will not cause any degradation of utilization of the regression farm.

Job satisfying the above equation can run in parallel mode.

6.3.1.2 Generate Test List

Static test list generation can be generated based on the following scheme:

1. Sort the runtimes in descending order – using the list returned above.
2. Distribute the jobs onto the available cores, starting with parallel jobs first.
3. Total of “Required memory” of jobs should not exceed the available memory – if more than one job is launched on the same machine.
4. Continuously monitor the total runtimes on each core.
5. Distribution has to be ensured that there is no high variation.

6.3.1.3 Job Submission

Here the machine and the queue to which jobs will be launched are known and the following steps are considered:

1. Submit the jobs using lsb_submit function.
2. Lock the hosts on which jobs are running to avoid cycle stealing. This would also protect from LSF directly launching other jobs.

Thus the application leverages functionality created using LSF APIs and could be extended for other server farm solutions. They provide additional intelligence in terms of scheduling jobs when multi-core executables are launched. It follows a “learn and apply” methodology. Therefore, it expects job’s runtime, and peak memory required data to be available before hand. This can be enhanced to take in dynamic inputs and change scheduling on the fly so that depending on the available compute, jobs which might not strictly meet the requirements to be launched as parallel jobs can still be scheduled on different cores leading to better runtimes.

The application deals with the following situations in different ways:

1. Jobs requiring high memory
2. Jobs with low memory requirement

Assuming that memory is not a bottleneck for the tests, tests runtimes are sorted so that longest running job will be launced first and the shortest at the end. Now, the number of cores available is the maximum number of jobs that can run together across various cores. The job submissions routine will identify parallel and serial simulations and will launch them accordingly.

If memory is a bottleneck, it only means that any additional job on that host can degrade the overall performance of each job. In such cases, it is better to run them individually, which would imply running them in parallel mode to get the best overall regression time.

Here are some more illustrative examples on how the scripts take care of submitting jobs based on available resource. This is for 6/4/2 tests on two machines (2+1 cores available)
Serial runtime – 48 minutes
Parallel runtime – 21 minutes (2-threads)

6 tests (T1, T2, T3, T4, T5, T6) – available cores 2 + 1 – available hosts 2

Core 1 (machine 1)	Core 2(machine 1)	Core 1(machine 2)
21 (T1)	21 (T1)	48 (T2)
21 (T3)	21 (T3)	48 (T5)
21 (T4)	21 (T4)	
21 (T6)	21 (T6)	

From the results, we can see that the script ensures that cores are equally loaded, and will not overload other jobs running. Hence jobs will complete as efficiently as when launched separately.

4 tests (T1, T2, T3, T4) – available core 2+1 – available hosts 2

Core 1 (machine 1)	Core 2(machine 1)	Core 1(machine 2)
21 (T1)	21 (T1)	48 (T2)
21 (T3)	21 (T3)	
21 (T4)	21 (T4)	

Here, it is ensured that most of the jobs run in parallel, ensuring better throughput compared to running in serial mode.

2 tests (T1, T2) – available core 2+1 – available hosts 2

Core 1 (machine 1)	Core 2(machine 1)
21 (T1)	21 (T1)
21 (T2)	21 (T2)

As number of cores>jobs, the script launches both the jobs in parallel mode. Hence the overall regression time is >2X.

7. RESULTS

Though the application does not cater to all the challenges which were enumerated earlier, it gives the user a mechanism to add in the additional configuration parameters. In its present shape, it has the following advantages:

1. Provides the ability to exploit the benefits of multi-core simulation wherever possible.
2. Based on changing priorities, can dynamically run the job in parallel mode
3. If slots are not available, as the total runtimes are known and the current time is known, it can approximately predict if the job has to wait for a longer duration for a slot. It then proceeds to start reserving slots if the job can run in multi-core mode.

We have also enumerated results of some of the smaller runs on the prototype at different points in the paper. Across, a larger set of serial and multi-core jobs over multiple designs across Synopsys VCS Benchmarks, we have seen gains ranging 2-3x when simulations are launched directly on the LSF.

8. ANALYSIS OF THE MODERN DAY COMPUTE SERVER CONFIGURATIONS

If we look at the roadmap of modern multi-core servers from major providers, we can see some interesting trends from the purview of multi-core simulations. What is apparent and common in the roadmaps posted by the most powerful server manufacturers is that the number of cores available on these servers is continuously increasing. What this would mean in the context of parallel compute simulations is that on compute farms, the availability of multiple cores for a single simulation will significantly increase reducing the chances of 'starvation' for parallel simulations. Also, with more memory, the effects of swapping might be minimized to some extent. However, with the increased numbers of cores, the number of permutations will also increase with respect to gains, cores used, and memory requirements and so on, there would be always a challenge to come up with an objective mechanism of ensuring the best possible utilization of these machines and hence applications like the one described in this paper can play an important role.

Also, there are performance benchmarks done by Intel on various Xeon based systems showing potential speed-ups. As this data tends to be highly dynamic, we highly recommend checking with the respective vendors for updated information. Intel has published several whitepapers focused on "EDA application performance" with its Xeon family of servers. With growing number of cores available per server, Intel has shown that the TCO (Total Cost of Ownership) for running large regressions can be significantly reduced by replacing several existing, old servers with single or few new servers. Readers are highly recommended to explore further through www.intel.com/it.

Recently several simulation case studies, benchmarks in EDA have emerged revealing some interesting results. Multi-core along with GPUs boosts simulation performance in certain classes of problems such as numerical computation. This may not apply to the wide range of problems but in general GPUs have been proven very useful if there are enormous numbers of "small" computations that are very similar in nature such as finite element analysis, image processing etc. At the time of writing this paper, detailed results from such experiments were not publically available.

9. CONCLUSION

Besides delivering superior gains in runtimes for individual jobs, we can see that there can be multiple scenarios where multi-core can give additional productivity in compute farms. As long as a set of guidelines are catered to, and some intelligence is added to existing farm based solutions, we can get the additional increase in server utilization. Also with time-to-market pressures, design teams often tend to erase the previous data as fast as they can to pave way for new databases. However as demonstrated by various experiments in this paper, having access to quality, reliable data specific to a company design can greatly help in predicting the need for next designs, arrive at optimal scheduling etc. For instance if the previous tapeout had regressions with 80% short runs and 20% long runs, having just this data can immensely benefit in arriving at a customized scheduling for the current design.
Though quite a few challenges for parallel compute on server farms are enumerated in this paper, there are a few other areas which can be analyzed further. These includes generating models to compute change in overall utilization because of launching of a parallel job, analysis of optimal means of freeing up slots to support a multi-core job/s, and considering the effects of I/O and data management and data caching when it comes to multi-core jobs on compute farms, Managing these will help in refining a model that was presented here in a more comprehensive manner.

10. ACKNOWLEDGMENTS

Our thanks to the Synopsys VCS R&D team for helping us understand the different intricacies of parallel simulation and giving us different live cases to experiment with using Parallel VCS. We would also like to thank the Synopsys Infrastructure team to provide us with compute farm resources for all our experiments.

11. REFERENCES

[1] http://www.design-reuse.com/articles/5732/maximize-cpu-power-for-physical-verification.html (Maximize CPU power for physical verification)
[2] www.exludus.com/PDFfiles/IBM%20Paper.pdf (Improving Compute Farm Throughput in Electronic Design Automation (EDA) Solutions)
[3] LSF Programmer Guide
[4] S. Cho and L. Jin. Managing Distributed, Shared L2 Caches through OS-Level Page Allocation. In MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, pages 455–468, 2006.
[5] G. E. Suh, S. Devadas, and L. Rudolph. A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning. In HPCA '02: Proceedings of the 8th International Symposium on High-Performance Computer Architecture, page 117, 2002.
[6] www.intel.com
[7] www.amd.com