Creating the Optimal Regression Farm Infrastructure That Meets All Your Team's Simulation Requirements

David Lacey[1] Ed Powell[2]
[1]Hewlett Packard Enterprise, 3404 E Harmony Road, Fort Collins, CO 80528, David.Lacey@hpe.com.
[2]Hewlett Packard Enterprise, 3404 E Harmony Road, Fort Collins, CO 80528, Ed.Powell@hpe.com.

***Abstract*- Creating a regression farm filled with lots of CPU cores running a continuous stream of simulations is standard practice for VLSI teams. However, it is difficult to understand what the optimal configuration of this regression farm really is when considering many factors such as hardware costs, license costs, and use models. This paper will explore the various requirements that impact the optimal construction of a regression farm and provide insight on how to best balance these requirements. Finally, we will explore how to continuously track the performance of a regression farm to ensure the optimization is maintained.**

I. Introduction

HPE, similar to many other VLSI development companies, has a centralized compute infrastructure that is shared across multiple VLSI business teams within the company. While this approach provides significant value, there are ongoing budget pressures regarding investment in server hardware upgrades. As a result, it is important to clearly show the value of consistent hardware refreshes. To do this, HPE has consistently collected performance data across multiple generations of servers to showcase the value brought by newer servers.

However, while installing newer servers based on the latest and fastest processors can clearly bring increased performance for EDA workloads, our team has analyzed additional aspects of our compute infrastructure to ensure we are maximizing our investment across multiple axes of our investment. Since budgets and investments are constantly scrutinized, we want to ensure that we are optimizing all of our investments needed to develop custom ASICs. This paper will present a background of CPU architectures to provide a foundation for the value brought by the different compute configurations we use. We will then present the results of many of our performance tests to support the conclusions we have made regarding our infrastructure configurations. Finally, we will describe different axes of optimization that look across more than just raw workload performance to include engineering experience and license costs. We will show how these additional aspects can influence the architecture of the deployed compute resources.

II. Experimental Setup

Characterizing the simulation throughput of a compute farm in a controlled repeatable way can be difficult. The method used followed these steps:

1. Analyze job statistics available from the regression farm queuing system to select a representative short running simulation that produces IO, memory, and CPU utilization close to the per job network averages. This exact simulation run will be used to measure simulation runtime between the start of simulation time advancing and end of the simulation. We exclude other parts of the simulation time, like model compile and load.
2. Select a collection of other simulations with close to the average load but with longer run times to use as a "background load". Make sure the memory used by the selected jobs does not exceed available memory when a host is executing jobs on all compute slots.
3. Run N of the background load simulations along with the simulation used for runtime measurement on an otherwise idle host. Use a fixed seed and consistent wave dumping and logging settings on all runs.
4. Repeat with N from 0 to the number of available compute slots on the host.
5. Repeat for different hardware types we want to compare.
6. Plot simulation wall clock time in seconds versus the total number of simulation jobs running, for each hardware configuration being characterized.

In these experiments, each simulation instance consumed one processor core. The graph in Figure 1 plots the simulation length when utilizing different number of processor cores. This data clearly shows the impact on individual simulation length (i.e. individual simulation performance) when using more and more cores on a socket. When

running only one simulation on the entire socket, the simulation completed in less than 900 seconds. When running sixteen instances of simulations across sixteen cores on the same socket, the reference simulation took an average of over 1800 seconds, which is more than double the simulation length when using only one core.

As a follow up experiment, we ran similar experiments across a wider variety of compute platforms and processor

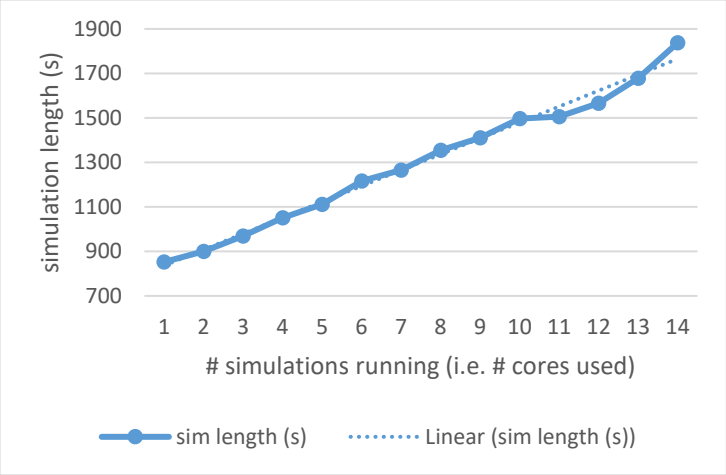


FIGURE 1 : SIMULATION RUN-TIME VERSUS NUMBER OF CORES UTILIZED

SKUs to confirm the data trends remained consistent. The results are show in Figure 2. In this graph, the data is normalized across a wide variety of platforms and thus the y-axis is presented in nanoseconds of simulation time versus the wall clock time needed to complete that simulation (ns/sec).

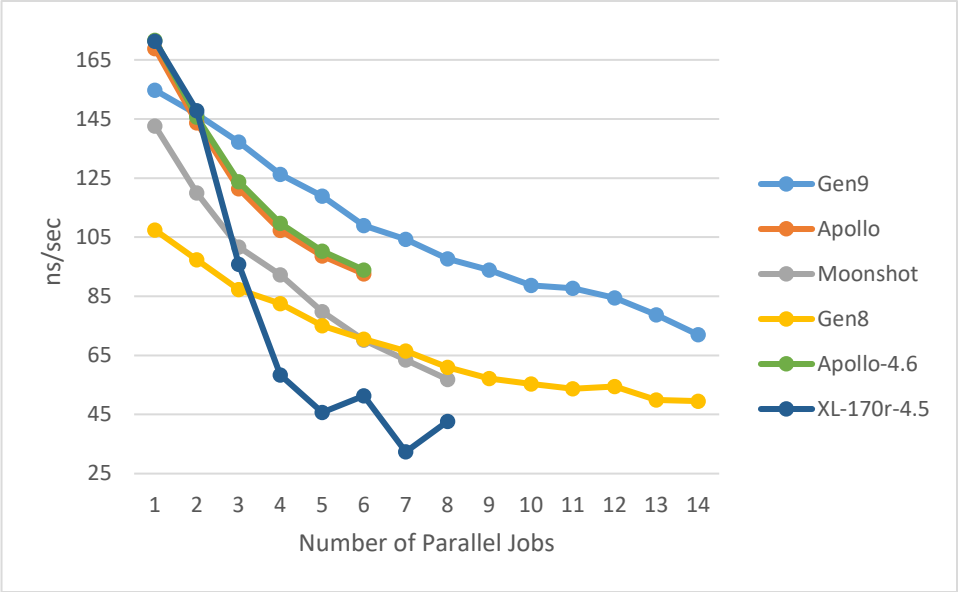


FIGURE 2: SIMULATION PERFORMANCE ACROSS DIFFERENT PLATFORMS

In all cases, the message is the same – running increasing number of simulations on a single socket negatively impacts the simulation performance. The next section will discuss reasons why the simulation performance is impacted as more cores on the host are kept busy.

III. BOTTLENECKS IMPACTING SERVER PERFORMANCE

To really understand how to optimally configure your compute environment, it is important to understand the architecture of modern processors. There are a number of processor architecture factors that have a significant impact on a simulation's performance. Understanding these factors and their impact on simulation performance is critical to

make the best decisions on how to configure the environment. Some typical performance bottlenecks on multiprocessor compute servers are bandwidth of the shared memory bus, network performance, local scratch disk performance, and insufficient memory causing virtual memory swapping to disk.

A. CPU Architecture

Through the years, processor companies have increasingly integrated more and more capabilities into the processor package. Modern processor architectures typically include high speed, proprietary interfaces for processor to processor communication, an interface to a chipset for bridges to other bus protocols such as PCIe, and integrated memory controllers for direct attach memory. In multi-socket architectures, the processors are enabled to access both the local memory connected to their own socket as well as access to remote memory connected to other processor sockets by communicating through the proprietary processor-to-processor high speed interfaces. This is shown in Figure 3.

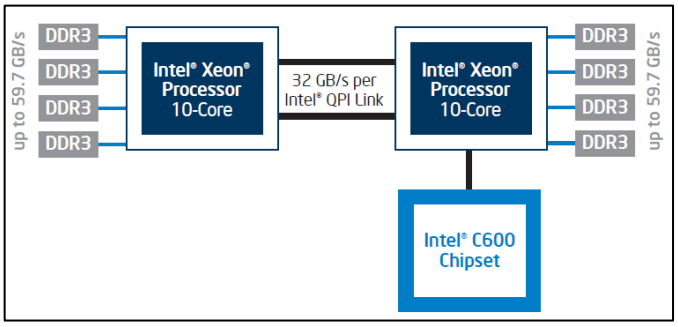


FIGURE 3: INTEL XEON PROCESSOR ARCHITECTURE [1]

B. Cores and Memory

There is a trend with CPU designs in which the number of cores increases with each new generation. This brings many additional processing cores to a standard two-socket server. Table 1 shows a sampling of the processor options available for the HPE DL380 2-socket server which range from 4 cores/socket to 22 cores/socket. Other processor options include the speed of the memory attached bus (in this case the DDR4 bus) and the amount of L3 cache/socket. Watch closely the L3 cache/socket ratio as this is a shared cache across all cores. An additional column is included to capture the amount of L3 cache per core for comparison purposes. For this generation of processor, even as the number of sockets increases, the L3 cache/socket stays static except in a few cases. Obviously, the L3 cache is not subdivided equally per core but looking at the cache per core gives a better perspective of the impact the cache will provide when there are simulations running across all cores.

TABLE 1: SAMPLING OF HPE DL380 SERVER PROCESSOR OPTIONS [6]

Model	CPU frequency	Cores	L3 Cache	L3 Cache / Core	DDR 4 Hz
E5-2637v4	3.5GHz	4	15MB	3.8MB	2400
E5-2643v3	3.4GHz	6	20MB	3.3MB	2133
E5-2643v4	3.4GHz	6	20MB	3.3MB	2400
E5-2640v3	2.6GHz	8	20MB	2.5MB	1866
E5-2667v4	3.2GHz	8	25MB	3.1MB	2400
E5-2640v4	2.4GHz	10	25MB	2.5MB	2133
E5-2660v3	2.6GHz	10	25MB	2.5MB	2133
E5-2650v4	2.2GHz	12	30MB	2.5MB	2400
E5-2690v3	2.6GHz	12	30MB	2.5MB	2133
E5-2683v3	2.0GHz	14	35MB	2.5MB	2133
E5-2690v4	2.6GHz	14	35MB	2.5MB	2400
E5-2683v4	2.1GHz	16	40MB	2.5MB	2400
E5-2698v3	2.3GHz	16	40MB	2.5MB	2133
E5-2695v4	2.1GHz	18	45MB	2.5MB	2400
E5-2697v4	2.3GHz	18	45MB	2.5MB	2400
E5-2698v4	2.2GHz	20	50MB	2.5MB	2400
E5-2699v4	2.2GHz	22	55MB	2.5MB	2400

Just adding more cores to the processor does not increase individual EDA workload performance linearly as many workloads can only utilize a single core. This is beginning to change as place and route tools have utilized multicore for a while. Additionally, we are beginning to see event based simulators supporting multiple cores but this is still in very early stages of maturity. So the analysis we focused on was for EDA workloads using only a single core.

As you have more cores running EDA workloads such as simulations, they are competing for the same external interface bandwidth, such as the DDR4 bus. A larger cache in the processor will help, but typically a simulation will not fit in the cache memory on the processor, so there will always be lots of traffic generated on the memory bus for each simulation that is running. This memory bottleneck is responsible for the decreasing simulation runtime performance as we increase the number of jobs running on that host. See experimental results shown in Figure 1.

This degraded performance per simulation as more and more simulations are loaded onto the same socket can be easily traced back to increasing cache miss rates and oversubscribing memory bus bandwidth. To confirm this expectation, we collected performance metric data using the `perf stat` Linux utility. As captured in Figure 4, we see that the cache bandwidth drops and the cache miss rate rises as more simulations use additional cores in a socket. Similarly, as shown in Figure 5, a measure of simulation performance (ns of simulation time / seconds of wall clock time) is shown to decrease as more simulations are loaded on the same socket. This data shows the same messages as was seen in Figure 1 and Figure 2. Additionally, Figure 5 shows that as simulation performance degrades with more cores in use, eventually the CPU Utilization begins to be impacted as cores are stalled waiting for data to arrive.

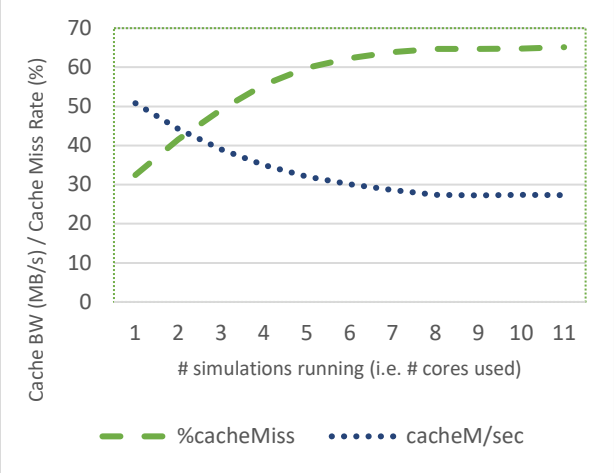


FIGURE 4 : CACHE BANDWIDTH AND MISS RATE

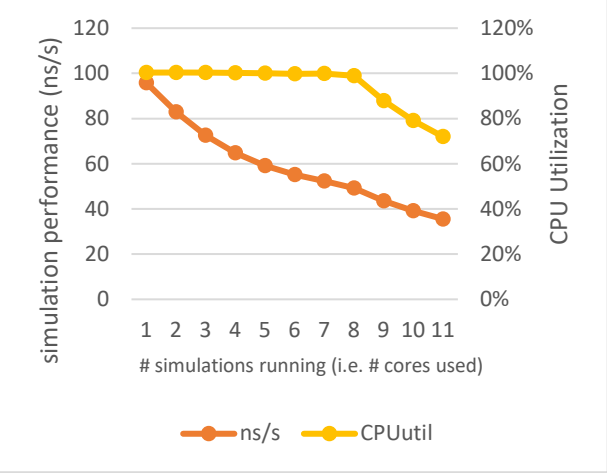


FIGURE 5 : SIMULATION PERFORMANCE AND CPU UTILIZATION

The experimental results presented above also match industry trends that clearly show the memory bandwidth per socket is flattening which when combined with the significant increase in core count per socket produces a significant decline in memory bandwidth per core. This is illustrated in Figure 7 and Figure 6. The lack of memory bandwidth creates the difficulty of keeping each simulation per core fed with the memory data those simulations require to run at full speed and ultimately results in slower simulation speed as more simulations are started on additional cores.

This message was repeated during a GenZ presentation at Super Computing ’17 where it was highlighted that the computer-memory balance is degrading. Figure 8 shows this trend of less memory bandwidth per core that today’s processors are experiencing.

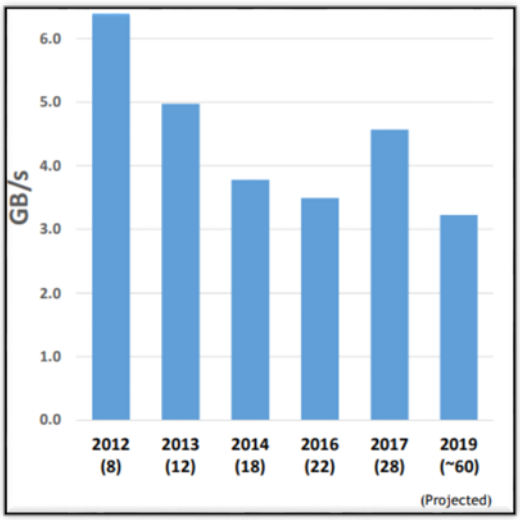


FIGURE 7: MEMORY BANDWIDTH PER CORE [8]

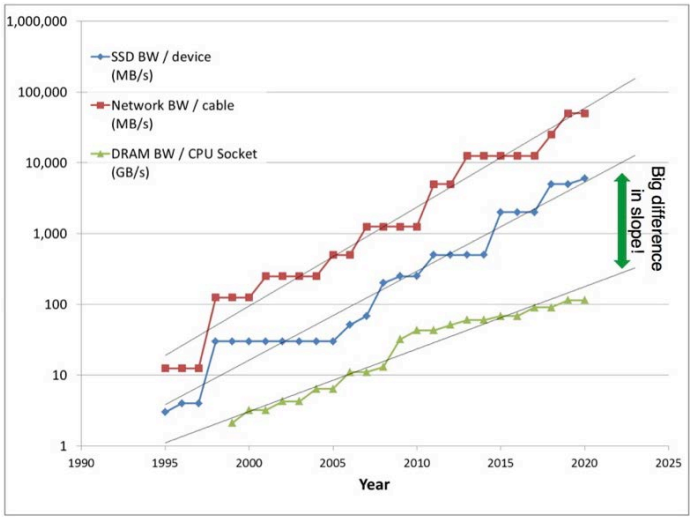


FIGURE 6: PROCESSOR MEMORY BANDWIDTH PER CORE IS FALLING [7]

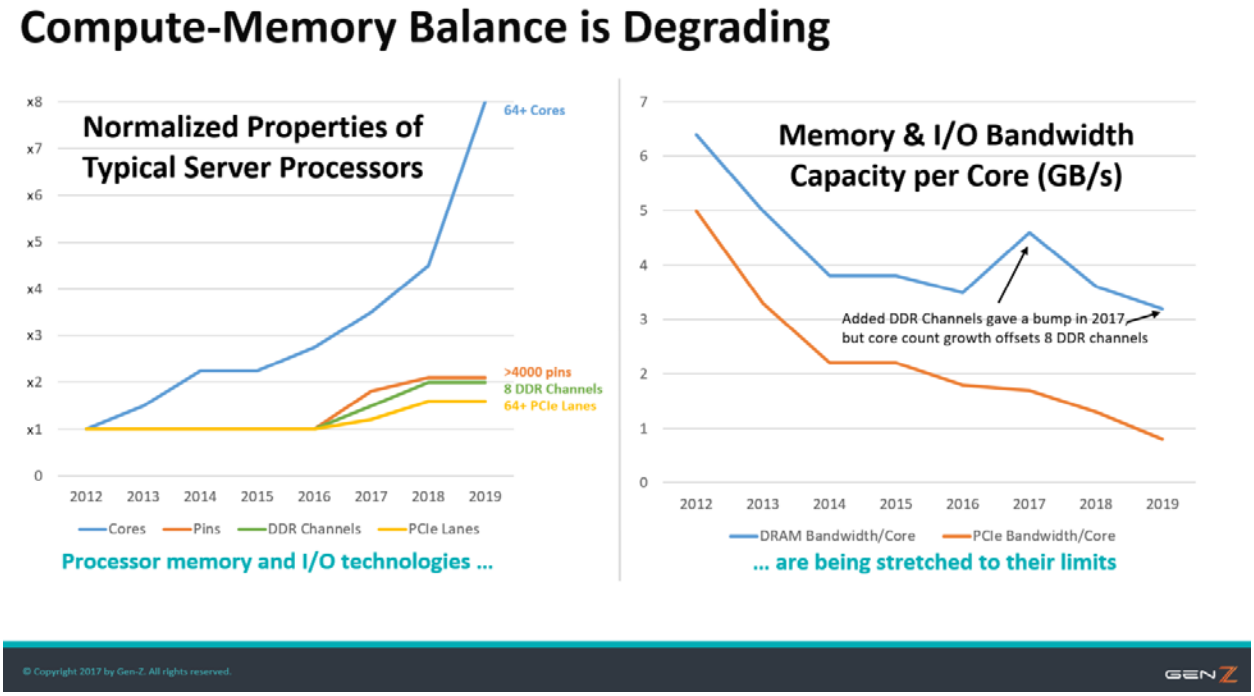


FIGURE 8: COMPUTE-MEMORY BALANCE IS DEGRADING

C. Clock Speeds and Cache

The last experiment we conducted was to compare performance of different processors with different clock speeds and cache capacities. Two of these are listed in Table 2. In this case, there is a significantly faster clock rate for one processor but it comes with less overall cache memory. The question we wanted to answer was how much faster clock rates help simulation performance.

TABLE 2 : CLOCK SPEED VERSUS CACHE SIZE EXPERIMENTS

HPE System	Processor SKU	uProc Clock Rate	# cores	L3 cache size
Apollo	Intel Xeon E5-1650 v3	4.4 GHz (overclocked)	6	15 MB
DL380 Gen9	Intel Xeon E5-2699	3.6 GHz	18	45 MB

As the graph in Figure 9 shows, the faster clock speed does clearly help simulation performance when only one core on the socket is being used. However, because of the significantly smaller cache size of the Xeon E5-1650 processor, as the number of cores running simulations is increased, the performance boost provided by the faster clock rate is nullified by the limited amount of L3 cache. Ultimately, the processor is running very fast as it waits for data to arrive from external memory. In typical regression scenarios where multiple simulations will be running on each core, it is better to choose processors with larger amounts of L3 cache over a processor with less L3 cache but a faster clock rate.

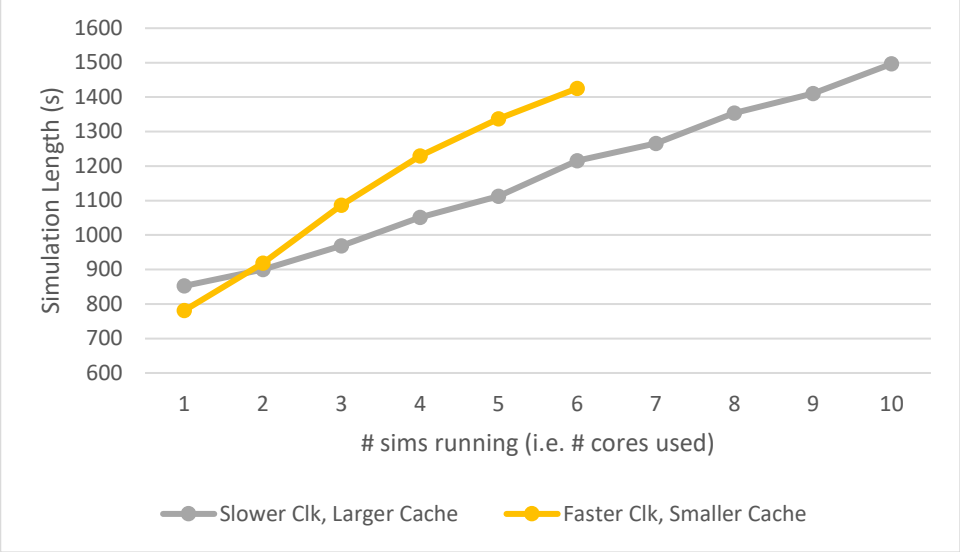


FIGURE 9 : PROCESSOR CLOCK SPEED IMPACT

Cadence Design Systems also conducted experiments related to the impact of clock speed and cache and their results showed similar outcomes as shown in Figure 10 and Figure 11.

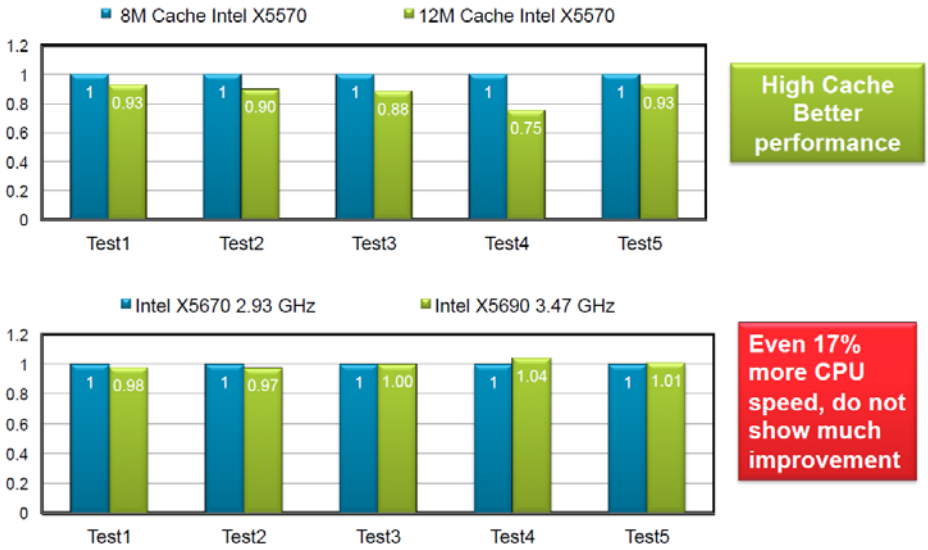


FIGURE 10: CADENCE DESIGN SYSTEMS SINGLE JOB CACHE VERSUS CLOCK RATE [9]

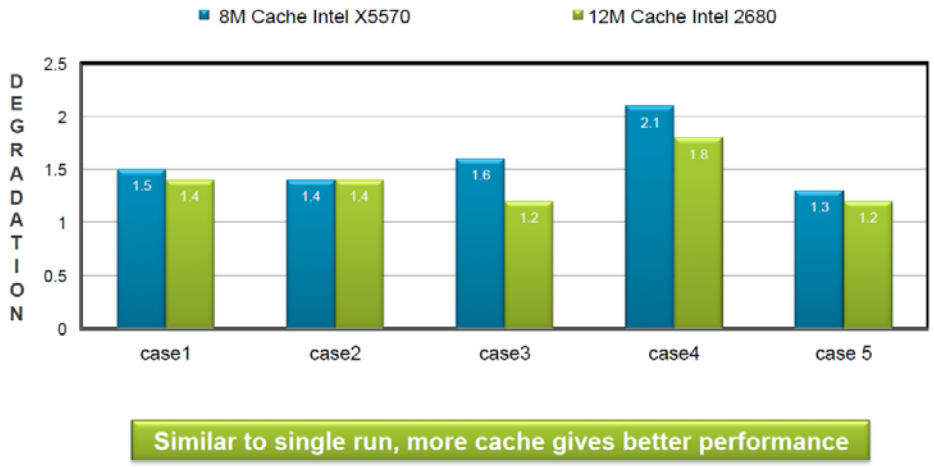


FIGURE 11: CADENCE DESIGN SYSTEMS 8 PARALLEL JOBS DEGRADATION ON 8 CPU MACHINE WITH DIFFERENT CACHE SIZE [9]

IV. AXES OF OPTIMIZATION

Based on the results from Sections II and III, HPE has investigated a number of axes of optimization that contribute to meeting a VLSI team's regression farm needs. Depending on the user's objective, the compute environment will be configured differently. To maximize simulation result total cost of simulation throughput, for example, we will show how simulation license cost and hardware cost must be considered to get the best overall cost for the required amount of simulation throughput. Another tradeoff is between the need to keep expensive simulation licenses busy running regression jobs versus the need for individual user interactive jobs to startup, execute, and complete quickly. The compute farm will be configured differently for each of these optimization axes. Finally, HPE has implemented techniques for dynamically reserving resources (such as licenses) as a function of time of day to insure user jobs can start up quickly, while letting regression jobs fill in and use the remaining licenses.

A. Optimizing Total Cycles Run

Most teams likely focus on getting as many simulation cycles completed during pre-silicon validation as possible. This makes good sense as generally speaking, with random stimulus, the more cycles you run, the more likely you are to flush out remaining bugs. As HPE looked at how to optimize for total number of cycles run, we needed to ask how the results presented in the previous sections impacted this axes of optimization.

When you look at the raw data presented in Section II, we concluded that an individual simulation would run half as fast when running 16 simultaneous simulations. At first glance, this data might steer a team towards using less cores per socket. This is exactly the approach one would take if there was an unlimited amount of servers available. However, most teams have a fixed number of servers and a fixed number of licenses. To illustrate these difference scenarios using numbers, refer to Table 3. In this example, we assumed the number of simulation licenses is fixed and we present data to show how investing in additional servers impacts the overall throughput of your simulation cycles. Given the earlier performance data, it is not difficult to understand that if you purchase more servers so that less cores/socket are used, the resulting faster performance will provide an increase in the overall number of simulations that can be completed.

TABLE 3 : OPTIMIZING TOTAL CYCLES THROUGH MORE SERVERS

Licenses	# two socket servers	cores/socket used	Speed of reference simulation (ns/sec)	Total simulated ms achievable in a year	# of average length (130k ns) sims/year
100	4	13	78.7	248,188	1,909,141
100	5	10	88.6	279,409	2,149,300
100	6	9	93.9	296,123	2,277,870
100	7	8	97.6	307,898	2,368,450
100	8	7	104.3	328,993	2,530,712
100	9	6	109.0	343,691	2,643,773
100	10	5	118.9	374,821	2,883,236
100	13	4	126.2	398,126	3,062,507

100	17	3	137.2	432,746	3,328,816
100	25	2	146.6	462,288	3,556,065
100	50	1	154.7	488,016	3,753,968

However, most of us do not have idle budget sitting around to purchase lots of additional servers. The other side to this area of optimization is to not only fix the number of licenses but also fix the number of servers available. The question that we wanted to answer is whether running fewer licenses than available so that less cores/socket are active will result in more simulation cycles being produced. This is a trivial exercise and it should be no surprise that leaving licenses idle just to force less cores/socket to be in use does not produce enough individual simulation performance increase to yield more overall cycles.

TABLE 4 : OPTIMIZING TOTAL CYCLES THROUGH LESS LICENSES

Licenses	2 socket servers	cores/socket used	Speed of reference simulation (ns/sec)	Total simulated ms achievable in a year	# of average length (130k ns) sims
100	5	10	88.6	279,409	2,149,300
90	5	9	93.9	266,511	2,050,083
80	5	8	97.6	246,319	1,894,760
70	5	7	104.3	230,295	1,771,498
60	5	6	109.0	206,214	1,586,264
50	5	5	118.9	187,410	1,441,618
40	5	4	126.2	159,250	1,225,003
30	5	3	137.2	129,824	998,645
20	5	2	146.6	92,458	711,213
10	5	1	154.7	48,802	375,397

In the Cadence Design Systems experiments, they conducted experiments to see how long it took to complete 16 jobs using different numbers of cores/socket. In this scenario, they would run N parallel jobs sequentially until 16 jobs has been completed. For example, for the one parallel job case, sixteen individual simulations were run sequentially. For the 8 parallel job case, two sets of 8 jobs were run sequentially. As shown in Figure 12, even though individual simulation performance is lower, a total of 16 jobs can be completed more quickly by utilizing many cores.

In these experiments, Cadence Design Systems also considered the case of overloading the number of cores. In this example, they ran a case of 16 parallel jobs on a processor with only eight physical cores. The results were interesting as it showed the jobs took about the same amount of time to complete these 16 parallel running simulations as it took to run two sets of 8 parallel simulations run sequentially. In this case, the individual simulations ran at half the performance when running two simulations per core. What we learn from this is we should never run more simulations than the number of cores that are available.

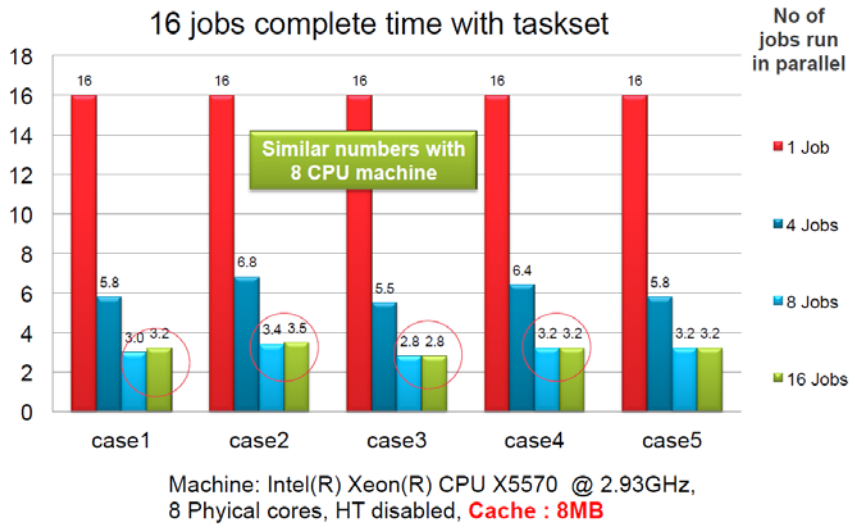
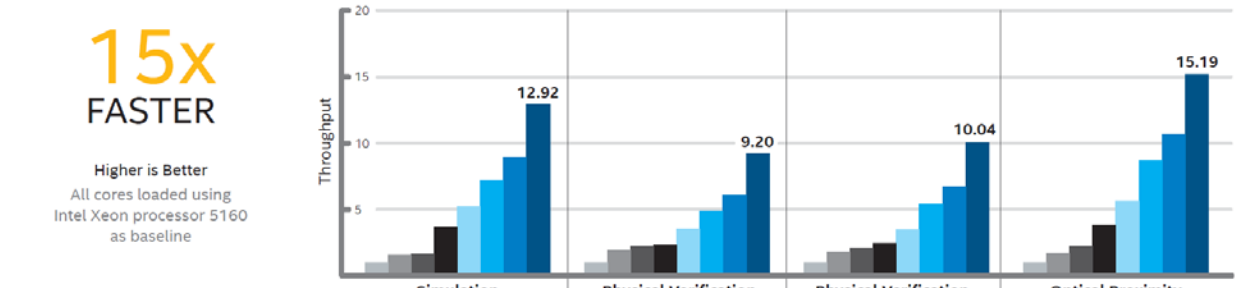


FIGURE 12: CADENCE DESIGN SYSTEMS 16 JOBS COMPLETION TIME [9]

Finally, Intel regularly conducts experiments showing the performance impact of new processor releases. An example of a similar process to the Cadence flow is show in Figure 13. For this test, different processor SKUs each with different numbers of cores were used. In all cases, all cores were utilized and the results are baselined against the processor SKU with two cores. These results also show that utilizing all cores on a socket yields greater overall throughput versus running fewer cores in parallel and serializing jobs to get to your overall throughput goals.



RELATIVE THROUGHPUT	Simulation 113 jobs	Physical Verification Design Rule Check (DRC) 1,680 jobs	Physical Verification Node Antenna Check (NAC) 1,680 jobs	Optical Proximity Correction (OPC) 561 templates
Baseline – Intel® Xeon® processor 5160	1.00	1.00	1.00	1.00
Intel® Xeon® processor X5365	1.58	1.94	1.79	1.68
Intel® Xeon® processor X5460	1.65	2.20	2.06	2.21
Intel® Xeon® processor X5570	3.63	2.33	2.43	3.82
Intel® Xeon® processor X5675	5.20	3.53	3.46	5.61
Intel® Xeon® processor E5-2680	7.22	4.88	5.40	8.71
Intel® Xeon® processor E5-2680 v2	8.94	6.09	6.75	10.65
Intel® Xeon® processor E5-2697 v3	**12.92**	**9.20**	**10.04**	**15.19**

FIGURE 13: INTEL PROCESSOR SKU PERFORMANCE COMPARISON [3]

The conclusion when optimizing for the most overall simulation cycles is to run all the licenses you have spread over as many servers as you have to minimize jobs per server. If you have the funds to purchase more servers to enable running with fewer cores/socket, the result will be greater overall throughput as shown earlier in Table 3. Finally, it should go without stating that one must have at least as many cores available as licenses. Otherwise licenses are sitting idle, or running with very low efficiency.

B. Optimizing Interactive User Job Performance

Another axes of optimization is to focus on minimizing the length of individual simulations versus focusing on overall throughput. This axes is important when you consider the interactive work the individual engineers do each day. Typically an engineer will make changes to their code, whether RTL or verification code, and run a test to see if the desired result is achieved. Engineering resources are extremely expensive so minimizing the turn-around time for an individual engineer's work is very valuable. Additionally, as the simulation length increases for environments such as chip and system level environments, the focus turns to the number of iterations that can be achieved in a typical workday.

When considering optimizing interactive job performance, the performance data shows there is a need to minimize the number of cores/socket being used. At HPE, we have addressed this area of optimization through several paths. First, we utilize a dynamic resource reservation system as described in Section IV D below. Second, we size the number of regression servers overall to optimize the number of cores/socket we use when all licenses are in use for the best tradeoff of cost and performance. Finally, we create specific queues which are configured to minimize the number of cores/socket for user jobs.

C. Optimizing Cost of Results

A considerable amount of data has been presented to this point and the conclusions reached should have not been surprising to the reader. The last area of focus we have had is to look at optimizing the overall cost of results. For this analysis, we looked at the following cost areas for simplicity: software costs and server hardware cost. The calculation formulas used in this analysis are shown below.

```
cost_per_sim_job        = Cost of hardware and software needed to run an average simulation job

cost_per_sim_job        = software_cost + hardware_cost

software_cost           = cost_per_license_second * sim_runtime_seconds
```

```
cost_per_license_second = annual_cost_per_sim_license / seconds_per_year

hardware_cost           = (cost_per_server_second * sim_runtime_seconds) /
                            jobs_per_server
cost_per_server_second  = (HW_cost_amortized_3_yrs + operating_costs) / seconds
```

The graphs below show the cost curve per job when using different number of cores/socket. For Figure 14, the cost of the example software license ($1,000) is significantly less than the cost of the server ($15,000) so the hardware costs initially dominate the job costs. However, eventually, the license costs begin to dominate and the cost of each job grows as more cores/socket are utilized. Figure 15 shows a similar result but the domination of software costs is accelerated since the software cost ($5,000) is about a third of the hardware costs.

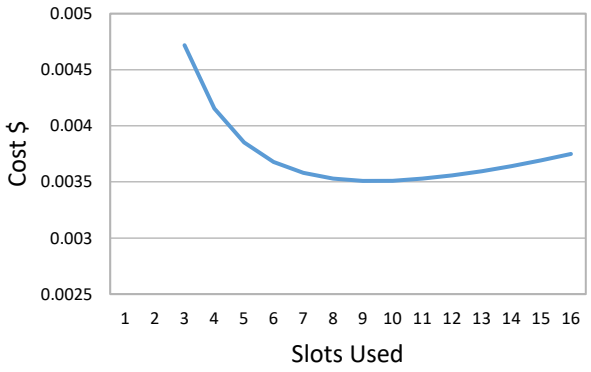


FIGURE 14: TOTAL JOB COST ($1000 SOFTWARE LICENSE)

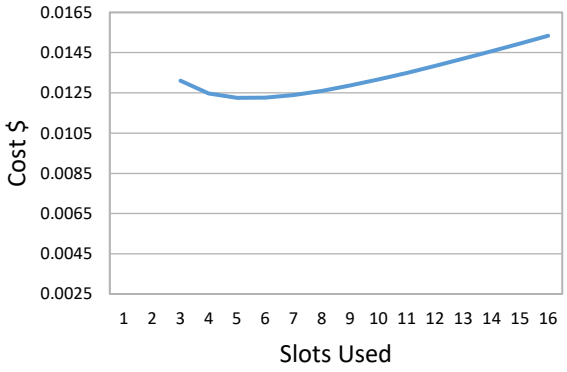


FIGURE 15: TOTAL JOB COST ($5000 SOFTWARE LICENSE)

For very expensive licenses which dwarf the cost of a server, there is generally no cost saving in running many jobs on a single socket as shown in Figure 16. It is easy to miss the financial impact of using either older server or fewer servers on the per job cost of very expensive software tools.

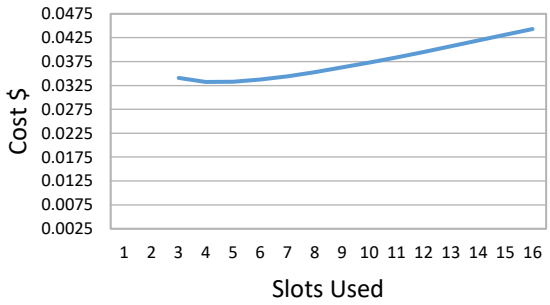


FIGURE 16: TOTAL JOB COST ($50,000 SOFTWARE LICENSE)

As presented previously, adding more servers to your regression farm with a fixed number of licenses would enable less cores/socket to be used and result in greater overall throughput while also positively impacting interactive performance. We analyzed two areas of how additional investment could provide a greater overall benefit to the cost of results. First, we looked at how investing in upgrading servers from one generation to the next would provide overall value to our team. Second, we looked at how investing in purchasing a larger number of servers would provide increased value.

The following figures use a $1,000 software license cost and a $15,000 server cost to show the benefit achieved from investing in an upgrade from Gen8 to Gen9 servers. Figure 17 shows the increase in simulation count throughput achieved from the newer, faster servers. Figure 18 translates this additional simulation capacity into a financial value. Finally, Figure 19 shows the financial value of adding incrementally more servers to your existing farm which results in using fewer cores/socket. This value results from the increased capacity of simulations a team can run with faster hardware. Often it is less expensive to invest in better hardware than to purchase more licenses. In all cases, there is

a definite financial value from investing in hardware. These curves accelerate more quickly as the license cost increases relative to the server cost.

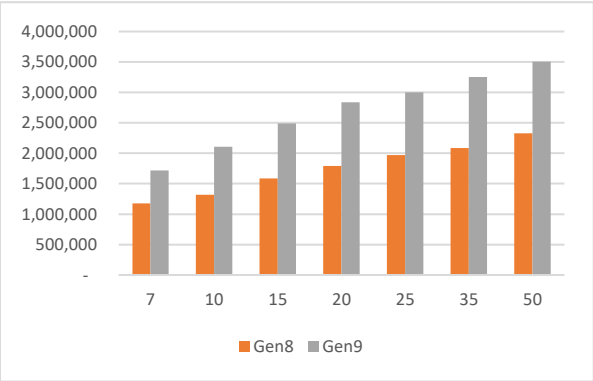


FIGURE 18: INCREASED SIMULATION THROUGHPUT FROM SERVER UPGRADE

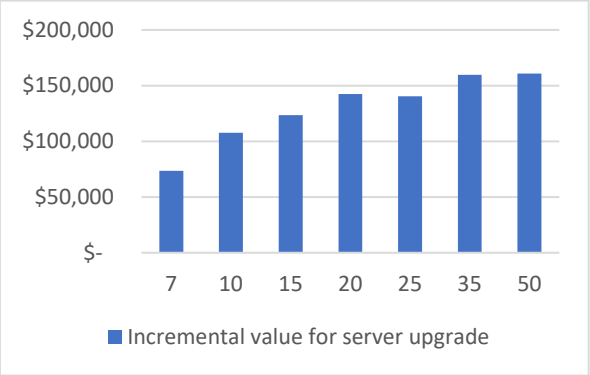


FIGURE 19: INCREASED VALUE FROM SERVER UPGRADE

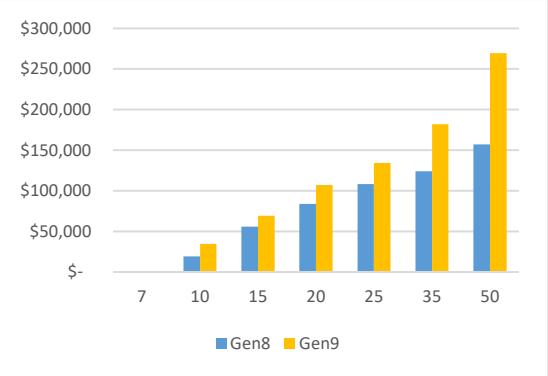


FIGURE 17: INCREASED VALUE FROM ADDING MORE SERVERS

D. Dynamic Resource Reservation

From the analysis above we see that keeping simulation licenses as busy as possible helps to minimize cost of results. But we also need to trade off the impact on productivity of having to wait for interactive simulations to start running. We want engineers to be able to run small groups of simulations with very little time spent waiting to check out licenses, while still using as many licenses as possible for background regression simulations that run 24x7. To balance these needs we define a resource in the queuing system that will specify the number of licenses to reserve for use by user jobs to be able to start quickly. We call this resource the "license buffer". We also set the queuing system to give higher priority to running user jobs since there will generally be lots of regression jobs already waiting in the queue. When background regression jobs are submitted to the queuing system, we add a resource request that says that free simulator licenses must be greater than the corresponding buffer before the job will start running. User jobs on the other hand request only that a license is available. The effect of this is that as user jobs are submitted they will immediately start using the free licenses. This causes the number of free licenses to go below the buffer setting and regression jobs will wait. As completed jobs free up licenses, the free license count will again be greater than the buffer setting, and more regression jobs will be able to start.

Note that the buffer is not a limit on the number of user jobs that can be running, rather the regression jobs "back off" automatically to accommodate more user jobs. As user jobs exit, then regression jobs fill in using any free licenses beyond the buffer setting. Note that since the buffer value is defined as a queuing system resource rather than as a fixed number in the resource request, we can change its value and that change will take effect immediately even for previously submitted jobs waiting in the queue. This means we can setup to have a larger buffer value during working hours, and smaller values at other times, thus improving license utilization, while preserving quick simulation feedback and responsiveness for users.

The plot below shows the buffer setting and actual free license resources on a typical workday on our compute farm. Using metric plots such as this we monitor our license utilization and adjust the buffer settings to minimize times

when user jobs would wait on licenses. The buffer is typically close to the number of simulation jobs that would be submitted by a user to run a quick integration test for a model change.

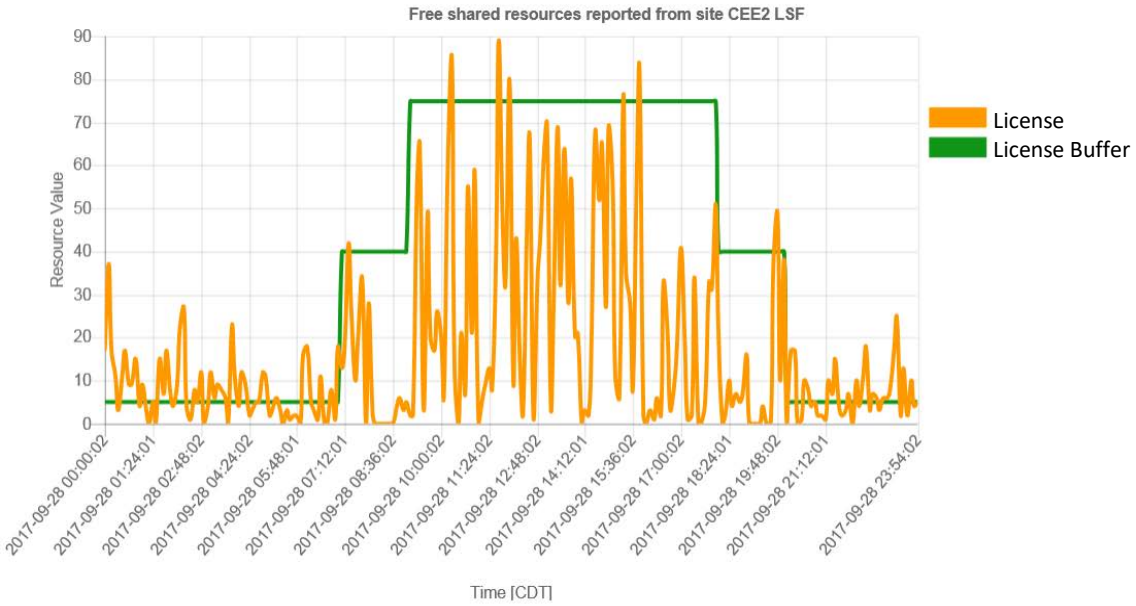


FIGURE 20: SIMULATOR LICENSE UTILIZATION AND LICENSE BUFFER SETTING

V. OPTIMAL REGRESSION FARM CONSTRUCTION

Given the foundation provided, we will bring everything together to showcase how to optimally balance the various factors, including cost, objectives, and licenses, to create the optimal regression farm. We consider many factors in building a compute farm including number of processor cores and memory per host, processor speed, cache size, amount and speed of local disk storage, structure of mounted disks, and network bandwidth.

A. Absolutes for your Configuration

There are a few absolutes when putting the exact configuration of your compute environment together. First, there must be enough physical memory for every job that is running. You never want to run any number of jobs which forces the system into use of virtual memory. This caching to disk will severely impact your job performance. Second, your core count must be sized to at least the number of jobs you will be running. As shown earlier, if you run more than one job per core, the performance of every job is cut in half. Finally, you should never run less than the number of licenses you have purchased just to gain faster individual simulation performance through lower jobs per socket. The performance boost does not provide enough gain to offset the loss of value that results from idle licenses.

B. Optimal Configuration

We have presented a variety of performance and cost data. Ideally, this would lead to one perfect, optimized answer for everyone. In reality, it is never that simple. In this section, we will present our thoughts on how the various factors can be considered in your specific situation.

For the HPE compute environment, we size our compute capacity to target the optimal number of jobs per socket that provides the optimal cost of results. With Gen8 and Gen9 systems, this optimal point was about five to seven jobs per socket. Obviously, budgets are a real part of the equation. In an ideal world, we would always stick with this target. However, there are times when we purchase additional licenses for peak use. In these cases, we do not have the server capacity to stay at 5-7 jobs/socket. During these peak times of our projects, we will run more jobs per socket in order to utilize the peak license capacity. From this, the reader should understand that there is flexibility build into the equation for an optimal configuration.

Our regression launching infrastructure is setup to direct simulation output to a disk that is local to the server running the simulation. Directing output to the local disk instead of directing all the simulation output across the network to an NFS disk drives increased performance. If your budget allows, investing in SSD drives for the local disks instead of spinning disks will also yield a performance increase. The amount of performance gain for this portion of our

configuration is highly variable, depending on factors such as verbosity settings, depth of traces, filtering of data returned, etc.

When we choose our processor SKU, we balance a number of factors. First, there is a high priority put on average cache per core. We have found higher average cache per core capacity often comes with smaller core count processors. We value maximum cache per core above clock speed and clock speed is rarely a key factor in our selection. Cost is also a factor as we want to stretch our budget as far as possible. However, by focusing on cache size, we have found that this tends to lead us to the best value. A variety of different server configurations are typically purchased to address different workloads and support different license needs. For instance, servers used for simulations typically require significantly less memory than servers used for physical builds. By sizing server configuration to major workload types, we are able to maximize the use of our budget.

C. Value of Regular Hardware Refreshes

As a final note on optimal configuration of your compute environment, the data clearly shows there is substantial business value gained from regular refreshes of your servers. HPE typically targets a three year refresh cycle but often will see a four year refresh cycle in practice. This is the result of the need to increase the capacity of our environment when our license pool increases or when budgets need to be trimmed. It is advised that you use the data shown in the various papers referenced to convince your own management team of the lost business value of not conducting regular refreshes of your computer environment.

VI. Tracking Simulation Performance

Once you have an optimal regression farm, it is important to track metrics to ensure that the performance results are maintained over the course of the project. HPE tracks a variety of metrics automatically that clearly highlight when simulation performance has been negatively impacted. For example, tracking the average CPU utilization on hosts in the compute farm can help determine if too many jobs are being run such that execution is outside the desired range for optimal simulation throughput cost. CPU utilization will be reduced when the processor cores are waiting on memory requests. We track license use by business unit over time to enable us to determine the optimal setting for license reservation buffers.

This data is available through both the queuing software we use to schedule our jobs as well as the license manager. We archive data from both sources to ensure that we can understand the perspective of both tools when licenses are not managed as we desire. This also allows us to track the proper fair-share allocation of licenses between business teams and projects.

Having lots of metrics recorded and quickly accessible is also important when we need to diagnose problems. The metrics that are automatically collected allow us to quickly determine the source of a problem and keep the compute farm running at maximum efficiency.

VII. References

[1] IT@Intel, “Increasing EDA Throughput with New Intel Xeon Processor E5-2600 v2 Product Family,” October 2013 [http://connectedsocialmedia.com/11126/increasing-eda-throughput-with-new-intel-xeon-processor-e5-2600-v2-product-family/].

[2] S. Krishnapura, V. Lal, T. Tang, S. Achuthan, M. Ayyalasomayajula, “Increasing EDA Throughput with New Intel® Xeon® Processor E5-2600 v3 Product Family,” September 2014 [https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/high-performance-servers-paper2.pdf].

[3] S. Krishnapura, V. Lal, T. Tang, S. Achuthan, M. Ayyalasomayajula, “Increasing Design Throughput with Workstations Based on New Intel Xeon Processor E5-2600 v3 Product Family,” September 2014 [http://media14.connectedsocialmedia.com/intel/09/11996/High_Performance_Workstations_Increase_Design_Throughput.pdf]

[4] Bruce Wile, John Goss, Wolfgang Roesner, “Comprehensive Functional Verification: The Complete Industry Cycle,” Morgan Kaufmann, 2005.

[5] Intel, “Intel Xeon Processor E5-2699A v4 Product Specification,” [https://ark.intel.com/products/96899/Intel-Xeon-Processor-E5-2699A-v4-55M-Cache-2_40-GHz].

[6] Hewlett Packard Enterprise, “HPE ProLiant DL380 Generation9 (Gen9) Overview”, [https://www.hpe.com/h20195/v2/GetDocument.aspx?docname=c04346247&doctype=quickspecs].

[7] SanDisk, “CPU Bandwidth – The Worrisome 2020 Trend”, March 23, 2016 [https://itblog.sandisk.com/cpu-bandwidth-the-worrisome-2020-trend/]

[8] GenZ Consortium, “Gen-Z Technology: Enabling Memory Centric Architecture”, Flash Memory Summit, August 8, 2017 [https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2017/20170808_FB12_Bowman.pdf]

[9] Dua, Amit, “Hardware Recommendations for Incisive Simulator Performance”, 2012 [Cadence Design Systems]