Compute Job Memory Recommender System Using Machine Learning

Taraneh Taghavi
Qualcomm Technologies Inc.
San Diego, CA, USA
ttaghavi@qti.qualcomm.com

Maria Lupetini
Qualcomm Technologies Inc.
San Diego, CA, USA
marial@qti.qualcomm.com

Yaron Kretchmer
Qualcomm Technologies Inc.
Santa Clara, CA, USA
yaronk@qti.qualcomm.com

ABSTRACT

This paper presents a machine learning approach to predict the amount of compute memory needed by jobs which are submitted to Load Sharing Facility (LSF®) with a high level of accuracy. LSF® is the compute resource manager and job scheduler for Qualcomm chip design process. It schedules the jobs based on available resources: CPU, memory, storage, and software licenses. Memory is one of the key resources and its proper utilization leads to a substantial improvement in saving machine resources which in turn results in a significant reduction in overall job pending time. In addition, efficient memory utilization helps to reduce the operations cost by decreasing the number of servers needed for the end-to-end design process.

In this paper, we explored a suite of statistical and machine learning techniques to develop a Compute Memory Recommender System for the Qualcomm chip design process with over 90% accuracy in predicting the amount of memory a job needs. Moreover, it demonstrates the potential to significantly reduce job pending time.

Keywords

Memory Utilization, Compute Memory Recommender System, Statistical and Machine Learning Techniques, Grid Computing.

1. INTRODUCTION

In a large scale semiconductor firm like Qualcomm, compute and license environment provides the infrastructure and resources to enable the full cycle of the chip design process. These resources include hardware infrastructure (CPU, storage, memory), software licenses, and etc. The amount of compute resources needed by an end-to-end chip design process is substantially dependent on the design complexity through the entire design cycle and specifically prior to the final stage of it where the design is shipped for manufacturing, also known as "Tapeout".

A key challenge in any large scale semiconductor company is to effectively utilize the compute resources and optimally allocate them to parallel chip design processes. Effective utilization of compute resources yields a significant impact on setting the cost of the end-user devices which helps improving company's position in today's competitive market.

Machine memory, as one of the key compute resources of any job execution, needs to be utilized effectively to improve the efficiency of the design process and prevents any major slowdown or halt in the design flow. Over-requesting memory by a job causes other jobs to go into a pending state since they are using shared pools of resources. On the other hand, under-requesting memory by a job may lead to forceful exit of the job due to lack of required memory to successfully finish the job. Hence an effective and asymmetric memory allocation methodology needs to be adopted to ensure that all the jobs are receiving their needed memory requirement for proper execution.

In this paper, we explored several different models from classic statistical methods to more modern machine learning techniques to develop a memory recommender system based on the historical job execution data on a grid computing platform. Our model is capable of handling large volume of data for training and prediction and can be easily ported to a cloud computing architecture platform.

KDD'16, August 13–17, 2016, San Francisco, CA, USA.

DOI: http://dx.doi.org/10.1145/2939672.2939717

2. BUSINESS OPERATIONS OVERVIEW

2.1 Chip Design Process

Integrated circuit design, or IC design, is a subset of electronics engineering, which consists of logic and circuit design techniques required to develop and build an IC [1]. The initial chip design process begins with system-level design and architecture planning. This step is where a chip functionality and design is decided. Upon agreement of a system design, the engineers implement the functional models in a hardware description language like Verilog, System Verilog, or VHDL. The next step in the IC design process, physical design phase, is to map the design into actual geometric representations of all electronics devices, such as capacitors, resistors, logic gates, and transistors that will go on the chip. Since errors are expensive, time consuming and hard to spot, extensive testing needs to be done to make sure the mapping to transistors was done correctly and the manufacturing rules were followed faithfully. At the final stage of the design cycle for ICs, Tapeout, the artwork for the silicon photomask of the circuit is sent to be manufactured. After the chip is manufactured and returned back to the design company, massive testing is done to make sure all parts are working according to the specifications. Figure 1 shows the high level chip design process.

One of the main differences between chip and software design is the high cost of fixing design errors. Errors that are not corrected by the time silicon masks are manufactured usually require re-spins of some or all of the masks associated with significant capital expense and opportunity costs, frequently exceeding $10 million. As a result, much of the discipline of chip design is involved with early and frequent verification of the chip as it goes through the design process. The complexity of modern chip design, as well as market pressure to produce designs rapidly, has led to the extensive use of Electronic Design Automation (EDA) software for design, verification andp simulation of different steps of the process. This entails utilizing massive amount of compute resources such as hardware, operating software, and EDA software licenses for simulation, design, and testing stages needed for a full cycle of a chip design. To have a successful and cost efficient design process, available compute resources such as CPU, memory,

storage and automation tools and licenses have to be optimally utilized.

2.2 Load Sharing Facility (LSF®[1]) Scheduler

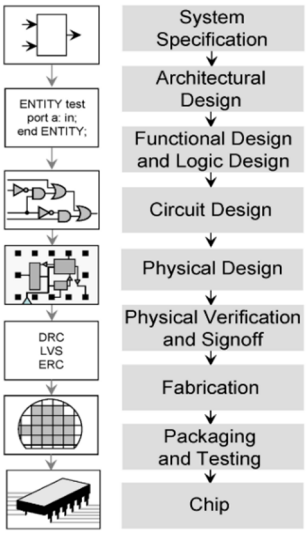

Figure 1. Major Chip Design Steps [1]

In A Grid Environment

IBM Load Sharing Facility (LSF®) is a workload management platform and job scheduler for distributed high performance computing environments. It can be used to execute batch jobs on networked UNIX and Windows systems on many different architectures. It provides a comprehensive set of policy-driven scheduling features that enable users to take full advantage of the available compute infrastructure resources needed for successful job execution [2][10].

In a large scale design process with massive amounts of compute resources, there is a need for a scalable, fault-tolerant, and cost efficient computing power, such as that provided by a grid environment. Grid environments consist of a collection of clustered computers which form a heterogeneous grid computing foundation for a full cycle of design process. The grid environment can be thought of as a distributed system with non-interactive workloads and the ability to perform different tasks and applications [2][12].

The ability to handle scheduling of massive number of jobs on a heterogeneous network of resources make LSF® a viable choice as a workload manager to be utilized by chip design process in many semiconductor companies including Qualcomm Incorporation.

2.3 Compute Resource Management

Compute infrastructure and resources used in the chip design process make up a big portion of the total cost of the chip. An inefficient allocation of resources can cause a significant increase in job pending time which leads to inevitable delays in Tapeouts. Furthermore, it can reduce the productivity of the engineers while they wait a longer time for job completion. For these reasons, sustainable solutions for optimal utilization of compute resources are of utmost importance in a competitive market.

Using manual or trial and error methods to allocate compute resources needed for jobs are not regarded to be effective. In addition, due to the volume and complexity of the submitted jobs in a chip design cycle, it is very difficult, inefficient, and at times impossible for users to accurately predict the needed compute resources for successful job executions based on their experience. Hence, a need for machine learning intelligence emerges to effectively predict the resources needed in the chip design process.

3. BUSINESS CHALLENGE

In a grid environment, workload scheduling is done at global and local levels to increase efficiency and scalability. Job submission scripts are responsible for requesting computing resources at the local level and LSF® scheduler is responsible for communicating with all job submission scripts to manage the workload at the grid level. Hence, smart resource requesting mechanisms by job submission scripts enable efficient resource allocation by the LSF® scheduler.

Every job submissions script is configured for a specific type of job and includes the job parameters and paths, files and formats, selected EDA software products, and required compute resources. Job submission scripts are generally developed and maintained within design teams and are specific to each team's requirements. Compute resource requirements for each job is then specified by the team lead or the engineer based on their knowledge of past jobs and priority of the submitted job.

In general, engineers tend to over-request the amount of compute resources including memory in order to make sure their jobs execute successfully. This phenomena can result in a substantial over-request of memory requirements once they get accumulated over all the jobs submitted in a grid environment. Figure 2 shows

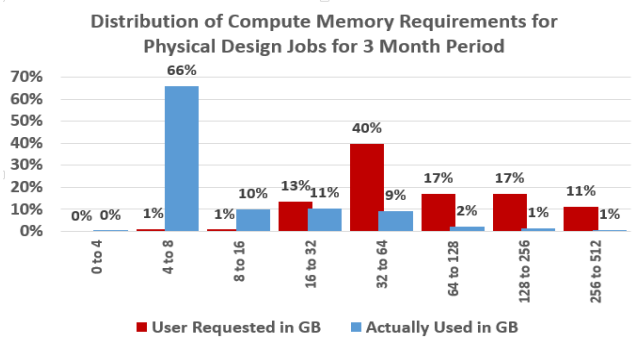

Figure 2. User Requested vs Actually Used Memory (GB)

that 87% of a specific type of EDA jobs (physical design) for a three month period actually consumed 1MB to 32GB memory, while the users requested 1MB to 32GB of memory for only 17% of the jobs.

Figure 3 illustrates the same point from another perspective, memory by run time. It shows 36% of jobs actually need more than 1 GB-Hour memory, in comparison to 81% of jobs requested more than 1 GB-Hour memory. These statistics exhibits great amount of memory over request. The main consequence of this over-request is that jobs have to wait for longer times in a pending state for memory resources to become available which results in longer design cycles. It also impacts engineer's efficiency as they have to wait longer for their jobs to complete which contributes to higher

[1] LSF® is a trademark of Platform Computing Corporation and is licensed exclusively to IBM.

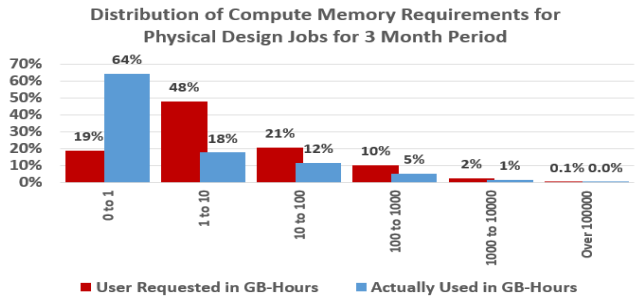

Figure 3. User Requested vs Actually Used Memory x Run Time (GB x Hour)

labor cost. Moreover, to accommodate for the peak request of memory resources, servers with larger memory blocks have to be purchased which leads to significant increase in operations cost, potentially resulting in higher price point for end-user devices, or lower profit margins.

These statistics demonstrates the huge potential for improvement in the memory allocation efficiency within Qualcomm's grid environment. An initiative was launched to explore this potential with the goal of closing the gap between the amount of memory requested by a job and the actual memory a job utilizes after it completes. A set of historical data on job submission was collected and an entire suite of techniques from statistical and machine learning paradigms were explored to build an efficient and accurate memory recommender system which includes dynamic user and management level dashboards, memory violation alerting system, and memory recommender engine. In the next two sections we explain the content of the data and the development of our recommender system.

4. DATA ANALYSIS, VISUALIZATION AND FEATURE SELECTION

At the start of the project, our approach was to build a predictive model that would consider the phase and week of the chip design life cycle, the intensity of the chip design analysis and testing at that point in time, plus the EDA software being used at that phase.

As shown in Figure 1, major chips follow this design flow. However, each chip is unique in the pace of its project plan and amount of required software licenses and compute resources. Within the available real-time data, the knowledge of how much of each resource is needed for the individual chip design was not readily available.

To better understand the basic patterns in the data we used descriptive statistics, machine learning, and visualization methods to mine the raw data. Some of our early findings were:

- Some design centers consume more memory than others for similar jobs
- Some contracting firm's temporary workers wasted more memory than others firms
- There are different patterns of memory requested by employees versus non-employees
- Work experience reflected by engineer title correlates with different memory use patterns
- The task influenced the amount of needed memory
- Individual user patterns are distinct
- The number of processors influence memory needs for a job
- The memory used in prior jobs is a good guess for subsequent jobs

The last four points from above became crucial features (variables) for our modeling efforts. These early findings confirmed our understanding of the behavior of a chip designer and test engineer. An engineer typically executes one design task or testing task for several hours up to several days. Their work will require the executions of computer jobs using only a few EDA tools out of many available.

We were able to exploit fields in the job submission scripts captured in the database that indicated the chip project under development, the phase of effort, (i.e., physical design vs physical implementation), the EDA software used, and the specific task (i.e., timing analysis vs floor-planning) for every engineer. Capturing this behavior was an essential factor in developing our compute memory prediction model. Figure 4 demonstrates the unique behavior for seven engineers during the same month for the same project. Some physical design engineers carried out all three tasks: chip timing, parasitic extraction and physical verification, some only one or two of these tasks. In addition, one can see, the average number of memory in gigabytes (GB) varied by task and engineer.

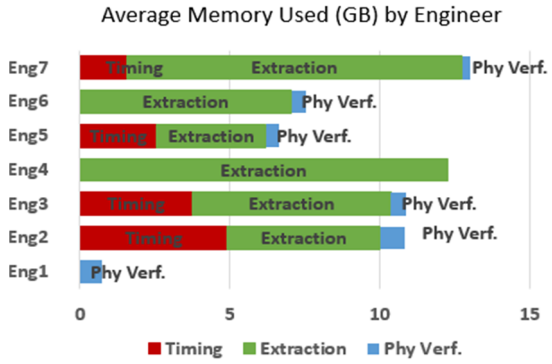

Figure 4. Sample of Memory Use by Physical Design Engineers for Project A during One Month.

We also considered the number of compute processors requested for each job, the user's location, employment status (employee vs contractor), and engineer's department as possible predictors. In the end, the engineer's department, title, and employment status did not become part of the predictive model. The number of compute processors needed for the job did become an important variable in the model.

Finally, a key behavior captured in the model was the memory needed by past similar jobs. Memory used by past computer jobs, project, phase, task, and number of needed processors submitted by an engineer became the main variables in our predictive model.

5. BUSINESS REQUIREMENTS

One of the main business requirements was to build a predictive model which substantially minimizes compute memory waste over current method driven by engineering teams based on their project needs. In addition, the model should be robust enough to predict the amount of memory a job needs with high accuracy consistently over varying chip projects and time frames. The model and its subsequent implementation should be easy to understand, deploy and maintain and must be able to handle the data volume of more than 1.4 billion computer jobs executed by Qualcomm every year.

Another important business requirement was to limit under-prediction below a certain threshold. If a job is not assigned enough memory for its completion, the LSF® scheduler can be configured to gracefully grant more memory blocks to that job if the machine has enough available memory resources at run time. But if the machine is running out of memory resources, the job is terminated if it does not have enough memory reserved for its execution a

priori. Job interruption carries an extremely exorbitant cost due to the high potential of delaying chip design.

6. STATISTICAL AND MACHINE LEARNING METHODS EXPLORED

In order to fulfill our business challenge with the given requirements, we explored an entire suite of models from classical statistics to more modern machine learning paradigms. Below we discuss the pros and cons of some of the best performing models we explored for this problem. In the next sections, we will explain the model selection process considering the tradeoffs between parsimony versus accuracy of different models. Here is the list of all models we explored:

1. *Neural Network (NN):* We used a multi-layer perceptron with one hidden layer. Neural network is generally very effective on a wide range of problems and can detect non-linear patterns very well even for very noisy or dirty data [15]. Although NN is a quite powerful tool to explore the patterns in the data, it has several disadvantages. The main disadvantage is that the generated model is not easily representable and is generally treated as a black box. Furthermore, due to the iterative nature of model training, the training run time is generally large specifically for big size data sets [3]. For these reasons, NN is not the best solution for our problem unless it shows a clear performance improvement compared to other models.
2. *Autoregressive Integrated Moving Average (ARIMA):* ARIMA is a time series analysis technique which can be tuned by using autoregressive, differencing and moving average parameters [4]. We built ARIMA models separately for every different task after we sorted our data by user, number of processors, and project in decreasing order of job submission time. After investigating stationarity and seasonality in our data, we determined that autoregressive models show the best performance among all ARIMA models, hence we included AR(3) and AR(11) in our model comparison. The main advantages of AR models are that they are simple and can easily be maintained. However, the main disadvantage is that they assume the same fundamental patterns exist over time which makes it somewhat not suitable for our purpose due to the change in requirements for different projects at different points of time.
3. *Kalman Filter (KF):* This time series analysis technique uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more precise than those based on a single measurement alone [5][13]. We used "astsa" package in R for our analysis of Kalman Filter. Like ARIMA, we build separate KF models by task. Even though KF models proved to be effective, model complexity is their major drawback.
4. *Multivariate Adaptive Regression Spline (MARS®)*[2]*:* This technique is a multivariate piecewise linear regression with adaptive knot selection process [6][14]. MARS® model is a very powerful tool to estimate non-linear patterns with piece wise splines and provides a somewhat light-weight model that can easily be deployed in real-time applications.
5. *Linear Regression (LR):* Least squares linear regression model is a simple and easy to deploy and maintain model which makes it a viable option for our problem. However, it is not capable of finding non-linear patterns in the data if they exist. Similar to the ARIMA model, we applied a linear regression model with an option of look backs of 3 and 11 lags. In addition, we build individual linear regression models for a combination of user, project, task and number of processors at each phase.
6. *Classification and Regression Trees (CART®)*[3]: Decision tree models are recursively partition the data space and fit a simple prediction model within each partition [16]. The main advantage of a CART® decision tree is its interpretability which makes it a suitable option for our problem, but the down side is that the tree size can get large quickly even with pruning.
7. *Chi-square Automatic Interaction Detector (CHAID):* CHAID performs multi-way split and uses Bonferroni adjustment to control tree size to overcome the disadvantage of the CART® model [7]. Unlike linear regression models, decision trees are capable of detecting non-linear patterns in the data, but generate more complex models.

7. EXPERIMENTAL RESULTS

To perform our experiments, we used the R statistical software and its packages and SPM Salford Predictive Modeler®[4]. We compared all the models we explored to the existing solution, which is to allocate memory based on user's request, specified in submission script. The results in this paper are based on the training data set of size over 0.5 million records which are collected for July 1 to Sep 30, 2015 time frame. The trained models were then used to predict the amount of memory needed on the testing set collected for Oct 1, 2015 for one major compute grid on all tasks for one specific phase, Physical Design. Table 1 captures the statistical metrics to compare different models using the test set. Training time for each model is also reported in the last column of this table.

Table 1. Statistical Metrics for Model Comparison

Model	R-SQ	RMSE (GB)	AIC	% Under Predicted Jobs	Training Time (min)
Requested by User	12%	84.3	NA	5%	NA
NN	88%	34.7	9330	58%	33
AR(3)	92%	32.6	9148	18%	8.5
AR(11)	93%	32.4	9152	16%	9.5
KF	94%	32.0	9118	49%	15
MARS®	92%	32.5	9158	54%	3
LR(3)+UPI$_{95}$	93%	32.6	9149	5%	1.1
LR(11)+UPI$_{95}$	93%	32.5	9156	5%	1.2
CART®	87%	35.2	9366	26%	5.5
CHAID	90%	33.8	9260	17%	6

[2] MARS® is a trademark of Jeril, Inc. and is licensed exclusively to Salford Systems.

[3] CART® is a trademark of California Statistical Software, Inc. and is licensed exclusively to Salford Systems.

[4] SPM Salford Predictive Modeler® is a registered trademark of Salford Systems.

As described in business requirements section, our major challenge was to make sure we limit under-prediction below a selected threshold due to severe consequence of under-allocation of memory slots to a job. Since currently about 5% of the jobs are submitted with memory request less than what actually is needed by those jobs, it is reasonable to set the threshold for under-prediction to be 5%. For this purpose, we build a 95% statistical upper prediction interval for our linear regression models using (1), where n is the data set size, X is the design matrix, S is the model standard deviation, and $t(0.05, n-2)$ is the upper 95% quantile from t-distribution with $n-2$ degrees of freedom [8].

$$UPI_{95} = t(0.05, n-2)\sqrt{S^2(1 + [1\ x](X^tX)'[1\ x]^t)} \quad (1)$$

We added the calculated prediction interval to our predicted values generated by the linear regression model to statistically guarantee that 95% of the jobs will be allocated memory more than what they actually need using our recommendations. Percentage of jobs with under-predicted amount of memory for each model is also reported for comparison.

In addition to minimizing the number of jobs which over-request memory, the recommender system must also minimize the amount of memory over-requested by long-running jobs. Once a job with a large run time acquires the memory amount it asks for, it will hold on to this amount until the job finishes. There are scenarios where a model results in smaller number of jobs with memory over-request compared to another model, but the jobs in the first model which over-requested memory might run for a longer time. To make sure we account for this situation we used a second set of metrics to compare different models as shown in Table 2.

These measures evaluate the ability of each memory recommender model to reduce wasted memory (GB) times the hours of the jobs' run time. We evaluated our models' behavior over different data sets over time. Average amount of over-prediction (GB) and over-prediction by run time (GB x Hour) are shown in Table 2. Average amount of under-prediction (GB) and under-prediction by run time (GB x Hr) is also recorded in this table so we can compare the behavior of different models in terms of under-prediction, specifically due to its potential sever consequences as described in section 5.

7.1 Model Selection Trade-offs

As Table 1 illustrates KF model shows the best R-SQ and the lowest RMSE. On the other hand, AR, MARS® and LR+UPI$_{95}$ model performances are also close to that of KF in these regards. The disqualifying factor for using KF and MARS® models is their high percentage of under-predicted jobs which violates our business constraints. AR models show significantly better result on their percentage of under-predicted jobs, but they are still above the threshold of currently used method which is requested by user. NN, CART® and CHAID model performances are not up to par with other models.

Table 2 shows the average under and over prediction per user for both memory (GB) and memory by run time (GB x Hr) which are consistent with the model performance results of Table 1. All models significantly improve the amount of over prediction over the current model. The KF model results in the least amount of over-prediction trading off larger amounts of under-prediction compared to AR and LR models. LR(11)+UPI$_{95}$ shows the lowest values on average under-prediction both for memory (GB) and memory by run time (GB x Hr), but it shows an slight increase in AIC/BIC values compared to LR(3)+UPI$_{95}$ model as shown in Table 1.

Considering all the trade-offs of model performance, parsimony, fulfillment of business constraints, and maintainability, overall we determined the LR(3)+UPI$_{95}$ model to be the most suitable solution for our memory recommender engine. Figure 5 shows the distribution of requested versus actual and predicted versus actual memory using LR(3)+ UPI$_{95}$ model for "Physical Design" phase on one compute grid which represents about 13% of all jobs submitted using EDA software in Qualcomm. As it can be seen in this figure, the predicted versus actual plot shows a significant performance improvement over the requested versus actual plot. The correlation of determination (R-squared) of the requested to actual memory was a dismal 12%, whereas the R-squared of the predicted to actual memory was an impressive 93%.

Table 2. Operational Metrics for Model Comparison

Model	Average Over Prediction (GB)	Average Over Prediction (GB x Hr)	Average Under Prediction (GB)	Average Under Prediction (GB x Hr)
Requested by User	365.6	1948.3	13.1	108.0
NN	138.7	1071.4	45.5	399.6
AR(3)	65.7	759.7	16.9	194.4
AR(11)	62.0	681.8	14.3	183.6
KF	47.4	642.8	42.9	324.0
MARS®	64.2	818.2	36.4	291.6
LR(3)+UPI$_{95}$	66.8	769.5	14.3	109.1
LR(11)+UPI$_{95}$	65.7	701.3	13.6	108.5
CART®	135.0	1032.4	41.6	388.8
CHAID	127.7	993.5	40.3	378.0

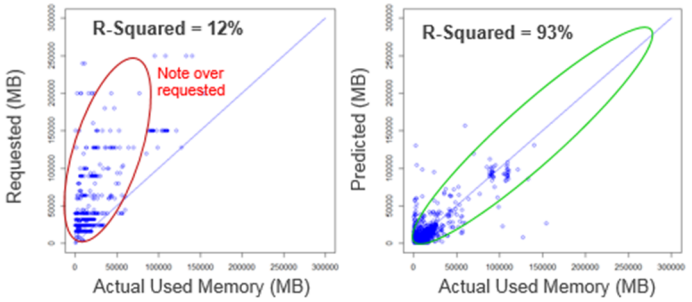

Figure 5. Plots of Requested by User and Predicted using LR(3)+UPI$_{95}$ Model versus Actual Memory Used

Besides memory requirements, another constraint which a job scheduler like LSF® needs to consider is the amount of CPU resources on a machine at the time of scheduling a job. Since CPU resources are associated with blocks of memory of size 4 or 16 GB, a useful metric to compare performance of a model is the histogram of the mismatch between predicted and actual memory built by discretizing the range of memory into bins similar to a confusion matrix. Figure 6 demonstrates the improvement in histogram of mismatch of memory bins between what a job requests versus what it actually uses when we use LR(3)+UPI$_{95}$ prediction model. The figure clearly points out the substantial improvement in terms of matching the predicted and actually used memory bins of size 16GB.

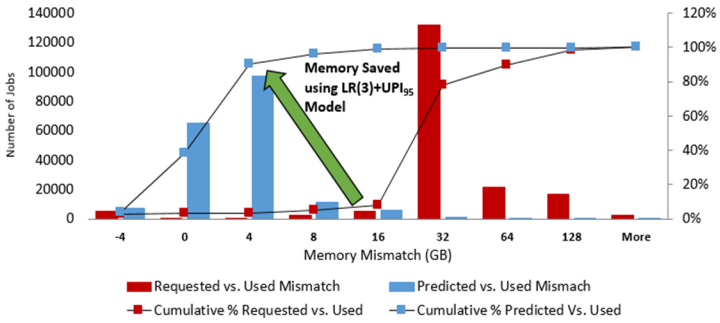

Figure 6. Improvement in Distribution of Memory Mismatch between Actual versus Predicted using LR(3)+UPI$_{95}$

Furthermore, to investigate the average performance of our model over time, we executed a comprehensive experiment which simulates the use of our model on a rolling window of 50 days. For the first day of the experiment, we built a model based on the data of last three months and applied the model to predict the amount of memory needed for all the jobs ran on that day for "Physical Design" task. Then we advanced our training set forward by one day, meaning we included the data of the actual memory used in that day in the training set. We re-modeled using this incremental change in our training set and applied the model to predict the amount of memory needed for the jobs ran on the second day and so on. Figure 7 illustrates the result of this comprehensive experiment. It compares the percentage of correct bins on the diagonal of confusion matrix of our model (shown in blue) versus the same value when the memory is requested by the user (shown in red). It can be seen that the percentage of entries on the diagonal of the confusion matrix improved from 15% to 91% on average by using our predictive model.

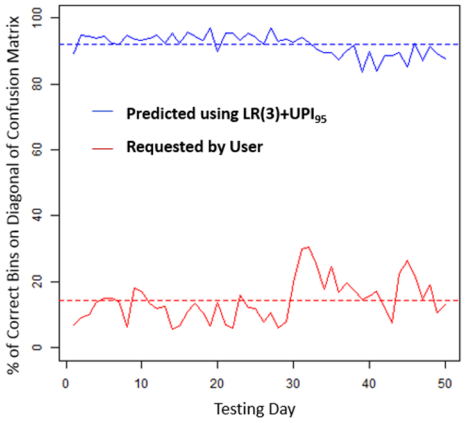

Figure7. Comprehensive Experiment to Assess Overall Performance of our Prediction Model (LR(3)+UPI$_{95}$)

In the next section we describe the alerting system we developed based on the above predictive model using Splunk®[5] software.

8. SPLUNK® ALERTING SYSTEM

Splunk® is a software developed by the Splunk Corporation for searching, monitoring, and analyzing machine-generated big data, via a web-style interface. Splunk® captures indexes and correlates real-time data in a searchable repository from which it can generate graphs, reports, alerts, dashboards and visualizations [9][11].

We developed our alerting system through the Splunk® alerting application in two parts: model training and alert triggering. Model training is setup as a cron job using Splunk® scheduler which triggers every day at a specific time. This time is different for every grid depending on geographical location of that grid. We will be able to capture engineers' daily behavior of job submission to different projects and design phases. Once the model training completes, the linear regression model coefficients and upper prediction interval values for every combination of user, project, phase, task, and number of processors is populated into a Splunk® index for that day. The alerting application then gets triggered which queries this index to predict the amount of needed memory for every submitted job by every user and returns back the top 10% violators, users with highest mismatch between their requested and actually used memory. Those users receive an automated email with a report of the jobs they submitted and the amount of memory they requested and used along with our recommendation. They also receive a Splunk® dashboard which visualizes their memory usage and our recommendations over the jobs they submitted on that day.

9. SPLUNK® MANAGEMENT DASHBOARDS

Besides user dashboards, a suite of management dashboards were developed which aggregates and summarizes the memory usage at different hierarchical levels. Figures 8 and 9 illustrate an example of such dashboards. The amount of used, requested and recommended memory in terabytes (TB), and memory by run time (TB x Hour), aggregated by project are shown in these tables.

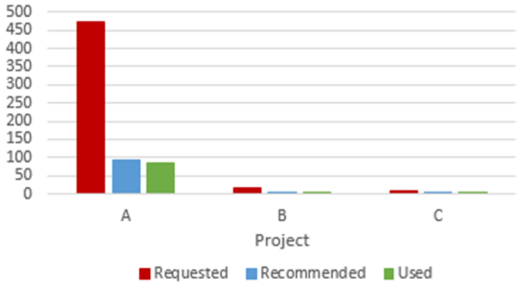

Figure 8. Requested vs. Recommended vs. Used Memory (TB) by Project

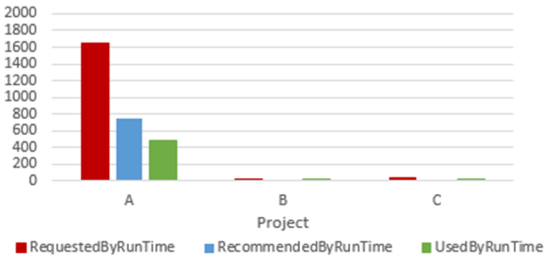

Figure 9. Requested vs. Recommended vs. Used Memory x Run Time (TB x Hour) by Project

Figures 10 and 11 show the same statistics aggregated at user level shown for top 10 violators. To preserve confidentiality, we substituted project and user names with generic names. It can be seen in all these figures that our recommender engine can greatly

[5] Splunk® is a trademark of Splunk Inc.

improve the memory allocation efficiency. Daily dashboards of these sorts are helping management team with tracking problematic areas from memory allocation efficiency standpoint leading to substantially optimizing operational cost.

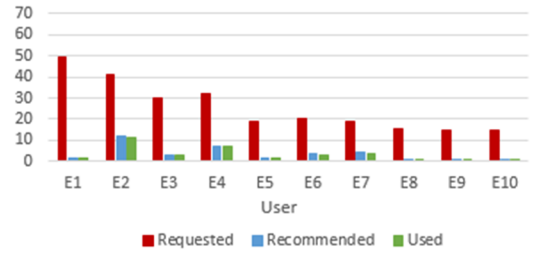

Figure 10. Requested vs. Recommended vs. Used Memory (TB) for Top 10 Violators

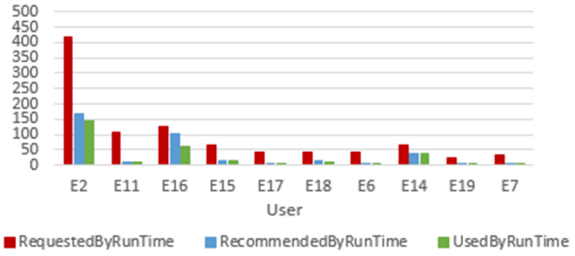

Figure 11. Requested vs. Recommended vs. Used Memory x Run Time (TB x Hour) for Top 10 Violators

10. DEPLOYMENT OF COMPUTE MEMORY RECOMMENDER SYSTEM

In order to integrate our recommender system as part of the chip design process with minimal disruption to this process, we decided to develop and deploy this system in three phases. Full development of this system was done using Python SciPy library in conjunction with the Splunk® alerting and dashboarding application. Here are the deployment phases of our recommender system:

Phase 1. We developed a comprehensive monitoring system to detect the behavior of individual engineers in terms of their memory requests. For this purpose, we set up an informational alerting mechanism to alert the top 10% worst violators. Alerts were sent through the Splunk® alerting application as explained in previous section. The alerts are originally targeted for engineers who submit jobs in "Physical Design" phase on two major grids, but are intended to be expanded to other grids and to other functional areas in third phase. Besides the alerting system, we provided visualization dashboards for both engineers and management teams through the Splunk® dashboarding application.

Phase 2. In this phase we develop a memory recommender engine using Python Sci-Py libray and deploy this engine as part of the EDA flow for "Physical Design" tasks to substitute the user-specified requests for memory with the automated recommendations generated by this engine. We provide users with the capability of overwriting our recommendations with their own memory requests if they choose to do such. However, they will be alerted if they have over-requested memory greatly.

Phase 3. In the last phase we will expand deployment of our memory recommender system (including the automation engine and monitoring and alerting system) to other design phases besides "Physical Design".

11. IMPROVEMENT OF JOB PENDING TIME

Submitted jobs compete with each other to acquire their requested compute resources such as processors and memory. They are dispatched for execution by LSF® scheduler which evaluates the jobs' priority and resource needs. When jobs are highly over-requesting memory, they have to wait longer to acquire their requested amount of memory to be executed. The situation gets even worse when a job with a long run time over-requests memory. Since run time of jobs are not known a priori, the scheduler cannot plan ahead for these types of jobs. Once these jobs get launched, they cause other jobs to wait in a pending state for a longer time before the scheduler is able to dispatch them for execution. To evaluate the effect of using our prediction model on reducing job pending time, i.e., the time computer jobs wait to be dispatched for execution, we need to simulate LSF® behavior. IBM has developed a simulator for LSF® for their internal use, but do not provide it to external customers. For this reason, we developed an in-house basic version of the LSF® scheduler and compared our model performance on job pending time reduction with status quo situation on a suite of real jobs.

11.1.1 Basic LSF® Scheduler Simulator

We imitated similar scheduling behavior of the real LSF® scheduler with less number of resources in our basic simulator. This simulator composed of two servers parameterized to be of different sizes (128GB and 256 GB) and three queues with different priorities: high, medium and low. The job scheduling method is FIFO in each queue with highest probability of scheduling a job from high priority queue versus medium priority queue and so on. We filled up the queues in our simulator using real jobs submitted for the "Physical Design" phase on our major grid. Table 3 compares the effect of using our memory recommender engine on job pending time versus what is currently requested by user using our basic LSF simulator. Note that these simulations are done using a mock-up version of the LSF® scheduler on much limited number of servers and considers memory as the only compute resource needed for job submission. The experiment clearly demonstrates the high potential for improvement in job pending time, however the actual improvement percentages should be looked at with caution and shall not be considered as standard.

Table 3. Average Pending Time Reduction for Physical Design Jobs on Grid 1

Server Size	Requested by User	LR(3)+UPI$_{95}$	Improvement (%)
128 GB	43 min	28 min	34%
256 GB	47 min	27 min	27%

12. BENEFITS OF IMPROVING MEMORY UTILIZATION

There are multiple benefits to Qualcomm if compute memory is optimally resourced. The first benefit is the reduced annual cost of compute hosts with high quantities of installed memory. Whenever possible, it is desirable to purchase cheaper computers with 16GB memory and four processors, rather than computers with high level of memory resources. It is estimated that the annual savings could be in the millions of dollars. This predictive model could also be leveraged to minimize compute costs for jobs that could be run in a cloud environment.

As demonstrated by simulations discussed in the preceding section, pending time can potentially be reduced 27% to 34% for a subset of the jobs. Thus, another benefit of improved memory allocation utilization is the potential for the reduced wait time of jobs. Reducing pending time will greatly improve the throughput of compute jobs, thus enabling more chip design and test jobs to be executed resulting in a better quality for the chip. This reduces the wait time for an engineer to get results back and improves the engineer's productivity. Finally, this new system can contribute by helping to bring the chip to market faster, which is an essential advantage in today's competitive markets. Qualcomm places a high value on all these contributions.

13. LESSONS LEARNED

This Compute Memory Recommender project showed that the compute memory needs could be predicted with an acceptable level of accuracy and be leveraged to improve throughput efficiency. This is in addition to the benefits achieved by reduction in computer resource cost. We confirmed that cleverly constructed simple predictive models, like linear regression, could provide the solid business value, and is easier to deploy than more sophisticated machine learning and statistical models like neural nets, Kalman filters, and ARIMA. The main advantage of using a linear regression model is that it is simple, parsimonious, a good predictor, easy to understand, and easy to implement.

Another lesson learned, (or rather re-enforced) is to take novel approaches to defining the problem, rather than tackling a challenge with an expected "chip-design life cycle" approach. We found we could use a recommender model approach that recognized the unique tasks and cadences of each engineer's work, along with their immediate past tasks to predict their next computer job's needs. Thinking more like a "social media" modeler lead us to construct many simple individual models. An alternate "life cycle" approach which was not feasible with our available data would have been to create a model that predicted a standard memory need for a chip project in the N^{th} week of the design cycle for a given task. With the high level of accuracy and flexibility which our recommender model provides, a "life cycle" model is not necessary.

14. SUMMARY

The implementation plan for our Compute Memory Recommender system is to deploy it across all chip design phases in 2016. At the time of writing this paper, we demonstrated the ability to create predictive models with R-squared of over 90% for the "Physical Design" phase. The tasks done in "Physical Design" phase are some of the most compute intensive jobs in semiconductor development. We believe we will see similar good results with other design phases.

In the course of the analysis and model development, we found we needed to take a "recommender" model approach that predicts the needs of compute memory for jobs which run the next day for an engineer doing a specific task for a given project. We tested many statistical models and machine learning techniques which produced similar results. In order to provide a simple, understandable model that could be readily implemented and updated, we chose a linear regression model with lagged variables capturing prior usage by the individual engineer. An upper 95% statistical prediction interval was added to our model recommendation to ensure jobs do not under-request memory more than 5% of the time based on the business requirements.

The implementation of the recommender model used the Splunk® data repository and software. Predictive models were written in Python code, and deployed in conjunction with the Splunk® application. This system is run daily to train and build unique prediction models for each user by project, EDA software, number of processors, and compute grid. The Splunk® alerting system contacts the engineers with the greatest mismatch of requested and actual compute memory usage on a daily basis.

Our long term goal is to incorporate our compute memory recommendations from the model into job submission scripts for all design phases. This would make the job resource requests at runtime efficient. Having these capabilities will not only save Qualcomm on the cost of computer hardware, but improve job turn-around time thus saving engineering time. Potentially, the greatest contribution of improving the utilization of all these resources, is expediting the semiconductor's time to market.

15. REFERENCES

[1] Lavagno, L., Martin, G., Scheffer, L., 2006. Electronic Design Automation for Integrated Circuits Handbook, Taylor & Francis.

[2] Goering, Richard, March 8, 1999. Load Sharing Brings Kudos, EE Times Online.

[3] Haykin, S., 2007. Neural Networks, a Comprehensive Foundation, Second edition, Prentice Hall of India.

[4] Shumway, R., Stpffer, D., 2011. Time Series Analysis and Its Applications, Third edition, Springer.

[5] Simon, D., 2006. Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches. Wiley-Interscience.

[6] Hastie T., Tibshirani R., and Friedman J.H., 2009. The Elements of Statistical Learning, 2nd edition. Springer.

[7] Kass, G. V., 1980. An Exploratory Technique for Investigating Large Quantities of Categorical Data, Applied Statistics, Vol. 29, No. 2, pp. 119–127.

[8] Sheather S., 2009. A Modern Approach to Regression with R, Springer.

[9] Carasso, D. 2012. Exploring Splunk: Search Processing Language (SPL) Primer and Cookbook. New York, NY: CITO Research.

[10] Zhou, S., Zheng, X., Wang, J., and Delisle, P. 1994. Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems. Software-Practice and Experience, Vol 23, No. 2, pp. 1305-1336.

[11] Zadrozny, P., Kodali, R., 2013. Big Data Analytics using Splunk. Apress.

[12] Foster, I., Zhao, Y., Raicu, I., Lu S., 2008. Cloud Computing and Grid Computing 360-Degree Compared, Grid Computing Environments Workshop, GCE'08, pp. 1-10, doi: 10.1109/GCE.2008.4738445.

[13] Grewal, M.S., Andrews A.P., 2014. Kalman Filtering Theory and Practice Using MATLAB, 4th Edition. John Wiley and Sons.

[14] Friedman, J. H. 1991. Multivariate Adaptive Regression Splines. The Annals of Statistics Vol 19, No 1. doi:10.1214/aos/1176347963.

[15] Widrow B., Rumelhard D.E., and Lehr M.A., Neural Networks: Applications in Industry, Business and Science, 1994, Communications of the ACM, Vol. 37, pp. 93-105. doi: 10.1145/175247.175257.

[16] Loh W.Y., 2011. Classification and regression trees. WIREs Data Mining and Knowledge Discovery, Vol 1, pp 14–23