Characterizing and Optimizing EDA Flows for the Cloud

Abdelrahman Hosny
Department of Computer Science
Brown University
abdelrahman_hosny@brown.edu

Sherief Reda
School of Engineering
Brown University
sherief_reda@brown.edu

Abstract— Cloud computing accelerates design space exploration in logic synthesis, and parameter tuning in physical design. However, deploying EDA jobs on the cloud requires EDA teams to deeply understand the characteristics of their jobs in cloud environments. Unfortunately, there has been little to no public information on these characteristics. Thus, in this paper, we formulate the problem of migrating EDA jobs to the cloud. First, we characterize the performance of four main EDA applications, namely: synthesis, placement, routing and static timing analysis. We show that different EDA jobs require different machine configurations. Second, using observations from our characterization, we propose a novel model based on Graph Convolutional Networks to predict the total runtime of a given application on different machine configurations. Our model achieves a prediction accuracy of 87%. Third, we develop a new formulation for optimizing cloud deployments in order to reduce deployment costs while meeting deadline constraints. We present a pseudo-polynomial optimal solution using a multi-choice knapsack mapping that reduces costs by 35.29%.

I. Introduction

EDA (Electronic Design Automation) tools expose hundred of parameters to tune for front-end and back-end design. Therefore, design space exploration and efficient physical implementation have been increasingly challenging and require a massive amount of compute to achieve acceptable Quality of Results (QoR). In the recent years, there has been a growing trend among EDA teams to utilize elastic compute environments (i.e. cloud) to gain near-instant access to compute resources [1]. Migrating EDA jobs to the cloud has helped teams meet the demands of their tapeout schedule, hence reducing the time to market [2]. For example, horizontal scaling by launching more compute servers allows EDA teams to complete a highly-parallelizable compute job such as simulation in less time. In addition, EDA teams have the flexibility to choose the configuration of hardware that meets their needs for the exact pending job and only for the time needed to complete it.

However, migrating EDA jobs to the cloud is not a straightforward path, especially for teams with little or no experience managing elastic resources. For example, design teams need to choose the right machine configuration that achieves the best performance for their job. While simulation and verification are known to be embarrassingly parallel (i.e. directly benefiting from the scale of the cloud), the compute requirements for the synthesis and physical design stages are not well-studied, especially in multi-tenancy environments.

This work is partially supported by NSF grant 1814920.

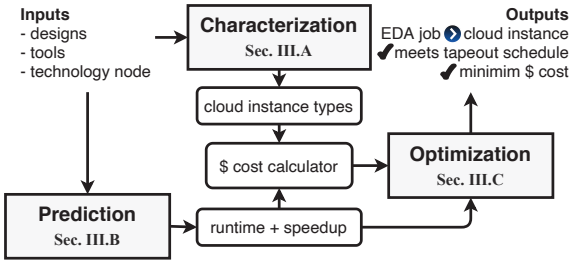


Fig. 1: Workflow of optimizing EDA cloud deployments

Furthermore, reducing deployment costs while meeting the tapeout schedule is a challenge, especially in teams with limited budget.

Our main contributions are:

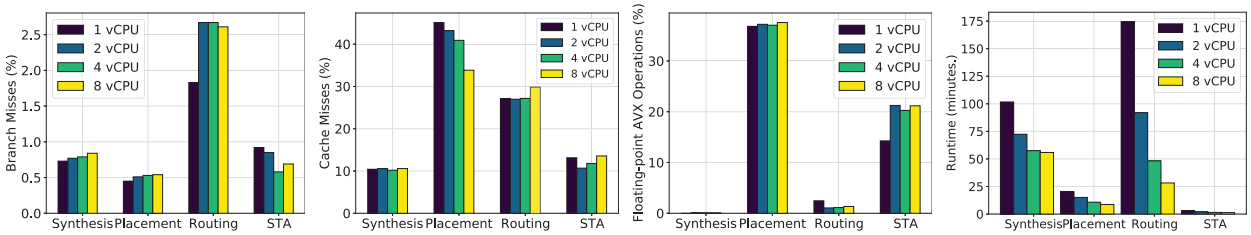
1) We characterize the performance of four EDA key applications (synthesis, placement, routing, and static timing analysis) under different machine configurations. Using our observations, we present recommendations for the configurations of cloud instances to provision for each application.
2) We propose a novel model based on Graph Convolutional Networks (GCNs) that predicts the total runtime a given job would take using certain machine configurations. Our model achieves a high prediction accuracy of 87%.
3) With recommended machine configurations and runtime predictions for each application, we reduce cloud deployment costs subject to deadline constraints by mapping the problem to the classical multi-choice knapsack problem (NP-hard). Our implementation recommends optimal machine configurations that minimizes the total cloud deployment cost by an average of 35.29%.

Next, we give a brief background in Section II. In Section III, we formulate the problem and discuss our proposed framework. In Section IV, we present our experimental results.

II. Background

Cloud computing refers to the elastic compute resources that can be provisioned, scaled up or shutdown on demand. Cloud providers virtualize their physical infrastructure to share processing time, memory, storage and network bandwidth among many users (known as tenants). In order to achieve this virtualization, cloud vendors use a specialized software called the *Hypervisor*. The Hypervisor isolates each tenant's resources in a Virtual Machine (VM) that is accessible only by its owner. In a standard cloud offering, VMs are sold in units of: (i) vCPU: a virtual CPU is seen as a single CPU thread, (ii) Memory: a fixed number of memory pages is solely reserved for the use of a VM and is expressed as the total memory size

978-3-9819263-5-4/DATE21/©2021 EDAA



(a) Branch Misses (b) Cache Misses (c) Floating-point Operations (d) Total Runtime

Fig. 2: Performance characterization of four representative EDA jobs

reserved, and (iii) Storage: the size and type of the underlying storage device partition mounted on the VM.

III. Optimizing EDA Cloud Deployments

Problem Definition. A fundamental question that faces EDA teams when migrating their EDA jobs to the cloud is: what configurations of VMs should be provisioned for each job? And how can the job completion time be reduced while minimizing the cost?

In order to answer these questions, Figure 1 draws our workflow that we propose in this paper. Specifically, we introduce the following problems:

Problem 1. What is the right VM configuration for a given EDA job? To address this problem, we characterize four main EDA applications, namely: synthesis, placement, routing and static timing analysis. We focus on characteristics that are intrinsic to the EDA job which affect the completion time.

Problem 2: Given a design (in RTL or Netlist), predict the runtime for a given job (e.g. synthesis or routing) when using 1, 2, 4 and 8 vCPUs. Our proposed prediction model, learns internal graph features of the design that affect the total runtime of a given job on different machine sizes.

Problem 3: Given the runtime for each job under 1, 2, 4 and 8 vCPUs, as well as a deadline constraint, select a machine size for each job such that the deadline is met and the total cost is minimized. We address this problem using a mapping to the multi-choice knapsack problem and implement an optimal solution using dynamic programming.

A. EDA Flow Characterization

To address Problem 1, we characterized the four jobs using a major commercial EDA flow and a SPARC core design from OpenPiton design benchmark [3] using a 14nm technology node. We then collected the execution data from the system's hardware performance counters for further analysis. In order to simulate a multi-tenancy environment, we used Linux Control Groups on a machine with a 14-core Intel Xeon E5-2680 processor running at 3.3GHz, and 128GB DDR4 memory. We used the Linux perf utility to instrument the hardware performance counters.

Branch Prediction. Figure 2-a summarizes our findings from the characterization experiments. First, we observe that routing has a higher percentage of branch misses. We attribute this value to the nature of the routing algorithms, where there can be few trials before a net is successfully routed with no design rule violations. In particular, graph search algorithms in the routing step encompass a large portion of conditional statements that cannot be avoided. Rip-up and reroute techniques also contribute to halting the continuous execution of the routing algorithms.

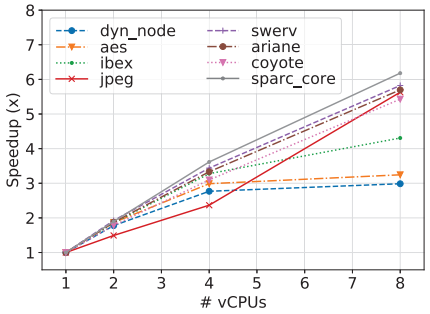


Fig. 3: Routing speedup for different designs. *dyn_node* is the smallest and *sparc_core* is the largest (#instances).

Memory Access Patterns. In Figure 2-b, we observe that placement and routing have significantly higher cache misses than synthesis and STA. Placement has a 45.11% cache misses rate when using 1 vCPU and 33.84% when using 8 vCPUs, while routing has 27.15% and 29.84% cache misses rate using 1 and 8 vCPUs respectively. We attribute this higher miss rate to the nature of the analytical component in the placement engine that tries to optimize the wirelength across all the chip instances using convex optimization methods. This needs access to large vectors to calculate the gradients, hence benefiting from the more cache available with more vCPUs.

Floating-point Operations. In Figure 2-c, we observe that the placement job requires more floating-point operations that run on Advanced Vector Extensions (AVX) hardware. This can be attributed to the analytical engine that tries to optimize the wire length across all the chip area using convex optimization methods. This involves calculating gradients which relies on floating-point operations. The STA job comes next in its percentage usage of the AVX hardware. This is consistent with the nature of STA algorithms where calculating slacks involves graph traversal from inputs to outputs, with access to floating-point values in the technology library.

Scalability and Speedup. In Figure 2-d, we observe that the routing job scales well with more #vCPUs. This is consistent with the nature of the routing job, where nets in independent grid cells can be routed in parallel with no conflict, as opposed to synthesis, placement and STA where internal algorithms have inherent dependencies. Further analysis of the routing job, Figure 3 plots the speedups achieved on different designs of different characteristics and sizes. It shows that adding more vCPUs does not eminently scale the routing job in all designs. Smaller designs (such as dynamic_node and aes) have almost equal speedups for 4 and 8 vCPUs. This means that the provisioned vCPUs might not offer the expected benefit from the cloud scale, and that there is an opportunity to achieve the same outcome in nearly the same time with less resources.

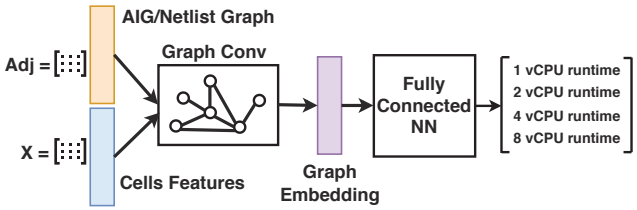


Fig. 4: Our proposed runtime prediction model

Main Takeaways. From the point of view of EDA teams running their EDA applications on the cloud, we summarize our main recommendations:

1) Synthesis and STA jobs perform well on general-purpose VM instances with a balance between computations and memory access. Placement and routing require VM instances with higher memory-to-core ratio, with routing demanding more available L1 and LLC cache.
2) Placement jobs should be run on a compute instance with an underlying processor that supports Advanced Vector Extensions (AVX). STA jobs would also benefit from AVX hardware.
3) On large designs, routing jobs scale well with the number of vCPUs allocated. However, on small designs, speedup is capped at a certain point.

These observations motivate our work in the next section.

B. Runtime Prediction

To address Problem 2, we state that the runtime of chip design tasks depends on a number of factors such as the design itself, the tools used, the technology node, the parameters used to instruct the tools and the VM configuration. Without losing generality, when using the same tools, technology node, default parameters and VM configuration, the runtime of a certain job depends on the complexity of the design itself. Figure 4 shows the architecture of our model. The model takes as input the design in RTL or netlist, and performs an embedding operation using Graph Convolutional Networks [4]. After that, a fully-connected neural layer transforms the embedding into predictions for the runtime under different machine sizes (i.e. #vCPUs). This model is trained for each application separately. Using the predicted runtimes, we can calculate the speedup gains from using 2, 4 or 8 vCPUs as compared to using only 1 vCPU.

Processing Input Design. When building a model to predict synthesis runtime, the input is usually in RTL, which is not a graph. However, synthesis tools map the RTL into an intermediate representation such as And-Inverter Graphs (AIG) before synthesizing and mapping to a technology library. Therefore, our model can operate on the AIG representation of the design. The AIG is a Directed Acyclic Graph (DAG), which means it preserves edge directions for the GCN. On the other hand, when building a prediction model for the placement and routing, the input is expected to be a netlist. In order to operate on the given netlist, we convert cells and I/O pins into graph nodes. In addition, we convert each net into a set of directed edges using the well-known star model, where there is one edge from the driving cell (or the input pin) towards each of the sinks (or the output pins).

Graph Convolutions. In Graph Convolutional Networks (GCNs), the key idea is to generate node embeddings based on local neighborhoods. The first layer's embedding of a node represents its input feature vector, x_i. Nodes aggregate information from their neighbors in each convolutional layer. This aggregation is followed by an activation function, such as *ReLU*, and a pooling operation, such as *sum*-pooling. With that in-place, every layer is written as a non-linear function:

$$\mathcal{H}^{(l+1)} = f(\mathcal{H}^{(l)}, \mathcal{A}), \tag{1}$$

where $\mathcal{H}^{(l)}$ represents the activation at layer l, and $\mathcal{H}^{(0)}$ is the input feature matrix, $\mathcal{X}$. $\mathcal{A}$ is the adjacency matrix. Looking at the embedding of each node, we can elaborate on Equation 1 as follows:

$$h_v^k = \sigma\left(W_k \sum\nolimits_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1}\right) \quad \forall k \in \{1, ..., K\}$$

where, h_v^k is a vector that represents k^{th}-layer embedding of node v. $N(v)$ represents the neighbors of node v. W_k and B_k are the trainable matrices which are shared with all nodes of the graph, and σ is the activation function. After K-layers of neighborhood aggregation, we get output embeddings for each node that can be fed into a loss function. We can then run stochastic gradient descent to learn W_k and B_k.

Model Design. We used 2 GCN layers with 256 and 128 hidden units each, followed by 1 fully connected layer with 128 units. The model is trained for 200 epochs using Mean Square Error (MSE) as a loss function and Adam as the optimizer (lr=1e-4). The loss function calculates the combined prediction error for all four runtimes (i.e. 1, 2, 4 and 8 vCPUs).

C. Cloud Deployment Optimization

Given the runtime estimates, we now address Problem 3. Our proposed solution maps the problem to the multi-choice knapsack problem (MCKP) [5]. Using our predictions calculated in the previous section, each job can be run on a different machine configuration (i.e. #vCPUs), where each configuration completes the job in t time and costs p in total.

Formulation. Let $z_l(C)$ be an optimal solution defined on l applications and with total runtime constraint C:

$$z_l(C) := \max \sum_{i=1}^{l} \sum_{j=1}^{N_i} s_{ij} \frac{1}{p_{ij}} \tag{2}$$

such that,

$$\sum_{i=1}^{l} \sum_{j=1}^{N_i} s_{ij} t_{ij} \leq C,$$
$$\sum\nolimits_{j \in N_i} s_{ij} = 1, i = 1, ..., l,$$
$$s_{ij} \in \{0, 1\}, i = 1, ..., l, j \in N_i$$

where $s_{ij} \in \{0, 1\}$ denotes whether we select VM configuration j for stage i or not, and N_i denotes the number of configurations in a given stage. t_{ij} denotes the runtime of stage i when using j's configuration, which is obtained from the runtime predictions. Similarly, p_{ij} denotes the cost of running stage i when using j's configuration, which is obtained from the pricing table of the selected cloud vendor. We assume that $z_l(C) := -\infty$ if no solution exists (i.e. the total runtime is not sufficient to complete all the stages using the fastest machine configuration).

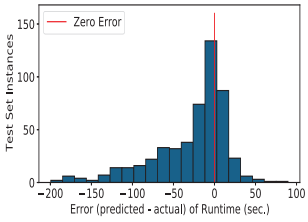


Fig. 5: Runtime prediction errors. Avg. Error: 13%.

Task		Synthesis (general-purpose VM)				Placement (memory-optimized VM)				Routing (memory-optimized VM)				STA (general-purpose VM)				Total Runtime	Min Cost ($)
vCPUs		1	2	4	8	1	2	4	8	1	2	4	8	1	2	4	8		
Runtime (sec.)		6100	4342	3449	3352	1206	905	644	519	10461	5514	2894	1692	183	119	90	82		
Cost ($)		0.16	0.15	0.19	0.37	0.04	0.04	0.05	0.08	0.32	0.25	0.21	0.25	0.02	0.01	0.02	0.05		
Total Runtime Constraint	10000		x			x						x			x			8561	0.41
	6000			x				x					x		x			5904	0.50
	5645				x				x				x				x	5645	0.75
	5000																	NA	NA

TABLE I: Minimizing total cloud deployment cost subject to a time constraint. The mark (x) denotes the recommended machine configuration. NA denotes Not Achievable.

To solve (2), we implemented a pseudopolynomial solution through dynamic programming utilizing Dudzinski and Walukiewicz approach [6]:

$$z_l(C) = \max \begin{cases} z_{l-1}(C - t_{l1}) + 1/p_{l1} & \text{if } 0 \le C - t_{l1}, \\ z_{l-1}(C - t_{l2}) + 1/p_{l2} & \text{if } 0 \le C - t_{l2}, \\ : & \\ z_{l-1}(C - t_{l_{n_l}}) + 1/p_{l_{n_l}} & \text{if } 0 \le C - t_{l_{n_l}} \end{cases}$$

This implementation provides an optimal solution provided that the runtime values are rounded to the nearest integer (second). This is an assumption that we can safely make in our case since cloud machines are billed per second (no fractions).

IV. Experimental Results

We demonstrate our predictions using GF 14nm technology node and a major commercial EDA flow.

Dataset. We use 18 representative benchmarks of different sizes and structures from EPFL benchmark suite [7] and OpenCores [8]. We synthesize each benchmark applying different logic optimizations to generate different netlists. The motivation is to challenge the GCN with netlists that have different physical structures, but perform the same logic function. We have a total of 330 unique netlists, with 2,640 data points (runtimes) for different machine configurations. These designs range from a few hundred instances to 200k instances. We divide the dataset into training and test groups with 80% and 20% respectively, where netlists of the test set belong to unseen designs in the training set.

Prediction Accuracy. Due to lack of space, we show a histogram of model prediction errors for placement and routing in Figure 5. Runtime predictions given a netlist (placement, routing, STA) achieves an average error of 13%. On AIGs (synthesis), the runtime prediction has an average error of 5%.

Optimization Results. Referring to Figure 1, our optimization module takes as input the predicted runtime for a given EDA job on certain machine configuration (Section III-B), and the cost of running the job on a machine type recommended for that job (Section III-A). In order to calculate the cost, we obtained the pricing table for the machine configurations from AWS at the time of this writeup, and calculated the total cost (in USD) for each EDA job (cost = runtime in hours $\times$ cost per hour). To demonstrate our optimization, we applied different runtime constraints on predictions the of the *sparc_core* design as shown in Table I. Our algorithm outputs the recommended machine configuration for each job that minimizes the total cost subject to the given total runtime constraint. As we tighten the time constraint, we observe that the algorithm chooses higher machine configurations in some tasks (but not all). A very tight time constraint cannot be met and no solution is presented. Figure 6 shows the cost savings that we get from running our optimization as compared to over-provisioning (using 8 vCPUs in all jobs) or under-provisioning (using 1 vCPU in all jobs) machines. It offers an average of 35.29% cost saving with minimal overhead to the best runtime.

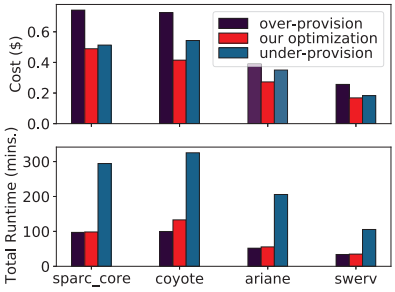


Fig. 6: Cost savings from running our muli-choice knapsack optimization algorithm. Over-provisioning runs all stages on 8 vCPUs. Under-provisioning runs all stages on 1 vCPUs.

V. Conclusions

We present an end-to-end workflow for a cost-efficient deployment of EDA workloads on the cloud. Our method saves 35.29% of the costs while meeting scheduled deadlines. The code is open-source under a permissive license (BSD-3) and is available publicly on GitHub[1].

References

[1] V. Kamath, R. Giri, and R. Muralidhar, "Experiences with a private enterprise cloud: Providing fault tolerance and high availability for interactive eda applications," in *2013 IEEE Sixth International Conference on Cloud Computing*, 2013, pp. 770–777.

[2] N. Sehgal, J. M. Acken, and S. Sohoni, "Is the eda industry ready for cloud computing?" *IETE Technical Review*, vol. 33, no. 4, pp. 345–356, 2016.

[3] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang *et al.*, "Openpiton: An open source manycore research framework," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 217–232, 2016.

[4] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[5] H. Kellerer, U. Pferschy, and D. Pisinger, *The Multiple-Choice Knapsack Problem*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 317–347. [Online]. Available: https://doi.org/10.1007/978-3-540-24777-7_11

[6] K. Dudziński and S. Walukiewicz, "Exact methods for the knapsack problem and its generalizations," *European Journal of Operational Research*, vol. 28, no. 1, pp. 3–21, 1987.

[7] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The epfl combinational benchmark suite," in *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, no. CONF, 2015.

[8] "Opencores." [Online]. Available: https://opencores.org/

[1] https://github.com/scale-lab/EDAonCloud