Exploration of Compiler Optimization Techniques

Sumanth Manduru[1] and Raviteja Gowtham[2]

Abstract— **Researchers and Programmers had been striving to apply machine learning based approaches to resolve many code optimization problems in compiler design since 20th century. Standard and Traditional Optimization techniques had failed in giving the best solutions since the output of their overall performance are in average value. So, by using Machine Learning techniques, We can observe the raise in standard of the results and it can bring an improvement of up to 60 percent compared with standard optimization. Additionally, It makes easier to handle the optimization problems namely Optimization Selection and Phase Ordering. The former one is about the selection of the best code optimizers and the latter one is about the phase-ordering of code optimizations. Indeed, with these techniques, we can solve many challenges like portability, traversing, reducing usage of huge space and adaptability. In this paper, We will discuss the strategies of code optimizations techniques. This paper also reviews the approaches done in compiler optimization i.e, choosing the best optimizations by both (DSE) Design Space Exploration techniques and machine learning techniques in order to tackle the challenges of selecting along with the phase-ordering of compiler optimizations.**

I. INTRODUCTION

A compiler is a computer application which converts a set of instructions scripted in (source-language) one computer language into (target-language) another computer language. Usage of compilers is not recent, it has been used for producing assembly language from high-level programming languages since 1950s. Developers have designed the compilers in such a way that it converts every part of the instruction in program to provide the optimized kind of it. Simultaneously, optimizing high-level programming languages (source-code) by hand is a long-winded task. Compiler has mainly three layers namely Front-End, Intermediate-Representation and Back-End. Optimization techniques could be applied at any one of these layers. Compiler optimizations plays an vital role in transforming the source code into its optimized form. But, applying optimization at Intermediate-Representation Layer is going to play an vital part in the performance metrics. Moreover, these metrics may be Power Consumption, Execution Time, Code Size, Resource-Allocation, Scheduling, Code-generation etc. In past, Developers have applied optimization techniques at Back-End phase in which code-generation, scheduling and resource-allocation are benefited. Currently, research is going on for introducing optimizations at Front-End and Intermediate-Representation. Several Performance metrics could be benefited by enabling the parameters of compiler optimization like register allocation, loop-unrolling etc.

*This work has done under guidance of Prof Viswanath P

[1]Prof Viswanath is with Faculty of Computer Science Engineering, Mathematics and Computer Science, Indian Institute of Information and Techology, Sri City, A.P viswanth.p at ieee.org

The biggest challenge lies in selecting the appropriate collection of compiler optimizations. The reason is that they are dependent of application, architecture and programming language. In addition to this, there is no assurance that the optimized code performs well compared to source code. To our knowledge, there are some optimizations which can reduce the performance of code. Aggressive optimizations are one kind of them. There are some complex problems like awareness in behavior of optimizations, understanding the result of source code and interactions between optimizations. To understand all of these, one need to consider different optimizations during several phases of compilation. These factors guided towards phase-ordering problem in which one must know ordering the different optimizations that are going to be used.

Still, we did not solve some of the open problems with compiler optimization like which optimization technique should use, what are the values to choose for parameters and what is the sequence order of optimization techniques need to be chosen so that accuracy of the output is higher. In above things, both first and second problem are the challenges for selection of compiler optimizations and the last one holds for phase ordering of compiler optimizations. Even for today, research is going-on for phase ordering in the compiler field. Since developers have the lack of capability to solve the phase ordering challenge completely, they are still working on selecting the group of optimizations. There is a process done manually for selection of right optimizations for kernels and with the insight existing in interaction among current and the preceding compiler optimizations made to construct the sequence of optimizations. To construct a right sequence is not easy as number of compiler optimizations are being increased daily in compiler frameworks. We have some examples like GCC contains 200 compiler options, LLVM-clang contains 150 and LLVM-opt too have 150 compiler passes, considering each pass to one option. In addition to that, these are not applied at same compilation phase. Some passes are used to analyze data access and some are used to explore loop nests. Unfortunately, Compiler researchers and developers should depend on software agencies because software application is profitable for certain optimizations. So that, developers need to work on only that set of optimizations in order to acquire benefit for the software application. Some of the standard optimizations like -Og, -O1, -O2, -O3, -Os etc. are named as they have certain sequence of compiler optimizations which achieve good performance on benchmark applications.

In coming sections, DSE and ML Techniques used in compiler optimizations are discussed.

II. PROBLEMS OF COMPILER OPTIMIZATION

Before, getting into problem statement, one need to know the terminology used in the optimization theory.
For example: "solution space, search space or feasible set" all of these implies the set consisting of points which satisfy conditions such as constraints like domain of the problem, potentiality in equalities, inequalities etc.
We have been facing two problems which are from two different areas. One is selection of optimization problem and the other is phase ordering problem.

1) Selection of Compiler optimizations: Optimization sequence consists of many compiler optimizations passes. Researchers had already proved "In a sequence, performance of a code can be altered by interactions and interdependencies among enabling or disabling optimization even if phase order is ignored".

Optimizations Spaces: If every compiler optimization is notated as O_i and each can be labeled as 0 or 1. Let O be an boolean vector which specifies an Optimization sequence. The size of O is n x 1. 0 refers to disabled and 1 refers to enabled.

$$|\Omega_{Selection}| = \{0,1\}^n$$

Let us redefine the above equation with discretized values. To our simplicity, let's consider m be the total choices present in the compiler optimization. As above equation is defined for continuous values, we cannot find discrete values. The reason for predefined version is that optimizations like tiling and loop unrolling provides multiple parameters of tuning and some optimizations may even need more than one factor. So, to satisfy all those needs, we need to go for predefined version. The predefined version is as follows:

$$|\Omega_{Selection}| = \{0,1....m\}^n$$

2) Phase-Ordering Problem: The most default problem is still we don't find any ideal order for phase ordering. If each phase transformation benefits the other, then there may be interdependent between the pair of optimization phases.

If n be the number of optimization that taken under analysis, then

$$|\Omega_{Phases}| = n!$$

If some of the phases are repeated, l be the desired maximum length for the optimization sequence and n be the optimizations under analysis, then

$$|\Omega_{Phases-Repetition-Variable-Length}| = \sum_{i=0}^{l} n^i$$

III. AUTOTUNING OVERVIEW & TECHNIQUES

Autotuning is a methodology in which a model is inferred by user for objectives with a minimal interactions. The methodology can be algorithm or search heuristic etc. In Compilers Field, Autotuning defines an optimization policy in the means of Design of experiment in which tuning parameter named space is registered. That space helps in creating the possible ways of a given program.
ATOS - Auto Tuning Optimization System.

- Capability to find best compiler configuration for the objective given, automatically.
- It should preserve source codes and files, build scripts too.

Iterative Exploration: The exploration of all possible versions of a input program. It can be achieved by analyzing the all possible versions of program. It can also be achieved through search exploration heuristic in which we sample the subset of the space, but the examining the entire search space may not be possible. This causes imbalance between the computational complexity(time) and the number of parameters used to build the model.
There must be desired outcome for autotuning framework. The desired outcome is expressed in terms of performance. The machine learning algorithm should be trained by taking the information which is extracted from an application and also some objectives. So that we can emit a predictive model. Usually, information from application could be high in terms of size. So we need to apply dimension reduction to the features which have been extracted, so that we can observe reduce in quantity of information. One need to extract consistent features from different applications. This can possible by Dimension Reduction. So, It is helpful and necessary. Researchers certified the principle component analysis as one of the best and popular dimension reduction technique.
After constructing the model, one can feed the test set of feature extraction and dimension reduction which in turns produce a feature matrix of size n x 1. This feature vector can be fed into the predictive model. The output which has been from model can be compared with known result to evaluate the accuracy and to predict error measurement. Most of the modern compilers are capable of working on several platforms through implementation of optimizations. But those optimizations are not applicable for every target program. There have been many articles and journals proposed the implementation of tools for Automatic Compiler Tuning. Many papers help in identifying the under performing compiler optimizations.
In one word, Autotuning refers to restructuring of the existing compiler. One should identify feature space that have the capability of capturing all the possible implementations. One need to prune the space available with the knowledge of compiler domain and features from architectures. One could give access to programmers for portability where source to source transformations occurs.

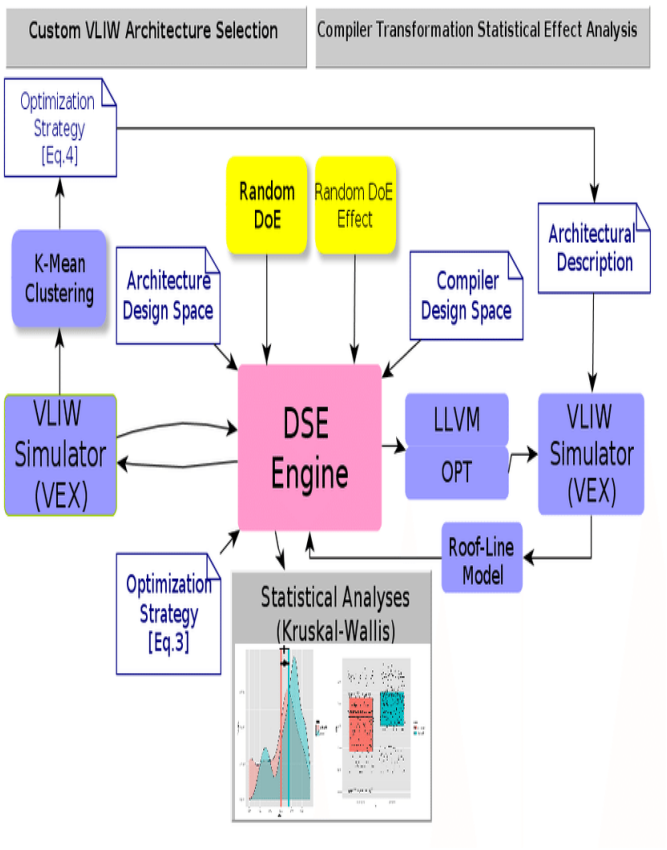

Fig. 1. Design Space Exploration Approach & Methodology

Option	GSM	AES	ADPCM	JPEG	Blowfish
Constprop	-	-	-	-	-
Dce	-	-	-	-	-
Inline	√	-	√	√	-
Instcombine	√	-	√	√	√
Licm	√	√	√	√	√
LoopReduce	√	√	√	√	√
LoopRotate	√	√	√	√	-
LoopUnroll	-	-	-	-	-
LoopUnswitch	-	-	-	-	-
Mem2reg	√	√	√	√	√
Memcpyopt	-	-	-	-	-
Reassociate	√	-	-	-	-
Scalarrepl	√	-	-	-	√
Sccp	-	-	-	-	-
Simplyfycfg	-	-	-	-	-

Table 1: Kruskal-Wallis Test on Performance

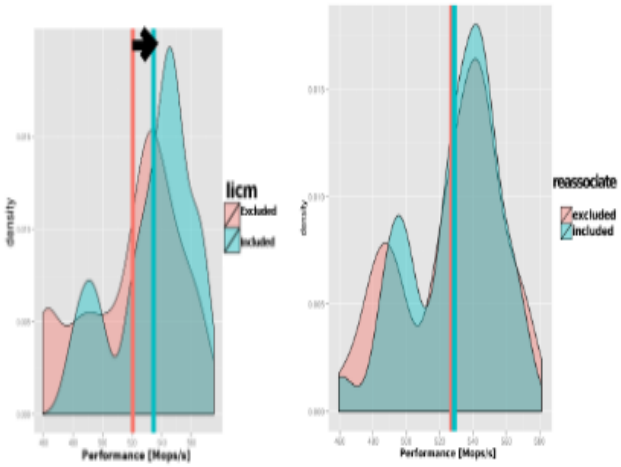

Fig. 2. Kruskal Wallis Test Table along with Graph Models

A. *Design Space Exploration*

We had already learned that Autotuning is nothing but generating a search space in which computing the all possible implementations. Among many computation implementations, Code Variant indicates a representation of the implementation of unique one. A set of elements which control the variant execution and generation of a code. Those set is discrete in nature. They are even called as Parameters. One need to measure the execution time and compare for evaluating the best implementation.
It is a method of activity where one can explore the parameters of space before the actual design. It is a strategy where pruning occurs. It is the model which can utilize the design space effectively. We can think it off in other way like,

- Platform should perform at atmost level.
- Consumption of power and non functional parameters should be minimized.

So, These factors make the problem as a Multi Objective as many objectives are playing major role in optimization problem. This method is concentrating on exploring the parameters of compiler so that it can automatically learn and analyze the design space as well as architecture of compiler. So, architecture properties also play an important role in the methodology as DSE needs to search parallel for candidate points of architecture which helps in reaching the properties of the target application.

We evaluated the DSE Method in an architecture named (VLIM) Very-Long-Instruction-Word. There exists a tool chain in architecture called Multi-Objective System Tuner (MOST) which is a wrapper. This tool helps in tackling the problems when DSE algorithm applied and thereby proposes an method which acts automatically. One can find two open source compilers such as VLIW-Example and LLVM. VLIW-Example is even called as VEX. The tool-chain makes the designer to search followed optimization, later on analyze the compiler options. There is a model named Roof-Line performance model which plays major role in configuring the architecture and also identifying the compiler passes effect and makes a relation between th e performance of a processor and off-chip memory traffic. Designer focuses more on analysis by adopting some methodology through cross-architecture or/and cross-application manner. Thus he can make delivery of insights present in the application. Thereby, he can explore the architecture as well as strategies of compiler optimization.

In Figure1, one can see the DSE Engine methodology and it's techniques. The methodology starts from left-side, of the diagram shown above, by deducing the design space of

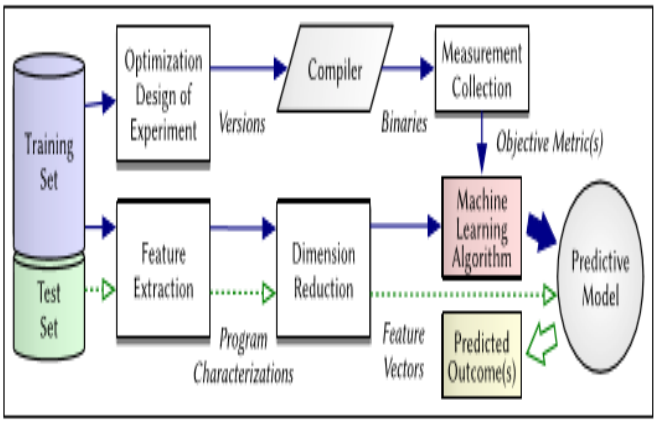

Fig. 3. Machine Learning Methodology

architecture. So that it acquires the properties of architecture. It feed these properties to the framework which is on right side of the diagram. In order to estimate the relation between the utilization of compiler options and the performance parameters observed, we will apply statistical analyses at the end.

We have analyzed the table which represents the estimation of utilizing specific compiler options on selected applications with the Kruskal-Wallis test. We already observed that not all compiler options which have been utilized, plays a part in regulating the performance parameters on selected application.

The graph in figure2 shows the possible distributions obtained through the DSE Methodology. These graphs exhibits the out-come of exploitation of various compiler options.

B. Machine Learning Approach

Iterative Exploration failed in bringing the optimal as well as efficient results since it is highly dependent of time along with number of iterations.

Before getting into methodology, Let's discuss the over view of the figure 3. The figure above shown is a example of Machine Learning Approach of Autotuning Framework. The top phase is training phase as the data which is used for training, flows along the way of various components. The bottom phase is testing phase as data which is used for testing, is passed through the trained model so that we can estimate the result for the selected application. The test set contains many applications.

Machine Learning explore the algorithms which help in making predictions by learning specific attributes from provided data. A Machine can explore the data by three ways

- Supervised Learning.
- Unsupervised Learning and
- Reinforcement Learning.

In coming sections, we try to explore each learning. We even try to understand each method and model in it's respective learning.

1) Supervised Learning: Supervised Learning is one of the most popular technique used in wide range. It predicts the learning function from the training data which is labeled. This functions helps in predicting the testing data which are unseen. Under Supervised Learning we would like to explore the techniques like SVMs and Linear Models, Random Forests and Decision Trees, Graph Kernels and Finally, Bayesian Networks.

SVMs and Linear Models: Linear Models are being used widely by researchers in solving many applications of Machine Learning. It is included in the list of popular techniques of Supervised Learning. There are many stable models in Linear package like regression, threshold and nearest neighbor. SVMs are also special types of supervised learning. They can be used for regression as well as classification. We can classify non-linear class of problems by kernel trick. It is a trick in which dimensions of higher class is reduced to lower class. This trick helps in constructing hyperplanes and picks the best hyperplane in classifying.

Random Forests and Decision Trees: Basically, Decision tree is a representation of feature space. Random Forests are the methods in which many learning methods are used to predict for regression as well as classification. Moreover, they start with construction of decision trees at training phase and predict the output class. They are even called as Random decision trees. These helps decision trees to prevent overfitting of training data. In past, Researchers used decision trees in order to explore the performance of loop unrolling.

Graph Kernels: Graph kernels are getting popular since the usage of machine learning applications starting from semi-supervised learning and till classification and clustering. They use a different representation of data in which cost function is preserving the data properties. Representing the data as feature vectors is much difficult compared to usage of kernel functions. So that is the reason why researchers are attracted to kernel functions.

Bayesian Networks: Bayesian Networks are the strongest classifiers because of their capability in optimizing the sequences of compiler optimization. A direct acyclic graph in which every node indicates variables and every edge is interdependent of another. Bayesian Network is same as above mentioned DAG. One need to compute probabilities for the variables observed under nodes. Those probabilities can be feed to the model as input which is called as evidence. The relationship lies between the optimizations which are applied is the major factor that the probabilities are depending on. One can say program features are also playing vital role in probability distributions. Researchers have proposed the Bayesian Network in order to select the good optimization technique for an embedded processor. This approach has features like static, dynamic and hybrid which are going to be the evidence for the trained network. The major change is that instead of using all sequence for the purpose of training, we can select the top 15% speedup compiler sequences. By this, BN is trained by good sequences which is quite good for given application. One can measure the speed with the the GCCs standard optimization level-O3.

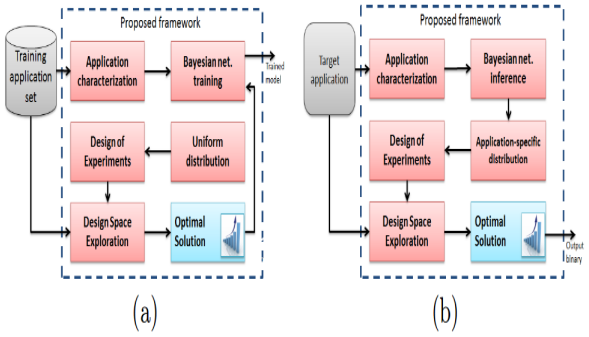

Fig. 4. Training Phase & Testing Phase

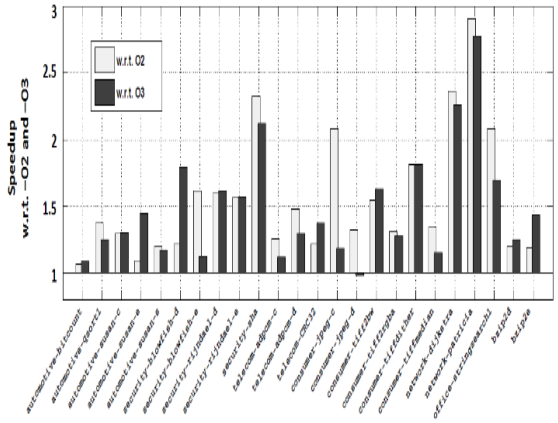

Fig. 5. Improvment in performance of BN w.r.t -O2 and -O3

In Figure 4, One can see the block diagram of training phase & testing phase of the approach.

In Figure 5, we are presenting the improvement in performance of Bayesian Networks w.r.t -O2 and -O3 on cBench suite. An average of 56% and 47% speedup in performance respectively -O2 and -O3.

It was confirmed that proposed approach of BN Machine Learning contributed in exploring speedup by 3 times compared with Random Iterative Compilation.

2) Unsupervised Learning: Unsupervised Learning is special type of machine learning in which a function is inferred to draw the hidden pattern from unlabeled data. Unlike supervised learning, it is free from error signals when evaluating the output. Since all the data which is used for learning is unlabeled. There is no presence of target in Unsupervised Learning, but presence of environment which permits the model to evaluate. Environment does the scoring to each model which is later passed to objective function of model. Under this Section, we would like to discuss Clustering Methods and Evolutionary Algorithms.

Clustering Methods: Clustering is one of the key model in unsupervised learning. It helps in many ways like minimizing the large amount of unrelated passes of compiler optimization into sensible clusters, reducing the compiler optimization space and many more. Many researchers have proposed different usages of Clustering methods. Here, I would like to mention some of those.

Researcher named Thomson presented a method, which is used to reduce the training time. In this method, they have used the clustering technique called Gustafson Kessel Algorithm which is used after undergoing dimension reduction process. They examined this approach on EEMBCv2 Benchmark suite and declared that reducing training time has been decreased by 7times compared to previous one.
The other developer named Ashouri proposed a software/hardware co-design tool chain to learn the compiler design space along with architectural properties of VLIW processor. They have utilized the technique of clustering in order to derive the four well and good hardware architectures. This is continued by selection of optimizations with Kruskal Wallis test and Parato optimal filtering. These are also called as statistical techniques.

3) Reinforcement Learning: Reinforcement Learning is a special area in machine learning which does not come under supervised or unsupervised learning. It is the field which is inspired by behaviorist psychology. It also uses the notion of penalties or rewards by which software agent can freely interact with environment and increase his total reward to maximum extent. The most interesting part in RL is testing and training phases are inter-twined. In order to interact with environment, RL takes the help of Markov decision process, also called as MDP. Author named Coons proposed that they used RL tool called NEAT in order to find the well instruction placements for an EDGE architecture. Indeed, they declared that it could perform better than state-of-the-arts methods. They are many more who are still using RL Prediction types in order to acquire better accuracy than traditional methods.

IV. PREDICTIONS

Under this section, we provide the prediction types for classifying the output. The most vital part of prediction is that the target function should do the following things:

- Choosing the finest collection of compiler optimizations to use.
- For given application, predict the speedup of the sequence of compiler optimization
- Using right set of features in order to identify the application.
- The intermediate speedup predictor
- To reduce the search space with the help of downsampling of optimizations.

We have some of the prediction types like Clustering & Downsampling Prediction, Speedup Prediction, Compiler Sequence Prediction, Tournament & Intermediate Prediction and Feature Prediction.

A. Clustering & Downsampling Prediction

Clustering Prediction is same as Clustering methods discussed under the section of Unsupervised Learning in Machine Learning category. Clustering is also called as Cluster

analysis. It scrutinizes the similarities between collection of variables so that same kind of variables under one group called cluster.

B. Speedup Prediction

Speedup Prediction is the process involved in construction, examination and validation of a model to obtain the outcome of unobserved timing. The speedup prediction tries to build a function which takes the tuple as a input (F , T). F is the vector of collected features of the optimized application and Let T be the one of the possible sequence in compiler optimization in the design space. The output can be the value which is the prediction of speedup that has to be achieved by sequence when applying the source code of the original state's application.

C. Compiler Sequence Prediction

Compiler Sequence Prediction is a special type of model which gives the output of set of best compiler sequences or passes for applying on a given application. Characterizing the application and feeding this to the model. The model produces the prediction of collection of compiler passes which are used to apply for the given application.
Equation : $\delta \in$ O

D. Tournament and Intermediate Prediction

Tournament Predictor is used in prediction of two input sequences of optimizations which are from two different classes. This predictor have the ability to rank the sequences and pick up the best one from them.
Intermediate Prediction is a model in which we repeatedly estimate the speedup of the application which is optimized. In each state we characterize each application along with the sequence of compiler, let it be T. Speedup is predicted by the model. Since both predictors functions in same manner, that's why we apply both on same optimization on the current state. Because of multiple executions of each application to optimize in intermediate predictor, it is slower compared to other methods. But one can say proudly that, it helps in tackling the phase-ordering problem.

E. Feature Prediction

For any prediction, it need features to predict the outcome. Outcome of prediction highly depends on type of features and their characters taken into consideration. Therefore, Feature Prediction Model uses the most important features which can affect performance of a model. So, Choosing the fine features is the most important step and crucial too in evaluating the performance of a model.

V. DISCUSSIONS

Till now, we have discussed about some of the popular using compiler optimization techniques namely DSE and ML Models. We have presented the work using ML Techniques and their applications used by authors and also their main motive in getting the higher accuracy. We also made a discussion about two main major problems namely Selection and Phase Ordering. Now-a-days, deep learning is used in every aspect and we have been witnessed by applications like classification of Images, Recognition of Speech and Text Classification etc.
Since the utility of data is high in deep learning, it needs large data sets which plays a vital role in acquiring the best results. Algorithms of Deep Learning derive unexceptional features through effective learning methods. Because of this, deep learning achieved effective results in many applications. Genesis and CLgen are the synthesizers which have capability to produce synthetic programs in machine learning based autotuning techniques, are bringing diversity as well as authority on new benchmarks to the user.

VI. CONCLUSIONS

This paper is mainly concentrated on Compiler Optimization Strategies which is one of the auto-tuning problem. We have discussed some of the popular approaches and mainly discussed thoroughly on DSE Techniques and ML Models. DSE is preferred as it achieves 40-60% speedup when compared with iterative exploration w.r.t GCC -O2 and -O3 on an ARM embedded board. We have briefly discussed on Phase-Ordering Problem which is an other problem in auto-tuning. But our discussion leads towards dependence of the input data on compiler options which inturns leads towards phase-ordering problem. In the future, research on this field helps in solving many challenges of computer science with high computation performance. In order to reduce the programmer's effort, machine learning along with deep learning becoming more useful and also powerful too. These methods made systems automatically perform tasks without man's effort.

VII. ACKNOWLEGDEMENT

All Images that we have used are taken from various sources and many are from reference papers. In order to present the real images from authors, we have taken from original published papers. The graphs which describes the performances are also imported from original authors.

REFERENCES

[1] Auto-tuning Techniques for Compiler Optimization Amir Hossein Ashouri, Gianluca Palermo and Cristina Silvano DEIB - Politecnico di Milano, ITALY DATE 2016, PhD Forum, Dresden, GERMANY.

[2] Amir Hossein Ashouri. 2012. Design space exploration methodology for compiler parameters in VLIW processors . Masters thesis. M. Sc. Dissertation. Politecnico Di Milano, ITALY. http://hdl.handle.net/10589/72083.

[3] Amir Hossein Ashouri, Giovanni Mariani, Gianluca Palermo, and Cristina Silvano. A bayesian network approach for compiler autotuning for embedded processors. In Embedded Systems for Real-time Multimedia (ESTIMedia), 2014 IEEE 12th Symposium on , pages 9097. IEEE, 2014.

[4] A.H. Ashouri, G. Mariani, G. Palermo, and C. Silvano. 2014. A Bayesian network approach for compiler autotuning for embedded processors. (2014), 9097. DOI: http://dx.doi.org/10.1109/ESTIMedia.2014.6962349

[5] Amir Hossein Ashouri. 2016. Compiler Autotuning Using Machine Learning Techniques . Ph.D. Dissertation. Politecnico di Milano, Italy. http://hdl.handle.net/10589/129561.

[6] Amir H Ashouri, William Killian, John Cavazos, Gianluca Palermo, and Cristina Silvano. 2018. A survey on compiler autotuning using machine learning. arXiv preprint arXiv:1801.04405 (2018).

[7] Amir Hossein Ashouri, Giovanni Mariani, Gianluca Palermo, Eunjung Park, John Cavazos, and Cristina Silvano. 2016. COBAYN: Compiler Autotuning Framework Using Bayesian Networks. ACM Trans. Archit. Code Optim. (TACO) 13, 2, Article 21 (June 2016), 25 pages. DOI: http://dx.doi.org/10.1145/2928270

[8] Amir H. Ashouri, Gianluca Palermo, John Cavazos, and Cristina Silvano. 2018f. Selecting the Best Compiler Optimizations:A Bayesian Network Approach.

[9] Amir Hossein Ashouri, Gianluca Palermo, and Cristina Silvano. An Evaluation of Autotuning Techniques for the Compiler Optimization Problems. In RES4ANT2016 co-located with DATE 2016 . 23 27

[10] A Survey on Compiler Autotuning using Machine Learning Accepted in ACM Computing Surveys 2018

[11] http://ctop.cs.utah.edu/downloads/ACACES/acaces-hall-L1.pdf

[12] P roceedings of 1 st Workshop on Re source Awareness and Application Autotuning in Adaptive and Heterogeneous C omputing (RES4ANT) 2016 An Evaluation of Autotuning Techniques for the Compiler Optimization Problems Amir Hossein Ashouri, Gianluca Palermo and Cristina Silvano Politecnico di Milano, Milan, Italy.