International Journal of Research in Engineering, Science and Management
Volume-2, Issue-5, May-2019
www.ijresm.com | ISSN (Online): 2581-5792

A Review on Different Approaches of Compiler Optimization

Adarsh Anand[1], Abhinandan Jha[2], Kanwaldeep Singh[3]
[1,2,3]*Student, Dept. of Computer Science and Engg., M. S. Ramaiah Institute of Technology, Bengaluru, India*

***Abstract*: This paper describes different compiler optimization techniques. It elaborates the comparison of those different approaches of compiler optimization. There are basically two types machine learning approaches and non-machine learning approaches. The non-machine learning part contains an algorithm for shared-memory multiprocessors. The algorithm considers data locality and parallelism. It also incorporates cache model for optimization. Comparison is done by putting the algorithm into sequential form parallelism. On the other hand, machine learning in compiler optimization has reached new heights. Over a decade from now it has fallen into one of the main stream activity. This paper also incites the relationship of machine learning and compiler optimization by briefing the terms features, models, training etc. It also provides a survey for the variety of research areas.**

***Keywords*: optimization, parallelism, machine learning**

1. Introduction

Compiler optimization is a great field to work on. In a modern day compiler a parameter that is being looked upon a great scale is optimization. Different sorts of compiler are being used for performance check and many of them were successful in comping up with profitable results. One of the method was feedback mechanism that is being used in vectorizing compilers. It was basically used in case of dusty desk programs. Parallelism is great tool and if used properly can fetch us satisfying results. Even compiler optimization can be divided into separate modules and independent modules can be incorporated with parallelism. By using several array references techniques we can also improve space complexity and a step forward towards optimization. The algorithm that has been discussed in this paper is a shared-memory multiprocessor algorithm [1]. It incorporates vectorization and does the performance analysis based on parallelism. In this paper we try to understand how different approaches of compiler optimization works and how will it react with ML incorporated in it.

Machine learning is a 21st century term which is building its market rapidly. It is always easier if we allow the machines to get the higher order computations. It is known that compiler optimization depends on various features. And whenever an algorithm is being designed all the factors effecting the optimization is kept in consideration. In case of ML we let the machine decide the best features responsible for compiler optimization and then design the algorithm. It is quite a sequential form of execution with a low factor of parallelism. In this type of approach the machine tries to improve the optimization based on previous datasets. We leave the analysis part to machines that which factor to be altered to get the best output.

2. Literature Survey

In the present world there are different types of work done in the field of compiler optimization. Compiler optimization still consists of vast scope of work to be done. People in market always try to stand profitable due to which much methodologies are yet not being made available publicly. Shared memory parallelization is mostly considered for dusty desk programs. Parallelism is considered via grid method, analysis and performance of the working algorithm doesn’t just depend on just parallelism but also on data locality and granularity. Caches are being preferred in performance analysis. As locality plays a vital role on system caches help a lot. It also improves the time complexity. The recent work that has being done by Polaris was also kept unpublished. But the algorithm that is being used, they have not used caches. The no of loops working parallelly is considered as a performance measure. The algorithm can use the predefined loop bounds and find granularity and locality. Wolf and Lang’s algorithm has certain expenses due to exponential behavior in depth n. The discussed algorithm has a complexity of O(n log(n)) but in worst case scenario it can go up to O(n*n) [1]. Finding the reuse is the expensive part. The proposed algorithm iterates through several steps and then finally finds an optimization to get the reuse. It evaluates the reuse for every permutation. This includes various other methodologies but none of them have fusion, distribution or interprocedural analysis.

3. Technical definitions

In case of shared microprocessor optimization algorithm some terms we need are:

A. Data dependence

When parallelism comes into picture the data dependency must be kept under watch. It is necessary to restrict that two different parallel functions mustn’t deal with one parameter simultaneously. On the other hand the process is quite smooth if the functions are independent on dealing with parameters. If we see in comparison with sequential model of programming the dependency is quite less after including parallelism [1].

B. Sources of Data Reuse

Data is power and why not make an optimum use if possible. Sometimes multiple arrays refer the same memory unit. If a particular set of data has to be considered for several different modules then instead of including it all of them and increasing the space-complexity we can have single array reference and multi array reference. Sometimes same array access different locality it comes under spatial locality while temporal locality means accessing the same memory unit with same or different arrays [1].

C. Feature Engineering

In a ML model the output is predicted based on the previous input. So, it is very important to select the feature carefully. The better the feature selection the better is the final prediction. The feature can be of different data types but while it is fed into a ML model it must be compatible. Generally standard models use numerical or boolean values, as it is proper for evaluation. Sometimes even after selecting the features they are not scalable together. A term feature scaling helps us to handle that situation. The process of feature selection and tuning is referred to as feature engineering. This process may need to iteratively perform multiple times to find a set of high-quality features [2].

D. Training a Model (Learning)

In this stage the ML model is trained with the training dataset. It plays a vital role in getting a better prediction. A good diverse dataset is the main component to complete this procedure [2].

4. Discussion

The parallelization algorithm is one of the non-machine learning approaches used for compiler optimization. It basically exploits both data locality and granularity of parallelism. The main component is driver which works in a top-down approach recursively. It means if any node has to be visited it's predecessor has to be visited first. Initially all the nodes are marked unvisited and then procedures are being called incorporated with parallelism. Parallelizer introduces parallelism, by using various transformations. If that doesn't happen further steps are taken. Driver handles interprocedural loop transformations.

Loop permutation is applied for locality and optimization. Target to use the inner loops more effectively. The algorithm performs the following four steps:

- It introduces both the two different types of localization based on space and temporal use which is being exhibited by array references.
- Whenever there is algorithm there must be cost optimization. In this algorithm the parameter taken for measurement is the magnitude of no of lines of cache.
- It finds all the combinations by which the cost complexity can be reduced i.e. no of lines of cache accessed can be reduced.
- It also incorporates parallelism as a functionality of the outer loop. It maintains locality in every processor. The total overhead is divided among all the components.

Reference Group algorithm tries to reduce the overhead due to the no of lines of cache. It finds reference with group related spatial and temporal locality. If the subscript is independent of the variable, then only one cache line is enough for all iterations. If the subscript depends on the variable then for every iteration, we need a new cache line which increases overheads.

This is quite the roll if we look at non-ML techniques. We should change the paradigm.

If we change our lens for ML techniques, then there are several other ways in which we will achieve the optimization. The two basic approaches of learning are supervised and unsupervised.

In supervised we have certain parameters and the domain very specific. The iterations and transformations are deterministic. It is generally under fully-observable environment. In case of unsupervised learning the transitions are not deterministic. The machine explores different states on its own.

```
INPUT:  $n$ is a procedure or call node in the $G_{ac}$, loop nodes are handled within Driver
ALGORITHM:
  procedure Driver($n$)
      if $n$ is a procedure and any predecessor of $n$ is not visited return
      mark $n$ visited
      if $n$ is a procedure and all predecessors of $n$ are marked optimized return
      if $n$ is a call node to procedure $p$ Driver ($p$) {skip over call nodes}
      if $n$ is a procedure node
        partition the outer loops nodes $\mathcal{L} = \{l_1, \ldots, l_k\}$ of $n$ into sets of adjacent outer loops
          $\mathcal{R}_i = \{\{l_1, \ldots, l_j\}, \ldots, \{l_r, \ldots, l_k\}\}$
        forall $i$ Parallelize($\mathcal{R}_i$)
        forall $l_k$
          if $l_k$ now contains a parallel loop
            mark it and call nodes nested within the parallel loop optimized and visited
          forall $s$, call nodes nested in $l_k$ not surrounded by a parallel loop Driver($s$)
        endforall
      endif
```

Fig. 1. Driver for parallelization algorithm

Some of the basic techniques of supervised learning are regression and classification are given in Table 1. There are several types of regression such as linear, logistic etc.

Even Classification is helpful if there is binary or boolean output. If the dataset is very large then random forest classification perhaps.

A. Linear Regression applied on a dataset

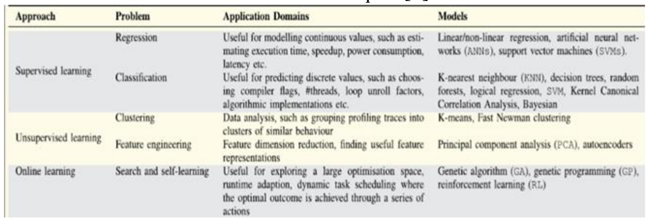
Regression is actually curve fitting and it is necessary to keep in check that the curve doesn't overfit in case of logistic regression and polynomial regression [3].

In classification [4] K-means classification is the most famous method, but it covers only a range of problems. It may be fast and effective but it actually doesn't get trained. It just finds the K nearest neighbors by using Euclidean Distance as parameter. There can be other parameters which can be helpful in some problems but it only takes the distance. This means that the algorithm is not robust to noisy training data and could choose an ill-suited training program as the prediction.

As an alternative, the decision tree has been used in prior works for a range of optimization problems. These include choosing the parallel strategy for loop parallelization, determining the loop unroll factor, deciding the profitability of using GPU acceleration, and selecting the optimal algorithm implementation [2]. The advantage of a decision tree is that the learned model is interpretable and can be easily visualized. This enables users to understand why a particular decision is made by following the path from the root node to a leaf decision node.

Decision trees mainly cover the major part sometimes, in decision trees the main part is again feature engineering. Initially we try to get the best feature for classification. After classifying with help of that feature we for the next best, in this way the decision tree is formed.

Table 1
Basic Techniques [2]



Approach	Problem	Application Domains	Models
Supervised learning	Regression	Useful for modelling continuous values, such as estimating execution time, speedup, power consumption, latency etc.	Linear/non-linear regression, artificial neural networks (ANNs), support vector machines (SVMs).
	Classification	Useful for predicting discrete values, such as choosing compiler flags, #threads, loop unroll factors, algorithmic implementations etc.	K-nearest neighbour (KNN), decision trees, random forests, logical regression, SVM, Kernel Canonical Correlation Analysis, Bayesian
Unsupervised learning	Clustering	Data analysis, such as grouping profiling traces into clusters of similar behaviour	K-means, Fast Newman clustering
	Feature engineering	Feature dimension reduction, finding useful feature representations	Principal component analysis (PCA), autoencoders
Online learning	Search and self-learning	Useful for exploring a large optimisation space, runtime adaption, dynamic task scheduling where the optimal outcome is achieved through a series of actions	Genetic algorithm (GA), genetic programming (GP), reinforcement learning (RL)

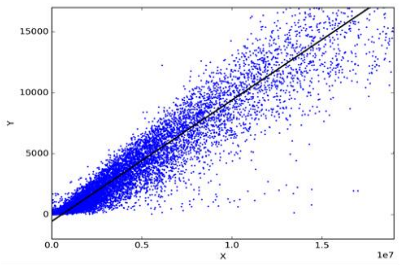
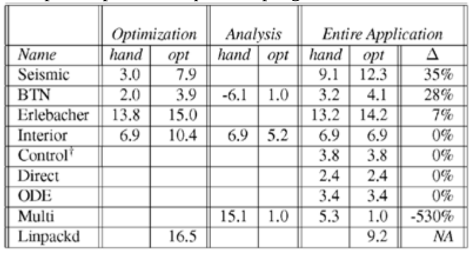


Fig. 2. Linear regression

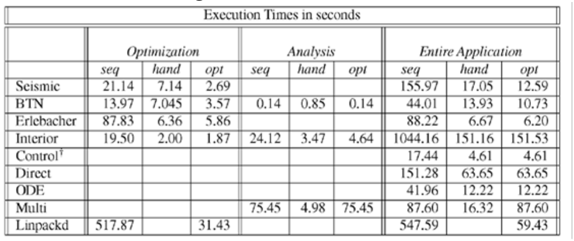
5. Results

Table 2
Speed-ups over sequential program versions [1]



	Optimization		*Analysis*		*Entire Application*		
Name	*hand*	*opt*	*hand*	*opt*	*hand*	*opt*	Δ
Seismic	3.0	7.9			9.1	12.3	35%
BTN	2.0	3.9	-6.1	1.0	3.2	4.1	28%
Erlebacher	13.8	15.0			13.2	14.2	7%
Interior	6.9	10.4	6.9	5.2	6.9	6.9	0%
Control†					3.8	3.8	0%
Direct					2.4	2.4	0%
ODE					3.4	3.4	0%
Multi			15.1	1.0	5.3	1.0	-530%
Linpackd		16.5				9.2	*NA*

Table 3
Program execution times [1]



Execution Times in seconds									
	Optimization			*Analysis*			*Entire Application*		
	seq	*hand*	*opt*	*seq*	*hand*	*opt*	*seq*	*hand*	*opt*
Seismic	21.14	7.14	2.69				155.97	17.05	12.59
BTN	13.97	7.045	3.57	0.14	0.85	0.14	44.01	13.93	10.73
Erlebacher	87.83	6.36	5.86				88.22	6.67	6.20
Interior	19.50	2.00	1.87	24.12	3.47	4.64	1044.16	151.16	151.53
Control†							17.44	4.61	4.61
Direct							151.28	63.65	63.65
ODE							41.96	12.22	12.22
Multi				75.45	4.98	75.45	87.60	16.32	87.60
Linpackd	517.87		31.43				547.59		59.43

6. Conclusion

Seeing the vast domain of problems, we can conclude that all the different form of approaches are important but under different circumstances. The first method that has been explained in this paper that has used parallelism can be helpful if the optimization has been done independent modules with very less interconnection. The modules are developed independently and then combined together for testing. In this case separate teams can work in parallel but if there is some sort of sequential execution then ML incorporation will be a great tool.

References

[1] K. S. McKinley, "A compiler optimization algorithm for shared-memory multiprocessors," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 8, pp. 769-787, Aug. 1998.

[2] Z. Wang and M. O'Boyle, "Machine Learning in Compiler Optimization," in *Proceedings of the IEEE*, vol. 106, no. 11, pp. 1879-1901, Nov. 2018.

[3] C. Luk, S. Hong and H. Kim, "Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping," *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, New York, NY, 2009, pp. 45-55.

[4] M. Stephenson and S. Amarasinghe, "Predicting unroll factors using supervised classification," *International Symposium on Code Generation and Optimization*, New York, NY, 2005, pp. 123-134.