PDPM
Indian Institute of Information Technology,
Design & Manufacturing, Jabalpur

Compiler Design Project Report

Machine Learning based Compiler Optimization

Group 17

Siddharth Ubana	–	2011145
A. Krishna	–	2011182
Ankit Chaudhary	–	2011187
Swapnil Bharadwaj	–	2011230
Abhishek Bhardwaj	–	2011242

Table of Contents

1. Abstract

Selecting a good set of compiler optimization flags is a challenging task due to the fast change and rapid improvement in microprocessor technology along with static and dynamic profile of programs. Recent work has shown that machine learning can automate and in some cases outperform hand crafted compiler optimizations. Central to such an approach is that machine learning techniques typically rely upon summaries or features of the program. The quality of these features is critical to the accuracy of the resulting machine learned algorithm; no machine learning method will work well with poorly chosen features. However, due to the size and complexity of programs, theoretically there are an infinite number of potential features to choose from. The compiler writer now has to expend effort in choosing the best features from this space. Our aim is to develop a machine learning based algorithm for self-tuning compilers which will generate a combination of compiler flags that will minimize Run-time/ Compile-time/ Code-size etc.

2. Introduction

Fast change and rapid improvement in microprocessor technology along with evolving static and dynamic profile of programing make the job tough for selecting a good set of compiler optimization flags. In some cases performance improvement can be made over maximum optimization level i.e.–O3 (GCC) or –fast (pgi) where in GCC O3 means the set of optimization flags that selected for the optimization of GCC.

Objective

The task is to select compiler optimization flags that give us best possible compiler optimization for a particular set of compilers and that is where we implement machine learning techniques.

Pros and cons

Many program optimization problems are NP-Complete because there is no way to proof that the given set is best possible optimization way.

Implications

- It assumed that the problem is intractable.
- Exponential algorithms require global optimization.
- Heuristics or approximation algorithms are used which provide fast solutions but efficiency is compromised.

3. Analysis Methods

Machine Learning

Goal of using Machine Learning is to predict good optimization flags for a new (unseen) program based on previously seen programs, while reducing time required by iterative compilation. Classification of unseen programs is performed by a learning algorithm that compares the features of the new program with others in the training database and selects the flags from the best match.

Supervised learning

A training set is provided to the learning algorithm that includes inputs and known outputs or targets. Algorithm generalizes a function from inputs and targets in training set.

Flags

Flags are environment variables that are used to tell the GNU compiler collection what kind of switches to use when compiling source code for GCC. These flags are referred as CFLAGS and are written in C.
Flags can be used to decrease the amount of debug messages for a program, increase error warning levels, and of course, to optimize the code produced.
Note: Flags can be a very effective means of getting source code to produce smaller and faster binaries. They can also impair the function of your code, bloat its size, slow down its execution time, or even cause compilation failures!

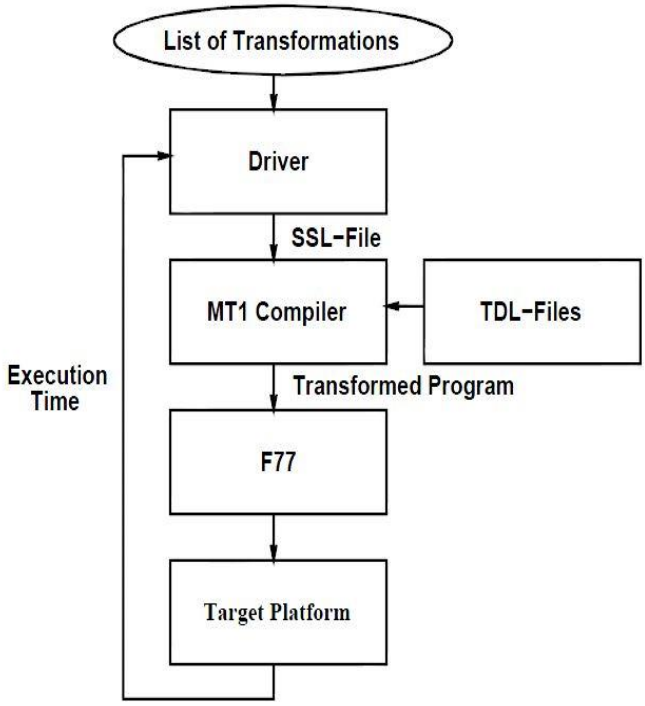
Optimization of flags

Turning on optimization flags makes the compiler attempt to improve the performance and code size at the expense of compilation time and possibly the ability to debug the program, not all optimizations are controlled directly by a flag. Only optimizations that have a flag are included in the scope of this project.

Iterative compilation

In this section we briefly discuss how the iterative compilation system is implemented. Figure shows an overview of the system. The compilation system is centred on a global driver that reads a list of transformations that it needs to examine together with the range of their parameters.

Global driver has two mechanisms to control the application of transformations: a Transformation Definition Language (TDL) and a Strategy Specification Language (SSL).



For more detail on iterative compilation the following link can be accessed
http://link.springer.com/article/10.1007/s10766-010-0161-2

Set of Flags

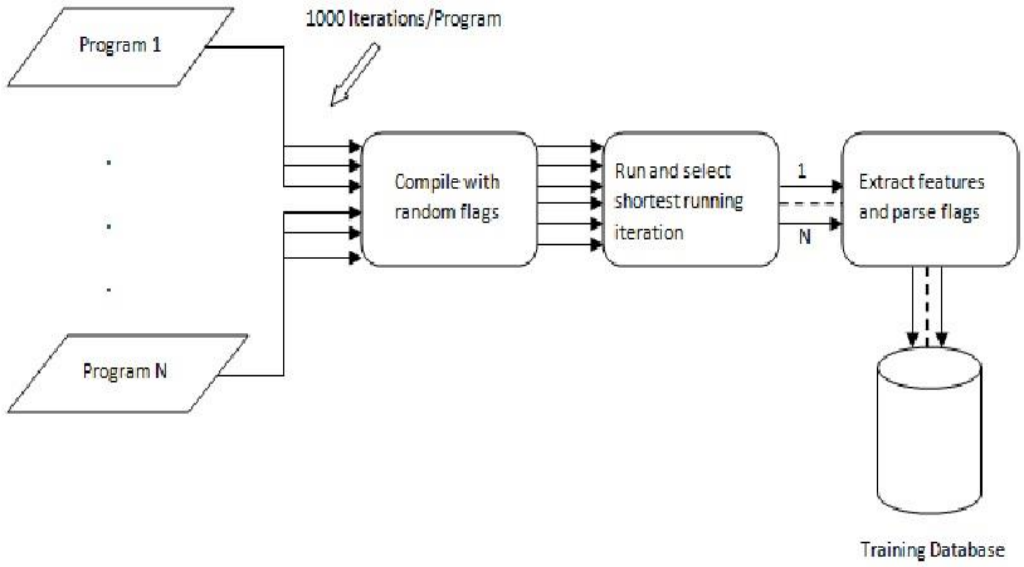
Depending on the target and how the compiler is configured, set of optimization flags are selected for GCC compilers. A set of flags is defined at each level of –O. –O is shown in the following figure. It means the set of optimization flags that selected for the optimization of GCC compiler. We can invoke GCC with –Q –help. Optimizers to find the exact set of optimizations that are enabled at each level.
With –O, the compiler tries to reduce the code size and execution time, without performing any optimizations that take a great deal of compilation time -O turns on the following optimization flags:

```
-fthread-jumps
-falign-functions  -falign-jumps
-falign-loops  -falign-labels
-fcaller-saves
-fcrossjumping
-fcse-follow-jumps  -fcse-skip-blocks
-fdelete-null-pointer-checks
-fdevirtualize -fdevirtualize-speculatively
-fexpensive-optimizations
-fgcse  -fgcse-lm
-fhoist-adjacent-loads
-finline-small-functions
-findirect-inlining
-fipa-sra
-fisolate-erroneous-paths-dereference
-foptimize-sibling-calls
-fpartial-inlining
-fpeephole2
-freorder-blocks  -freorder-functions
-frerun-cse-after-loop
-fsched-interblock  -fsched-spec
-fschedule-insns  -fschedule-insns2
-fstrict-aliasing -fstrict-overflow
-ftree-switch-conversion -ftree-tail-merge
-ftree-pre
-ftree-vrp
```

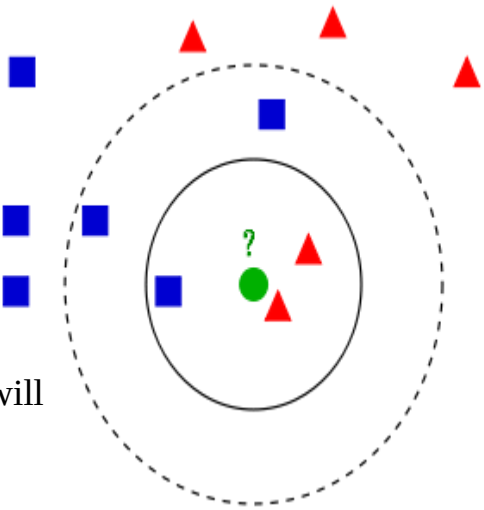
4. Selection of ML algorithm

K-nearest neighbour's algorithm is an instance based learning algorithm, it matches the given training instance to find the k closest match. It includes that instance in that class and treats it like that. It uses Euclidean distance to calculate the closest match between two vectors.
Major drawback of this classifier is that it is a lazy learner and it does a lot of processing during classification and it requires a large set of training data to classify the instances correctly.

Training of algorithm



2D representation of K-NN algorithm



- Assume square and triangle represent two program classes.
- Green circle represents the new query instance and we have to classify it into one of the two classes.
- For K = 1. Green query point belongs to red triangle.
- For K = 3. Green query point belongs to red triangle.
- For K = 5. Situation changes and green query point belongs to the blue square class.
- As we increase k we get more generalize result but it will increase time complexity for algorithm

Formula used in K-NN algorithm

Given x_q take vote among its k nearest neighbours (discrete-valued target function). Take mean of f values of k nearest neighbours (if real-valued).

$$f(x_q) = \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

Distance weighted NN algorithm

Since nearer neighbours affect the new classified instance more crucially. We must give them more preference to attach weight to each neighbour.

Might want weight nearer neighbors more heavily...

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

and $d(x_q, x_i)$ is distance between x_q and x_i

Note now it makes sense to use *all* training examples instead of just k

5. Implementation

Python Code of K – NN Algorithm

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Feature set containing (x,y) values of 25 known/training data
trainData = np.random.randint(0,1000,(25,2)).astype(np.float32)

# Labels each one either Red or Blue with numbers 0 and 1
responses = np.random.randint(0,2,(25,1)).astype(np.float32)

# Take Red families and plot them
red = trainData[responses.ravel()==0]
plt.scatter(red[:,0],red[:,1],80,'r','^')

# Take Blue families and plot them
blue = trainData[responses.ravel()==1]
plt.scatter(blue[:,0],blue[:,1],80,'b','s')

plt.show()

newcomer = np.random.randint(0,100,(1,2)).astype(np.float32)
plt.scatter(newcomer[:,0],newcomer[:,1],80,'g','o')

knn = cv2.KNearest()
knn.train(trainData,responses)
ret, results, neighbours ,dist = knn.find_nearest(newcomer, 3)
print "result: ", results,"\n"
print "neighbours: ", neighbours,"\n"
print "distance: ", dist

plt.show()
```

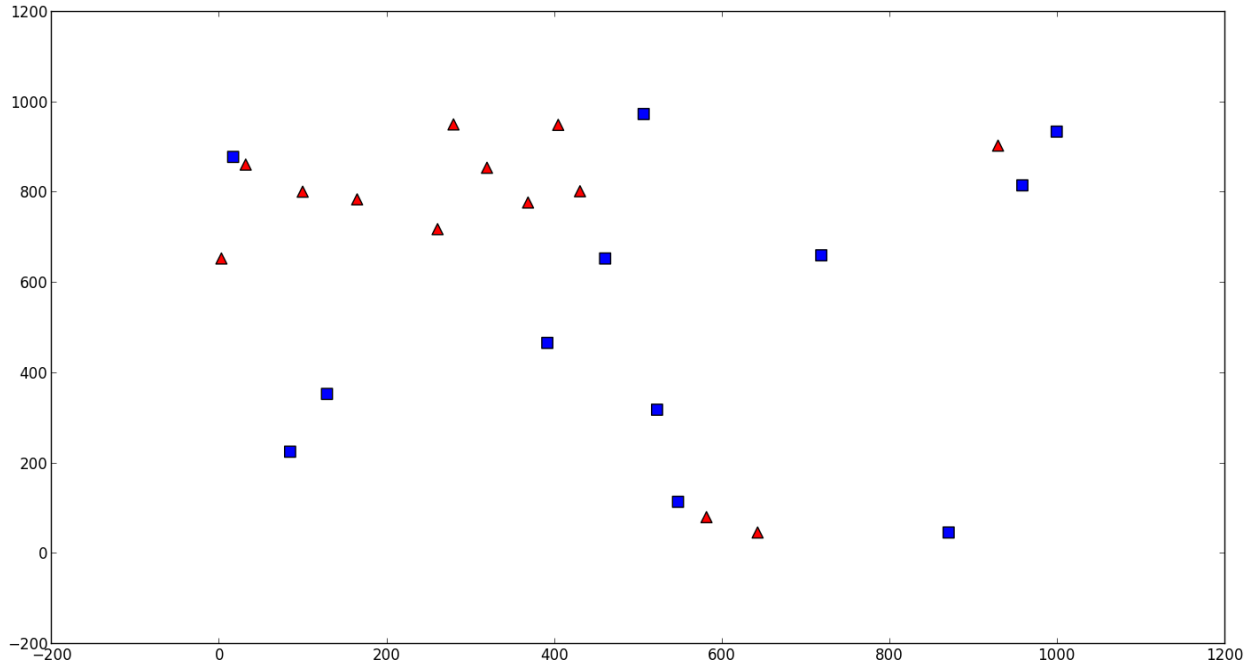
6. Results

Result and Demonstration

```
Python 2.7.6 Shell
File  Edit  Shell  Debug  Options  Windows  Help
Python 2.7.6 (default, Nov 10 2013, 19:24:18) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
result:  [[ 1.]]

neighbours:  [[ 1.  1.  1.]]

distance:  [[  34025.  102265.  281573.]]
>>>
```



7. References

- [FKMC11] Grigori Fursin, Yuriy Kashnikov, Abdul Wahid Memon, Zbigniew Chamski, et al. Milepost. GCC: Machine Learning Enabled Self-tuning Compiler. *International Journal Parallel Programming*. 39:296-327, 2011.
- [MAR09] Stephen Marsland. *Machine Learning An Algorithmic Perspective.* Chapman & Hall/CRC, Boca Raton, FL, 2009.
- [MIT97] Tom M. Mitchell. Machine Learning. McGraw-Hill, Boston, MA, 1997.
- [SEG07] Toby Segaran. *Programming Collective Intelligence.* O'Reilly, Sebastopol, CA, 2007.