Analysis of Mibench Benchmark Applications Using GCC Compiler

Andrews.J, Dr.Sasikala.T

Abstract—Tuning compiler optimization for rapidly evolving hardware makes really tough task as for as optimizing compiler is concerned. The main objective of compiler optimization is to run the code faster with lesser memory. Recent version of compiler provides larger number of optimization techniques. By applying all these techniques to a given application degrade the program performance, so selecting best set for a given application is not an easy task. Selecting best set of optimization techniques depends upon computer architecture and problem domain. Finding the best compiler optimization for a particular problem can have a significant impact on program performance. There have been several proposed techniques to search the space of compiler options to find good solutions but such approaches can be expensive. In this paper, it is proposed to study the various benchmark problems and comparison of its execution time using core i7 processor. For this work we proposed an automated framework to select the compiler options for a particular problem from a larger set of options. Many previous works consider only limited set of options. For this framework, we implemented compiler optimization selection algorithm as advanced combined elimination algorithm and this algorithm is compared with different levels of optimization techniques.

Keywords—Compiler Optimization, GCC Optimization, Advanced combined elimination, MiBench.

I. INTRODUCTION

RECENT version of compiler provides a larger set of optimization techniques, for users to fine tune the performance of various benchmark applications. But selecting the best set of options is not an easy, task especially for average users who do not have in depth understanding of the compiler options. Because selecting best set of options from various candidate options depends on system architecture and problem domain. Each optimization tries to improve the performance of the application, although they are not always effective. Through command line flags, the user can decide which optimizations are to be applied in a given compilation run, but clearly it is not a long term solution. As compiler options get increasingly numerous and complex, the problem must find an automated solution.

Because modern compiler provides larger number of optimization techniques and complex interactions among its various techniques finding optimal functions for specific benchmark applications is hard and time consuming process. There has been much previous work on automatically searching for the best optimization settings. This previous work is based on iteratively enabling certain optimizations, running the compiled program and, based on its performance, deciding on a new optimization setting. Pan et al introduce a new algorithm called combined Elimination (CE) [2],[3],[4-5] that was shown to outperform all previous search-based techniques in finding good optimization settings with considerably fewer evaluations. However, these pure search or “orchestration” approaches do not use prior knowledge of the hardware, compiler, or program and instead attempt to obtain this knowledge online. Thus, every time a new program is optimized, the system starts with no prior knowledge.

In this paper we have considered modified version of combined elimination algorithm. This algorithm is compared with different levels of GCC compiler optimization techniques. GCC compiler provides three levels of optimization techniques [1],[8-9]. To obtain the best performance a user usually applies the highest optimization level –O3. In this level the compiler perform the most extensive code analysis and expects the compiler generated code to deliver the highest performance. In this paper we proposed an automated framework to select the compiler options for a particular problem from large set options for an i7 processor. Many previous works consider only limited set of options. For this framework, we implemented compiler optimization selection algorithm advanced combined can decide which optimizations are to be applied in a given compilation run, but clearly it is not a long term solution. As compiler options get increasingly numerous and complex, the problem must find an automated solution.

Automatically selecting the best set of compiler optimizations for a particular program is a difficult task and there has been much previous work on automatically searching for the best optimization settings. This previous work is based on iteratively enabling certain optimizations, running the compiled program and, based on its performance, deciding on a new optimization setting. Pan et al introduce a new algorithm called combined Elimination (CE) [3],[4],[6-7] that was shown to outperform all previous search-based techniques in finding good optimization settings with considerably fewer evaluations. However, these pure search or “orchestration” approaches do not use prior knowledge of the hardware, compiler, or program
and instead attempt to obtain this knowledge online. Thus, every
time a new program is optimized, the system starts with no prior
knowledge.

In this paper we have consider GCC Version (4.3.2).GCC
provides three levels of optimization techniques [1]. To obtain
the best performance a user usually applies the highest
optimization level –O3. In this level the compiler perform the
most extensive code analysis and expects the compiler
generated code to deliver the highest performance. In this paper
we proposed an automated framework to select the compiler
options for a particular problem from large set options. Many
previous works consider only limited set of options. For this
framework, we implemented compiler optimization selection
algorithm advanced combined elimination. This algorithm
compared with different levels of optimization techniques.

The remainder of this article is organized as follows. Section
II describes levels of different optimization techniques. Section
III describes generalized framework. Section IV describes
advanced combined elimination algorithm. Section V describes
experimental setup and results.

II. LEVELS OF DIFFERENT OPTIMIZATIONS

The general optimization flags implemented in GCC are as
following.
• O0 (Level 0)
  No optimization.
• O1 (Level 1)
  The compiler tries to reduce code size and execution time,
  without performing any optimizations that take a great deal of
  compilation time.
• O2 (Level 2)
  The 2nd level optimization enables all optimization invoked
  by the 1st level optimization flag and a few other optimization
  sets tailored to increase the performance of the code. In this
  optimization level, GCC performs nearly all supported
  optimizations that do not involve a space-speed tradeoff.
• Os (Level 2.5)
  Optimizes for code size over performance.
  This special optimization level (-Os or size) enables all -O2
  optimizations that do not increase code size.
• O3 (Level 3)
  The 3rd and highest level optimization level enables even
  more optimizations by putting emphasis on speed over size.
  This includes optimizations enabled at -O2 and rename register.
  Although optimization level 3 (-O3) can produce faster code,
  the increase in the size of the binary image can have adverse
effects on its speed.

III. GENERALIZED FRAMEWORK

The option selector first selects a set of optimization options
from the list of available optimization options [6]. The source
program is next compiled with this set of optimization options
to generate the target code. The target code is then simulated
with the performance profiled. The profiled performance is then
sent back to the option selector as a feedback to select hopefully
a better set of optimization options.

The option selector is mainly an optimization algorithm that
tries to search a set of best optimization options for an input
source program. There are three issues that make this
optimization problem a very hard and time consuming problem.
First, the large number of optimization options makes the search
space huge. Second, the performance of a set of optimization
options for such a source program can only be known by
compiling source program with this set of optimization options
and then profiling the performance of the generated code using
simulated execution. Third the interaction between optimization
options depend on the input source code is very hard to find any
heuristics to speed up search process.

IV. PROBLEM STATEMENT

Given a set of “n” ON-OFF optimization options
{F1,F2…Fn}, find the best combination of flags that minimizes
application execution time using advanced combined
elimination algorithm. In this paper a novel performance tuning
algorithm advanced combined elimination algorithm
implemented which picks best set of options which improves
execution time.

A Advanced Combined Elimination (Ace) Algorithm

Let S be the Set of available Optimization functions.
Let B represents selected compiler options set.
i) S= {F1, F2…Fn} and B= {F1=1, F2=1…Fn=1}
ii) Compile the program with baseline configuration and
measure the program performance. Calculate RIP (Fi) for each
optimization. Check if RIP (Fi) has already computed, if so use
the pre computed value. If RIP (Fi) <0, then
iii) If RIP (Fi) < 0, then
Store Ni, where i=1, 2..j .Elements in N are sorted in an increasing order. Eliminate Nj from S and set Nj to 0 in B. For each k=2 to j, compute RIP (Nk), if RIP (Nk) < 0, then remove Nk from S and set Nk to 0.
Else
B represents the selected compiler option set
iv) Repeat steps ii and iii
v) Stop

V. EXPERIMENTAL SETUP

For testing different set of benchmark applications we have considered recent version of GCC compiler. GCC provides three levels of optimization techniques. Previous work considered only limited set of optimization techniques. This paper proposes larger set of optimization techniques. Table I shows different levels of optimization techniques from –o0 to –o3. We have run an experiment on a set of benchmark application using advanced combined elimination algorithm. We select the following benchmarks from the MiBench.

A Benchmark Applications

The Mibench benchmark [7],[10-11] suite programs were used to experiment the proposed algorithm. These benchmark suites are comparable with SPEC benchmark suite.
1. Bzip2: for compression
2. Consumer_jpeg_c: To add annotations, titles, index terms, etc in JPEG files.
3. consumer_tiff2bw: converts an RGB to a grayscale image by combining percentages of the red, green and blue channels.
4. qsort: for performing sorting
5. dijkstra: for computing shortest paths.
6. patricia: datastructure used in place of full trees with very sparse leaf nodes.
7. security_blowfish: document encoding and decoding.
8. SUSAN: feature detection

B Metrics used for Evaluation

Relative Improvement Percentage (RIP), RIP (Fi), which is the relative difference of the execution times of the two versions with and without Fi.
RIP (Fi) = \( \frac{T(Fi=0)-T(Fi=1)}{T(Fi=1)} \times 100 \)  \[1\]
If Fi=1 then Fi is ON, else OFF
The baseline of this approach switches on all optimizations.
\( T_B = \frac{T(Fi=1)+T(F1=1,F2=1,..,Fn=1)}{N} \), Where \( T_B \) represents base time.
RIP (Fi=0)=\( \frac{T(Fi=0)-T_B}{T_B} \times 100 \)  \[2\]
If RIP (Fi=0) < 0, the optimization of Fi has a negative effect, so it is better to turn off the function.

Architecture used for testing was Intel Corei7 -2630 QM CPU 2.2Ghz. With 8GB RAM, using ubuntu operating system. And the compiler was GCC 4.3.2

VI. RESULTS AND DISCUSSIONS

The above benchmark applications tested with the help of advanced combined elimination algorithm. This algorithm is compared with the execution time speed up over different levels of optimization techniques. Results shows that o3 level performs better when compared with other levels of optimization techniques. Finally this result is compared with advanced combined elimination.

Table I

<table>
<thead>
<tr>
<th>Benchmark applications</th>
<th>Levels of optimization</th>
<th>Execution Time in Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bzip2</td>
<td>O0</td>
<td>8.81</td>
</tr>
<tr>
<td></td>
<td>O1</td>
<td>5.28</td>
</tr>
<tr>
<td></td>
<td>O2</td>
<td>6.26</td>
</tr>
<tr>
<td></td>
<td>O3</td>
<td>6.38</td>
</tr>
<tr>
<td>Consumer_jpeg.c</td>
<td>O0</td>
<td>9.59</td>
</tr>
<tr>
<td></td>
<td>O1</td>
<td>7.00</td>
</tr>
<tr>
<td></td>
<td>O2</td>
<td>6.86</td>
</tr>
<tr>
<td></td>
<td>O3</td>
<td>6.81</td>
</tr>
<tr>
<td>Consumer_tiff2bw</td>
<td>O0</td>
<td>13.78</td>
</tr>
<tr>
<td></td>
<td>O1</td>
<td>7.49</td>
</tr>
<tr>
<td></td>
<td>O2</td>
<td>7.16</td>
</tr>
<tr>
<td></td>
<td>O3</td>
<td>6.97</td>
</tr>
<tr>
<td>Network_dijkstra</td>
<td>O0</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>O1</td>
<td>0.77</td>
</tr>
<tr>
<td></td>
<td>O2</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>O3</td>
<td>0.69</td>
</tr>
<tr>
<td>Network_Patricia</td>
<td>O0</td>
<td>4.52</td>
</tr>
<tr>
<td></td>
<td>O1</td>
<td>2.99</td>
</tr>
<tr>
<td></td>
<td>O2</td>
<td>3.02</td>
</tr>
<tr>
<td></td>
<td>O3</td>
<td>3.00</td>
</tr>
<tr>
<td>Qsort</td>
<td>O0</td>
<td>8.12</td>
</tr>
<tr>
<td></td>
<td>O1</td>
<td>5.65</td>
</tr>
<tr>
<td></td>
<td>O2</td>
<td>5.50</td>
</tr>
<tr>
<td></td>
<td>O3</td>
<td>5.31</td>
</tr>
<tr>
<td>Security_blowfish</td>
<td>O0</td>
<td>14.18</td>
</tr>
<tr>
<td></td>
<td>O1</td>
<td>7.57</td>
</tr>
<tr>
<td></td>
<td>O2</td>
<td>7.33</td>
</tr>
<tr>
<td></td>
<td>O3</td>
<td>7.16</td>
</tr>
</tbody>
</table>

The table I shows execution time speeds up of every benchmark application over different levels of optimization techniques. Advanced combined elimination algorithm implemented to find execution time of various benchmark applications.

36
VII. CONCLUSIONS AND FUTURE WORK

In this paper we proposed an automatic selection framework which picks up best set of optimization techniques from “n” number of optimization techniques. It is very hard to find any heuristics to speed up the searching of the optimal option. We propose an advanced combined elimination algorithm to search for optimization options better than different levels of optimization techniques. Advanced combined elimination algorithm works better for most of the benchmark applications. In future we can design a generalized framework independent of architecture by considering larger set of benchmark applications.

REFERENCES

[10] www.networktheroy.co.uk

J. Andrews received the B.E degree in Computer Science & Engineering from Dr. sivanthi Aditanar college of Engineering, Manonmanium Sundaranar University Tirunelveli, India in 1999 and M.E degree in Computer Science & Engineering from Sathyabama University, Chennai, India in 2006. Where he is currently doing research in the area of Compiler Optimization in Computer Science & Engineering at Sathyabama University, Chennai, India.

He works currently as an Assistant Professor for the Department of Information Technology at SRR Engineering College, Chennai and he has more than 10 Years of teaching experience. He has participated and presented many Research Papers in International and National Conferences. His area of interests includes Compiler Design, Theory of Computation and System Programming.