Abstract:
Compilers have many optimization sequences to be run on a program IR (Intermediate Representation). Applying all optimization sequence leads to degradation of performance. It needs a best optimization set from optimization sequence for every program to improve performance. The literature addresses standard universal optimization sequences set derived from set of program spaces. Retrieving best optimization sequences form this set consumes more time. This paper proposes a sequence selection approach that reduces time for selecting best optimal sequence set and further it reduces runtime of a program space. The experimental results showed improved performance on different program spaces in different bench mark suites.

Keywords: Optimal Sequence, Benchmark Suites, Intermediate Representation

Introduction
Compiler is the main component for any programming language which is used for converting the one language to another language. Compilation of any program has 6 phases [24]. The compiler generates Intermediate representation (IR) for a program when it is processed through the syntactic and semantic analysis [3]. The next phase of the compiler is code optimization which is used to optimize the IR code with respect to parameters such as memory, run time and power consumption [2].

The optimization phase which consists of many optimization sequences which are helpful for optimization of IR code [8]. The optimized code is given as input for the next phase of the compiler for generating the code for target system [10]. The execution time and memory of a program depends on the optimization phase.
Related Work

John Canvazos et al [1] has proposed a best machine learning approach using hardware performance counters [2] to automatically select the best optimization options. In recent research, learned-models [4, 5, 6, 7] are useful for knowing the optimization sequences of the compiler.

Dynamic compilers can select best “optimizations use logistic regression technique [8]. The optimization sequence space size grows exponentially as a function of sequence length. If there are $k$ optimizations, then there are $k^l$ optimization sequences of length $l$ [7, 8]. An optimization sequence is optimal for a program if it hints to least program runtime when associated to all other sequences [7, 6]. This definition is incomplete as the program runtime and its behavior could be a function of the input [8, 9]. An optimal sequence for a program depends on the program characteristics as defined by the input to the program [12, 13]. A program with different input distribution and parameters with respect to that input distribution almost remains constant [11].

Predictive heuristic technique is an approach to find optimization sequences. Unpredictable interaction among optimizations in this technique leads to performance degradation. Spyridon et al [19] proposed optimization sequence exploration technique [20, 21, 22, 23] (OSE) as first iterative compilation model for general purpose compilers to overcome the problem in predictive heuristics.

Iterative compilation techniques [15] use efficient algorithms to find good optimization sequences. In this technique program evaluations are very large and not feasible to apply in applications. In machine learning approaches [17] prediction models are used to determine good optimization sequences. Prediction models learn by running and observing some set of programs and their runtimes. It will take decisions on optimizations to be applied on program space [18].

Thomson et al [16] proposed a clustering approach to cluster training programs using feature vectors. The best optimization sequence for a new program space lies at the cluster centroid [13, 14]. Park et al proposed a technique to select optimization technique to select optimization sequence selection using tournament predictors [16]. Suresh purini et al [2] used three different benchmark suites are to find best optimization sets.

These techniques used LLVM (low level virtual machine) test suite [14] where 61 optimization sequences are present. These programs are also called as microkernel programs. Microkernel programs have lesser execution times and consist of programs like searching and floating Point arithmetic. Iterative compilation techniques [13] are applied to find near
optimal optimization sequence sets. Finally testing the effectiveness of this approach using Mibench benchmark suite [12] and polybench benchmark suite programs shows an improvement in speedup.

**Prior Work**

In the existing system compiler optimization phase has default optimization sequences [2]. The sequences are applied on a program in sequential manner which optimizes different sections of the program. These sequences may wrongly communicate with each other which may affect the speed up of a program.

![Program compilation setup in LLVM.](image)

**Fig1: Program compilation setup in LLVM.**

LLVM [14] (low level virtual machine) compiler generates machine independent code. Clang [14] is one of the front-ends supported by llvm. Fig-2 describes the execution sequence of the program. Clang runs the programs and produces non-optimized Intermediate representation.

In the next phase of compilation clang run default optimization sequences like O1, O2 and O3 on the program to produce optimized IR. Finally optimized IR converted to machine code as bit code file.

**Proposed Work**

In the proposed System selection of best optimization sequences are done for poly-bench benchmark suite programs. Best set of optimization sequences are selected based on different program classes.

It uses the sequence selection algorithm to reduce the optimization sequence set so, that for every program class there exist at least one optimization sequence set.

By using the clustering algorithm it groups the similar sequences set. So, this approach reduces the execution time of a program by select the best optimization sequence thus increases the speedup of a program.
Above figure describes the proposed approach using LLVM compiler execution sequence. Optimized IR produced by clang using default optimization sequences wrongly interact with each other. These kinds of interactions cannot give efficient runtimes to programs. Sequence reduction algorithm is applied on the program space selects good optimization sequences. These optimization sequences are given as input to the optimization phase of the compiler to reduce runtime of the program.

**Sequence Selection pseudo code**

**Input:** program and optimization sequence  
**Output:** Best optimization sequence  
**Method:**

Begin  

bseq=optimization sequence;  

btime=program execution time using bseq;  

v=1;  

while v=1 do  

v=0;  

bseqlength=length of bseq;  

for i=1 to bseqlength do  

current sequence=remove the $i^{th}$ optimization from the bseq;  


current sequence time=execution time using of current sequence;

If current sequence time≤btime then

Update b time with current sequence time;

Update bseq with current sequence;

Change v to 1;

break from for loop

End of if

End of for

End of while

Return bseq.

End of main

Fig 3: Pseudocode for Sequence Selection

The above algorithm extracts best optimization sequences by giving the program and optimization sequence as input.

Experimental Analysis

Default optimization sequence(O1) and best optimization sequence(O2) applied on the poly benchmark program by using the VPO tool which gives the execution time for each program in the program space.

Result Analysis

Table 1.1 Runtime of default optimization sequence and best optimization sequence set over the poly benchmark suite.

<table>
<thead>
<tr>
<th>Benchmark Programs</th>
<th>Default sequence</th>
<th>optimization</th>
<th>Best sequence set</th>
<th>optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>correlative.c</td>
<td>0.0976</td>
<td></td>
<td>0.097</td>
<td></td>
</tr>
<tr>
<td>covarien.c</td>
<td>0.198</td>
<td></td>
<td>0.193</td>
<td></td>
</tr>
<tr>
<td>2mm.c</td>
<td>1.8</td>
<td></td>
<td>1.65</td>
<td></td>
</tr>
<tr>
<td>3mm.c</td>
<td>4.74</td>
<td></td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>atax.c</td>
<td>3.46</td>
<td></td>
<td>3.2</td>
<td></td>
</tr>
<tr>
<td>cholesky.c</td>
<td>5.36</td>
<td></td>
<td>5.1</td>
<td></td>
</tr>
<tr>
<td>doitgen.c</td>
<td>2.15</td>
<td></td>
<td>1.5</td>
<td></td>
</tr>
<tr>
<td>gemm.c</td>
<td>2.65</td>
<td></td>
<td>2.35</td>
<td></td>
</tr>
<tr>
<td>gemvar.c</td>
<td>6.54</td>
<td></td>
<td>6.14</td>
<td></td>
</tr>
<tr>
<td>gesummv.c</td>
<td>4.34</td>
<td></td>
<td>3.29</td>
<td></td>
</tr>
</tbody>
</table>
The following graph illustrates the comparison between the default and best optimization sequence set runtime over the polybenchmark suite programs.

![Graphical Representation of default and best optimization sequence sets execution time over the polybenchmark programs.](image)

**Fig 3.** Graphical Representation of default and best optimization sequence sets execution time over the polybenchmark programs.

The above graph illustrates the compilation time of each polybenchmark program using the default optimization sequence is less than the compilation time of each polybenchmark program using the best optimization sequence. From the graph we can say that best optimization sequence gives better performance than the default optimization sequence.

**Table 1.2 Speedup for default and best optimization sequence of polybenchmark programs**

<table>
<thead>
<tr>
<th>Benchmark Programs</th>
<th>Speedup of Default Optimization Sequence</th>
<th>Speedup of best Optimization Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>correlative.c</td>
<td>1.006</td>
<td>1.006</td>
</tr>
<tr>
<td>covarien.c</td>
<td>1.009</td>
<td>1.025</td>
</tr>
<tr>
<td>2mm.c</td>
<td>0.74</td>
<td>1.09</td>
</tr>
<tr>
<td>3mm.c</td>
<td>0.89</td>
<td>1.053</td>
</tr>
<tr>
<td>atax.c</td>
<td>0.095</td>
<td>1.0812</td>
</tr>
<tr>
<td>cholesky.c</td>
<td>1</td>
<td>1.05</td>
</tr>
<tr>
<td>doitgen.c</td>
<td>1.35</td>
<td>1.433</td>
</tr>
<tr>
<td>gemm.c</td>
<td>1.015</td>
<td>1.1276</td>
</tr>
<tr>
<td>gemvar.c</td>
<td>1.019</td>
<td>1.065</td>
</tr>
<tr>
<td>gesummv.c</td>
<td>1.102</td>
<td>1.319</td>
</tr>
</tbody>
</table>
Table 1.2 demonstrates the speed up of each benchmark program. Speed up of default optimization and best optimization sequence of each benchmark program has been plotted on the graph. Speed up is calculated as follows.

\[
\text{Speedup} = \frac{\text{default optimization}}{\text{new optimization}}
\]

(Fig 4: Speedup for default and best optimization sequence over polybenchmark programs.)

The above graph illustrates the speedup of each benchmark programs using the default optimization sequence is less than the speedup of each benchmark programs using the best optimization sequence. From the graph we say that performance of the best optimization sequence is improved over the default optimization sequence.

**Table 1.3. Percentage improved for benchmark programs**

<table>
<thead>
<tr>
<th>Benchmark program</th>
<th>Percentage improved</th>
</tr>
</thead>
<tbody>
<tr>
<td>correlative.c</td>
<td>6</td>
</tr>
<tr>
<td>covarian.c</td>
<td>2.5</td>
</tr>
<tr>
<td>2mm.c</td>
<td>9</td>
</tr>
<tr>
<td>3mm.c</td>
<td>5.3</td>
</tr>
<tr>
<td>atax.c</td>
<td>8.12</td>
</tr>
<tr>
<td>cholesky.c</td>
<td>5</td>
</tr>
<tr>
<td>doitgen.c</td>
<td>43.3</td>
</tr>
<tr>
<td>gemm.c</td>
<td>12.76</td>
</tr>
<tr>
<td>gemvar.c</td>
<td>6.5</td>
</tr>
<tr>
<td>gesummv.c</td>
<td>31.9</td>
</tr>
</tbody>
</table>
Fig 4: Performance improvement of best optimization sequence over the default optimization sequence.

\[
\text{Percentage improved} = (\text{speedup} - 1) \times 100
\]  

(2)

The above graph illustrates the performance improvement of benchmark program of best optimization sequence over the default optimization.

**Conclusion:**

In this paper we proposed a practical approach to solve the false interaction between optimization sequences. This method is quite different from the iterative compilation and machine learning-based prediction techniques. The idea is to filter the infinitely large optimization sequence on a program space using sequence selection algorithm. Selection of good optimization sequences set is done in very fast on a particular program space. Then given a new program we can try all the sequences from the good sequences set and choose the best sequence.

**References:**


**Corresponding Author:**

Ramani S*,

Email: ramani.s@vit.ac.in