Evaluating Heuristic Optimization Phase Order Search Algorithms

by

Prasad A. Kulkarni
David B. Whalley  Gary S. Tyson  Jack W. Davidson

Computer Science Department, Florida State University, Tallahassee, Florida

Computer Science Department, University of Virginia, Charlottesville, Virginia

2007 International Symposium on Code Generation and Optimization (CGO)
Compiler Optimizations

• Optimization phases require enabling conditions
  – need specific patterns in the code
  – many also need available registers

• Phases interact with each other

• Applying optimizations in different orders generates different code
Phase Ordering Problem

- To find an ordering of optimization phases that produces optimal code with respect to possible phase orderings
- Evaluating each sequence involves compiling, assembling, linking, execution and verifying results
- Best optimization phase ordering depends on
  - source application
  - target platform
  - implementation of optimization phases
- Long standing problem in compiler optimization!!
Addressing Phase Ordering

• Exhaustive phase order space evaluation [CGO ’06, LCTES ’06]
  – possible for most functions
  – long search times for larger functions

• Heuristic approaches
  – commonly employed, extensively studied
  – allow faster searches
  – no guarantees on solution quality
Survey of Heuristic Algorithms

• Cost and performance comparison
  – with optimal
  – with other heuristic searches
• Analyze phase order space properties
  – sequence length
  – leaf sequences
• Improving heuristic search algorithms
  – propose new algorithms
Outline

• Experimental setup
• Local search techniques
  – distribution of local minima
  – local search algorithms
• Exploiting properties of leaf sequences
  – genetic search algorithm
• Conclusions
Outline

• Experimental setup
• Local search techniques
  – distribution of local minima
  – local search algorithms
• Exploiting properties of leaf sequences
  – genetic search algorithm
• Conclusions
Experimental Framework

• We used the VPO compilation system
  – established compiler framework, started development in 1988
  – comparable performance to gcc –O2
• VPO performs all transformations on a single representation (RTLs), so it is possible to perform most phases in an arbitrary order
• Experiments use all the 15 re-orderable optimization phases in VPO
• Target architecture was the StrongARM SA-100 processor
# VPO Optimization Phases

<table>
<thead>
<tr>
<th>ID</th>
<th>Optimization Phase</th>
<th>ID</th>
<th>Optimization Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>branch chaining</td>
<td>l</td>
<td>loop transformations</td>
</tr>
<tr>
<td>c</td>
<td>common subexpr. elim.</td>
<td>n</td>
<td>code abstraction</td>
</tr>
<tr>
<td>d</td>
<td>remv. unreachable code</td>
<td>o</td>
<td>eval. order determin.</td>
</tr>
<tr>
<td>g</td>
<td>loop unrolling</td>
<td>q</td>
<td>strength reduction</td>
</tr>
<tr>
<td>h</td>
<td>dead assignment elim.</td>
<td>r</td>
<td>reverse branches</td>
</tr>
<tr>
<td>i</td>
<td>block reordering</td>
<td>s</td>
<td>instruction selection</td>
</tr>
<tr>
<td>j</td>
<td>minimize loop jumps</td>
<td>u</td>
<td>remv. useless jumps</td>
</tr>
<tr>
<td>k</td>
<td>register allocation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Benchmarks

- **12 MiBench benchmarks; 88 functions**

<table>
<thead>
<tr>
<th>Category</th>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>bitcount</td>
<td>test processor bit manipulation abilities</td>
</tr>
<tr>
<td></td>
<td>qsort</td>
<td>sort strings using quicksort sorting algorithm</td>
</tr>
<tr>
<td>network</td>
<td>dijkstra</td>
<td>Dijkstra’s shortest path algorithm</td>
</tr>
<tr>
<td></td>
<td>patricia</td>
<td>construct patricia trie for IP traffic</td>
</tr>
<tr>
<td>telecomm</td>
<td>fft</td>
<td>fast fourier transform</td>
</tr>
<tr>
<td></td>
<td>adpcm</td>
<td>compress 16-bit linear PCM samples to 4-bit</td>
</tr>
<tr>
<td>consumer</td>
<td>jpeg</td>
<td>image compression and decompression</td>
</tr>
<tr>
<td></td>
<td>tiff2bw</td>
<td>convert color .tiff image to b&amp;w image</td>
</tr>
<tr>
<td>security</td>
<td>sha</td>
<td>secure hash algorithm</td>
</tr>
<tr>
<td></td>
<td>blowfish</td>
<td>symmetric block cipher with variable length key</td>
</tr>
<tr>
<td>office</td>
<td>stringsearch</td>
<td>searches for given words in phrases</td>
</tr>
<tr>
<td></td>
<td>ispell</td>
<td>fast spelling checker</td>
</tr>
</tbody>
</table>
Terminology

- **Active** phase – an optimization phase that modifies the function representation
- **Dormant** phase – a phase that is unable to find any opportunity to change the function
- **Function instance** – any semantically, syntactically, and functionally correct representation of the source function (that can be produced by our compiler)
Terminology (cont…)

- **Attempted sequence** – phase sequence comprising of both active and dormant phases
- **Active sequence** – phase sequence only comprising active phases
- **Batch sequence** – active sequence applied by the default *(batch)* compiler
Setup for Analyzing Search Algorithms

• Exhaustively evaluate optimization phase order space
  – represent phase order space as DAG
• For each search algorithm
  – use algorithm to generate next optimization phase sequence
  – lookup performance in DAG
Phase Order Search Space DAG

- Performance evaluation of each phase order is traversal in the DAG
  - $a\cdot b\cdot d = 52$
Outline

• Experimental setup
• Local search techniques
  – distribution of local minima
  – local search algorithms
• Exploiting properties of leaf sequences
  – genetic search algorithm
• Conclusions
Local Search Techniques

• Consecutive sequences differ in only one position

• if $m$ phases and a sequence length of $n$, then will have $n(m-1)$ neighbors

<table>
<thead>
<tr>
<th>bseq</th>
<th>neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b c a a a a a a a</td>
</tr>
<tr>
<td>b</td>
<td>b b a c b b b b</td>
</tr>
<tr>
<td>c</td>
<td>c c c c a b c c</td>
</tr>
<tr>
<td>a</td>
<td>a a a a a a b c</td>
</tr>
<tr>
<td>a</td>
<td>a a a a a a b c</td>
</tr>
</tbody>
</table>
Local Search Space Properties

- Analyze local search space to
  - study distribution of local and global minima
  - study importance of sequence length

up to 100 attempts to generate seq. of length n

is new sequence?

get perf. of seq. & (m-1)n neighbors
record if node is minima
mark node as seen

exit

Y

N
Distribution of Minima

- % (num. minima / total samples)
- % (num global minima / total minima)

Diagram: Graph showing the distribution of minima as a function of the multiple of batch sequence length.
Hill Climbing

• Steepest decent
  – compare all successors of base
  – exit on local minima

randomly generate sequence of length n

get perf. of seq. & (m-1)n neighbors

is seq. minima? N Y

record local minima

select best neighbor as new base seq.
Hill Climbing Results (cont...)
Local Search Conclusions

• Phase order space consists of few minima, but significant percentage of local minima can be optimal

• Selecting appropriate sequence length is important
  – smaller length results in bad performance
  – larger length is expensive
Outline

• Experimental setup
• Local search techniques
  – distribution of local minima
  – local search algorithms
• Exploiting properties of leaf sequences
  – genetic search algorithm
• Conclusions
Leaf Sequence Properties

• *Leaf function instances* are generated when no additional phases can be successfully applied
  – sequences leading to leaf function instances are *leaf sequences*

• Leaf sequences result in *good* performance
  – at least one leaf instance represents an *optimal* phase ordering for over 86% of functions
  – significant percentage of leaf instances among optimal
Focusing on Leaf Sequences

• Modify phase order search algorithms to only produce leaf sequences
  – no need to guess appropriate sequence length
  – likely to result in optimal or close to optimal performance
  – leaf function instances comprise only 4.2% of the total instances
Genetic Algorithm

- A biased sampling search method
  - evolves solutions by merging parts of different solutions
Modified Genetic Algorithm

- Only generate leaf sequences
Genetic Algorithm – Performance

% Performance from Optimal

Multiple of Batch Sequence Length

- All Sequences
- Leaf Sequences
Genetic Algorithm – Cost

![Graph showing the relationship between the number of generations and the multiple of batch sequence length for both all sequences and leaf sequences. The graph illustrates a decrease in the number of generations as the multiple of batch sequence length increases.]
Leaf Search Conclusion

• Benefits of restricting searches to leaf sequences
  – no need for apriori knowledge of appropriate sequence length
  – near-optimal performance
  – low cost
Outline

• Experimental setup
• Local search techniques
  – distribution of local minima
  – local search algorithms
• Exploiting properties of leaf sequences
  – genetic search algorithm
• Conclusions
Conclusions

• First study to compare heuristic search solutions with optimal orderings
• Analyzed properties of phase order search space
  – few local and global minima
• Illustrated importance of choosing the appropriate sequence length
• Demonstrated importance of leaf sequences
  – achieve near-optimal performance at low cost
Questions ?
Simulated Annealing

• A worse solution is accepted with
  \[ \text{prob} = \exp (-\delta f/T) \]
  – \( \delta f \rightarrow \text{diff. in perf.} \)
  – \( T \rightarrow \text{current temp.} \)

• Annealing schedule
  – initial temperature
  – cooling schedule

\[ \text{randomly generate sequence of length } n \]
get perf. of seq. & (m-1)n neighbors

is seq. minima?

\[ \text{select best with prob } p? \]
select best neighbor as new base seq.

record local minima

\[ \text{select with best with prob } p? \]
Simulated Annealing (cont...)

- Simulated annealing parameters
  - sequence length → 1.5 times batch length
  - initial temperature → 0.5 to 0.95
  - cooling schedule → 0.5 to 0.95, steps 0.5

- Experimental results
  - perf. 0.15% from optimal, std. dev. of 0.13%
  - avg. perf. 15.95% worse, std. dev. of 0.55%
  - 41.06% iter. reach optimal, std. dev. of 0.81%
Random Algorithm

- Random sampling used for search spaces that are discrete and sparse

Randomly initialize (leaf) sequence → Lookup performance → No improve > 100? → Y: Output best sequence found & exit
                                     N: Record best performance → Randomly initialize (leaf) sequence
Random Algorithm – Performance

% Performance from Optimal

Multiple of Batch Sequence Length

- All Sequences
- Leaf Sequences
Random Algorithm – Cost

![Random Algorithm – Cost Graph]

- **All Sequences**
- **Leaf Sequences**

Number of Attempt

Multiple of Batch Sequence Length

180
170
160
150
140
130
120
110
100

1
1.75
2.5
3.25
4
4.75
5.5
6.25
7
7.75
8.5
9.25
10
10.75