Introduction

• Phase ordering problem
  - Traditional compilers have a fixed order in which optimization phases are applied.
  - This problem can be more severe when generating code for embedded applications.
  - VISTA allows the user to finely control both the order and scope of applying optimizations.
Introduction (cont...)

• Enhancing VISTA to make it more proficient at finding effective optimization sequences
  – Getting program performance measures anytime
  – Performance driven *interactive* code tuning
  – High level language like constructs to specify optimization phase orders
  – Performance driven *automatic* code tuning
Outline of the Talk

- Overview of VISTA
- Getting performance measures in VISTA
- Support for interactive code tuning
- Support for automatic code tuning
- Experimental results
- Future work
- Conclusions
Overview of VISTA
Getting Performance Measures

Compiler (VPO)

Send Static and Dynamic Frequency Measures For Current Function

Viewer

Request for Measurements

Save Current Program State

Apply Required Transformations

Restore Previous Program State

EASE

Instrument Code

Generate Instrumented Assembly

Link and Execute Assembly File

Last Function

Yes

No

Read Next Function
Interactive Code Tuning

- VISTA provides the user with performance measures during interactive compilation.
- VISTA currently provides two types of performance counts:
  - *Static counts* – a count of the number of static instructions in that function
  - *Dynamic counts* – a count of the number of instructions executed during a particular run of the program
- VISTA also displays the execution frequency of each basic block.
Interactive Code Tuning (cont...)

- VISTA provides two options for getting measures interactively.
  - Get frequency measures
  - Start / Stop measurements
Get Frequency Measures
Interactive Code Tuning (cont...)

- Control Statements in VISTA
  - High-level programming language like constructs are used in VISTA to conditionally invoke an optimization phase.
    - if-changes-else
    - if-changes-then-else
    - do-while-changes
    - while-changes-do
Automatic Code Tuning

• The previous approach requires user knowledge, intuition and effort to guide the code improvement process.

• We provided two new constructs in VISTA to support automatic code tuning
  – select best sequence
  – select best combination
Select Best Sequence

• The user selects two or more different optimization sequences.

• Each sequence is evaluated by the compiler for its performance.

• The user can specify weights for static and dynamic counts to determine the overall improvement.

• The best performing sequence is found and re-applied by the compiler.
Select Best Combination

• The user specifies a set of optimization phases.
• The compiler tries to determine the best ordering of this sequence of phases.
• The compiler forms different combinations of phases.
• Each is evaluated for performance, depending on weights specified by the user.
• Only the best performing sequence is re-applied.
Select Best Combination (cont...) 

- The compiler finds the next combination to evaluate based on the search option specified by the user.

- Search options
  - Exhaustive search – All possible combinations are attempted by the compiler
  - Biased sampling search – Compiler uses a genetic algorithm to probe the search space for an effective sequence
  - Permutation Search – Compiler attempts to evaluate all permutations of the specified length
Genetic Algorithms

- These are search algorithms designed to mimic the process of natural selection and evolution in nature.
- Some genetic algorithm terms
  - Chromosome – optimization sequence
  - Gene – individual optimization phase in a sequence
  - Population – set of chromosomes
  - Fitness value – performance of that optimization sequence
  - Crossover – combination of sequences to form new sequences
  - Mutation – individual phases in a sequence are replaced
  - Generation – time step for evaluation of sequences in one population and formation of the next population
Genetic Algorithm Used

- Initialization of first population
  - The first population of optimization sequences is randomly generated
Genetic Algorithm Used (cont...)

- The performance of each sequence in the population is evaluated.
- The chromosomes are sorted based on performance.
- The population is divided into two halves.
- Some chromosomes from the poorly performing half are deleted.
- The vacancies are filled using the crossover and mutation operation.
Genetic Algorithm Used (cont...)

- Crossover operation
  
  - upper half of the first chromosome is combined with lower half of the second and vice-versa.
Genetic Algorithm Used (cont...)

• The chromosomes are subjected to mutation.
• The best performing chromosome over all the generations is maintained.
Experimental Results

- A set of experiments were conducted to illustrate the effectiveness of using VISTA's biased sampling search.
- The experiments were conducted on a set of mibench programs.
- The target architecture for the experiments was the SPARC.
- The genetic algorithm was used to find the best sequence among 14 phases between register assignment and fix entry exit.
- The sequence length was set to 1.25 times the length of sequence applied during batch compilation.
Experimental Results (cont...)  

• Interactive compilation measures
  
  – An attempt was made to find an optimization sequence giving equal or better performance than that given by the batch compiler.
  
  – Genetic algorithm was used to probe the search space.
  
  – The population size was fixed at 20.
  
  – The algorithm was repeated for 100 generations.
  
  – Results were obtained for 3 different criteria, static count only, dynamic count only and 50% for each factor.
Overall Effect on Dynamic Instruction Count

- Non-Iterative Batch Compilation
- Iterative Batch Compilation
- Optimizing for 50% Dynamic
- Optimizing for 100% Dynamic

The graph illustrates the overall effect on dynamic instruction count for various benchmarks. The x-axis represents different benchmarks, while the y-axis shows the instruction count ratio. The bars indicate the comparison between different compilation and optimization strategies.
Future Work

• Obtaining measurements on a real embedded systems architecture
• Getting a more accurate measure of the dynamic performance
• Study the effect of varying the parameters in the genetic algorithm
• Study the result of performing genetic algorithm searches on sets of basic blocks in a function
Conclusion

• We have developed an interactive compilation system that automatically provides performance feedback information.

• Structured constructs are provided for specifying optimization sequences interactively.

• Constructs are provided to automatically select optimization phase sequences.

• Experiments were performed to illustrate the effectiveness of using a genetic algorithm to search for effective optimization sequences.