INCREASING EFFICIENCY AND PORTABILITY OF cTuning, A MACHINE LEARNING BASED SELF-TUNING TOOL

Presented By:
Santosh Sarangkar

Mentor:
Davide Del Vento
NCAR, Boulder, CO

*Note: Few slides are adopted from Dr. Apan Qasem
Roadmap

- Motivation
- Compilation Techniques
- Basic Idea behind self-tuning
- Two approached
  - Iterative Compilation (IC)
    - What is the biggest challenge using IC
  - Machine Learning Based
- Few example program features
- My work at NCAR
- Results with iterative compilation
- Results with ML approach
- Conclusion
- Future work
Motivation

- Dramatic change in Computer Architecture
- New architecture with new features
  - It’s a source of major concern for the HPC community
  - Today we have multi-core microprocessors with Multiple Levels of Cache
  - Greater computing capabilities
- Traditional compiler techniques are less effective because they rely on static information
  - E.g.: -O3 (GCC), -fast (Intel), -O3 (Open64) etc.
  - These are statically chosen by the compiler developers, and always the same for a given architecture
Scientific American, Feb 2005. Courtesy: Markus Puschel, CMU
Compilation Technique

• Compile the program with particular set of optimizations, which can exploit the target features
  • E.g gcc -flag1 -flag2 *.c

• Problem with the compilation technique:
• Finding a best set of optimizations is a NP problem
• Dependency
  • Apply -flag1: Performance Increases
  • Apply -flag2: Performance Increases
  • Apply –flag1 & -flag2 together: Performance Degrades
• Also, if we change the order of A and B, we get different performance
  • E.g. gcc –c -flag1 -flag2 test.c Performance: x
  • gcc –c -flag2 -flag1 test.c Performance: Y
  • This is called phase ordering problem
• Solution:
  • We can use self-tuning to address this problem
Basic Idea Behind Self-tuning

- **Self-tuning**
  - Self-tuning is nothing but automatically optimizing the compiler with a specific combination of optimizations to yield close to peak performance on current microprocessor-based platforms.
  - Unlike "-O3 (GCC), -fast (Intel), -O3 (Open64)", self-tuning finds/predicts a set of compilation flags that are individually chosen for any given source code, potentially giving much better results.

- **Goals of Self-tuning**
  - Reduce the tuning time (Make the compiler more efficient)
  - Speed up
  - Minimize the Code Size
Two Approaches in Self-Tuning

- **Iterative Self-Tuning**
  - Recompile the program with different set of transformations and find out a best set
  - Need to compile each program many times
  - I have worked on this approach at NCAR

- **Machine Learning Based self-Tuning**
  - Use previous knowledge of programs gathered with iterative self-tuning
  - Need to compile the program only once
  - Bill Petzke worked on this
Iterative Self-Tuning

Compiler

Transformation Tool

Performance Measurement Tools

Measure and Analyze
Biggest Challenge with Iterative Self-Tuning

- **Large search space**
  - Many transformations
    - Approximately 100 in GCC, number of points $2^{100}$
  
- Interaction between transformations
• A collaborative open source project
• It uses ML algorithms to match new program features with the programs in the database
• Uses iterative compilation: Random Search, to train the database(For training)
• Program Features(56)
Machine Learning Based Self-Tuning

- Program 1
- Program n
- New Program

1. Extract Program Features (56 Feats.)
2. Iterative Compilation

Database

- Training Set
- Testing

Explain the process flow and the role of machine learning in self-tuning.
Few Example Program Features:

- Number of basic blocks in the method
- Number of instructions in the method
- Number of unconditional branches
- Number of static/extern variables that are pointers in the method
- .............etc.
My Work at NCAR

- Installation of cTuning tool
  - Took 5 weeks to make it work, as it is so complex
  - Tool is written in 5 languages
- Plugged in PGI compiler into the tool
- Plugged in the benchmarks: SPEC, PARSEC, HPCC
- Trained the database
  - Benchmarks: cBench(10 apps), SPEC(4 apps), Parsec(3 apps), HPCC(3 apps).
  - 1000 iterative compilation for each apps
  - Performance Improvement using iterative compilation
- Used iterative compilation to test NCAR water model
- Used ML algorithm: KNN, to tune NCAR HPC shallow water model on lynx (shallow3.f)
Iterative Compilation with PGI on LYNX

% Performance Imp. over -fast

- automative-bitcnt: 6
- automative-susan-c: 14
- office-stringsearch: 24
- telecom-CRC: 6
- clustal: 32
- mcf: 77
- NCAR Shallow Water Model: 14

Speed up in %
ML Based self-tuning of NCAR HPC Shallow Water Model

% Performance Improvement over -Fast

- ML(KNN): 1.6
- Iterative Compilation: 14

Series 1
There is room for improving the performance over -fast, imaging having a 14% more money to buy a larger cluster!
Conclusion

• **Iterative compilation approach:**
  We are successful to get the performance improvement up to 76% over the highest level of optimizations: -fast

• **Drawback:** Tuning time is high (Not good for big applications)

• **Machine Learning based approach**
  • Less effective (in case of performance) as compare to iterative compilation
  • Less tuning time
Future Work

• Adding Hardware performance counters to increase the efficiency of the tool
• Plug-in another compilers like: LLVM, pathscale, Intel etc. to increase the portability of the framework
• Adding genetic algorithm in the iterative compilation might make the framework more efficient
• KNN is not the best algorithm to tackle this problem and so we need to plug-in SVM
Thank You!