Introduction

Scientists running high performance geophysical models want to achieve the fastest runtime possible for their software on any machine. For this goal, they usually select compilers’ default aggressive optimization flags, however this is often a suboptimal choice. In fact, the best matching set of optimization flags depends both on the underlying hardware and on the characteristics of the program. The fast change and rapid improvement in microprocessor technology, and diversity of program profiles make finding such a best set of optimization flags a challenging task. This problem is NP-Complete and thus it is not possible to find the general exact solution. However, an approximate solution may be found.

Recently, computer science researchers have applied machine learning algorithms on both static and dynamic profiling features of computer programs including hardware performance counters to achieve semi-optimal set of compiler optimization flags, which are often better than the default aggressive optimization such as –O3 for GCC and –fast for the PGI compiler.

Motivation

Execution time of LU kernel compiled with:
- gcc –O2 -ffast-math
- gcc –O3

Result:
- 20% improvement!

Contribution

In this project, we extended CTuning by adding new kernels, more relevant to atmospheric models, to the training database. We tested the framework on few simple models in use at NCAR: Shallow Water, EULAG and HD3D. We performed our experiments on Janus high performance computer system (at the university of Colorado Boulder).

Method

Objective: Self-Tuning (automatically generate a combination of compiler flags that will minimize Run-time)

Three steps:
- Iterative Compilation: repeatedly compile a training kernel with a randomly picked set of flags, and benchmarking the corresponding runtime, selecting the best set of flags that provides the fastest run-time.
- Training: Learn a prediction model based on the training data set
- K-Mean is used for similarity calculation
- Training kernels (CTuning kernels and Pollybench)
- Testing: (Apply the learned model on an un-seen program and suggest the optimal set of flags).

Results

Conclusions

- Practical use of self-tuning or adaptive methods to improve performance by compiler flag selection
- In some cases performance improvement can be made over maximum optimization level i.e. –O3 (gcc) or –fast (pgi)
- Machine Learning methods and Milepost GCC framework are promising and having it trained over appropriate set of features and diverse range of kernels, Milepost GCC can improve the execution time of unseen programs by choosing optimal set of compiler flags.
- Future Opportunities
- Exploring feature space in an attempt to:
  - prune the feature space
- Extract better features or combination of features
- Using different machine learning algorithms/similarly measurement

Bibliography