Tuning hardened compiler optimizations for rapidly evolving hardware makes porting an optimizing compiler for each new platform extremely challenging. Our radical approach is to develop a modular, extensible, self-tuning compiler that automatically learns the best optimization heuristics based on the platform. In this poster we describe MILEPOST GCC, a machine-learning-based compiler that automatically adjusts its optimization heuristics to improve the execution time, code size, or compilation time of specific programs on different architectures. Our preliminary experimental results show that it is possible to considerably reduce execution time of the MiBench benchmark suite on a range of platforms entirely automatically.

Current state-of-the-art compilers often fail to deliver best performance due to:

- hardened optimization heuristics (cost models) for rapidly evolving hardware (often impossible to fine-tune programs externally)
- interaction between optimizations
- large irregular optimization spaces
- difficult to add new transformations to already tuned optimization heuristics
- inability to reuse optimization knowledge among different programs and architectures
- lack of run-time information and inability to adapt to varying program and system behavior at run-time with low overhead

Need modular self-tuning compilers that can continuously and automatically learn how to optimize programs, and have an ability to make program adaptable at run-time for different behavior and constraints

Optimization spaces (set of all possible program transformations) are large, non-linear with many local minima

Finding a good solution may be long and non-trivial

Matmul, 2 transformations, search space = 2000
Swim, 3 transformations, search space = 10^16

Iterative compilation: learn program behavior across executions

High potential (O’Boyle, Cooper since 1998), but:
- slow
- the same dataset is used
- no run-time adaptation
- no optimization knowledge reuse

Solving these problems is non-trivial

Training: Gathering information about the structure of programs and record how they behave when compiled under different optimization settings to build machine learning models.

Deployment: ML model is able to predict good optimization strategies for a given set of program features and is built as a plugin within MILEPOST GCC.

The ICI provides opportunities for external control and examination of the compiler. Optimization settings at a fine-grained level, beyond the capabilities of command-line options or pragmas, can be managed through ICI plugins.

Modifications needed to enable GCC ICI:
- ic-framework.c: GCC plugin (dynamic library) invocation
- save-executed-passes.c: Plugin to monitor executed passes. It registers an event handler function executed_pass on an IC-Event called pass_execution.
- passes.c: Modifications in GCC Controller (Pass Manager) to enable manipulation with optimization passes.

Features:
- use ICI for adaptive libraries
- enable run-time adaptation and parallelization for static programs
- continue research on pass selection and reordering for reconfigurable processors and design space exploration
- improve and automate selection of static and dynamic (hardware counters) program features
- use ICI for adaptive libraries
- enable run-time adaptation and parallelization for static programs

For more information, visit the following links:
- MILEPOST GCC: http://www.milepost.eu
- MILEPOST Framework: http://unidapt.org
- Project news: http://www.milepost.eu
- Contact: grigori.fursin@inria.fr