MILEPOST GCC: machine learning based research compiler

Grigori Fursin
UNIDAPT group, INRIA Saclay, France
• Motivation: tackling the complexity of future computing systems

• MILEPOST project and MILEPOST framework

• GCC with Interactive Compilation Interface
  • Dynamic pass manipulation plugins
  • Selecting “good” GCC optimization flags and passes
  • Program static feature extractor plugin

• Continuous Collective Compilation framework to reuse optimization knowledge

• Experiments

• Future work, collaborations, public software and dissemination
Developing and tuning current compilers for rapidly evolving architectures is a tedious and time consuming process. Current state-of-the-art compilers and optimizers often fail to deliver best performance.

- **Hardwired 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**
Motivation

Developing and tuning current compilers for rapidly evolving architectures is a tedious and time consuming process. Current state-of-the-art compilers and optimizers often fail to deliver best performance.

- hardwired 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
## Motivation (GCC)

What can we do with current open-source compilers?

<table>
<thead>
<tr>
<th></th>
<th>SUIF/Octave</th>
<th>Open64</th>
<th>GCC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Architectures:</strong></td>
<td>Any</td>
<td>Limited</td>
<td>Many</td>
</tr>
<tr>
<td><strong>Transformations:</strong></td>
<td>Many</td>
<td>Many</td>
<td>Improving</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>Soon high-level loop optimizations</em></td>
</tr>
<tr>
<td><strong>Performance:</strong></td>
<td>-</td>
<td>Very good</td>
<td>Improving</td>
</tr>
<tr>
<td><strong>Modularity:</strong></td>
<td>Yes</td>
<td>No</td>
<td>Pass manager (!)</td>
</tr>
<tr>
<td><strong>Stability:</strong></td>
<td>So so</td>
<td>Ok</td>
<td>Good</td>
</tr>
<tr>
<td><strong>Future:</strong></td>
<td>Not supported</td>
<td>Unclear</td>
<td>Very promising</td>
</tr>
</tbody>
</table>

Grigori Fursin et al.        “MILEPOST GCC: Machine learning based research compiler”  GCC Summit’08, Canada     19th of June, 2008
**Motivation (GCC)**

**What can we do with current open-source compilers?**

<table>
<thead>
<tr>
<th></th>
<th>SUIF/Octave</th>
<th>Open64</th>
<th>GCC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Architectures:</strong></td>
<td>Any</td>
<td>Limited</td>
<td>Many</td>
</tr>
<tr>
<td><strong>Transformations:</strong></td>
<td>Many</td>
<td>Many</td>
<td>Improving Soon high-level loop optimizations</td>
</tr>
<tr>
<td><strong>Performance:</strong></td>
<td>-</td>
<td>Very good</td>
<td>Improving</td>
</tr>
<tr>
<td><strong>Modularity:</strong></td>
<td>Yes</td>
<td>No</td>
<td>Pass manager (!)</td>
</tr>
<tr>
<td><strong>Stability:</strong></td>
<td>So so</td>
<td>Ok</td>
<td>Good</td>
</tr>
<tr>
<td><strong>Future:</strong></td>
<td>Not supported</td>
<td>Unclear</td>
<td>Very promising</td>
</tr>
</tbody>
</table>

**Difficulties in GCC:**

- Retargeting/tuning to a new architecture
- Adding new transformations and re-tuning cost models
- Selecting “good” combination of passes on average and per program
- Finding compiler bugs
- Optimizing for different datasets (run-time adaptation)

Grigori Fursin et al.  "MILEPOST GCC: Machine learning based research compiler"  GCC Summit’08, Canada  19th of June, 2008
Objective is to develop compiler technology that can automatically learn how to best optimize programs for re-configurable heterogeneous embedded processors and dramatically reduce the time to market.

Partners:
INRIA, University of Edinburgh, IBM, ARC, CAPS Enterprise

Developed techniques and software are publicly available and hopefully will influence the future compiler developments.

Iterative compilation

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 = $10^{52}$
Iterative compilation

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 = $10^{52}$

Learn program behavior across executions

High potential but:
- slow
- no optimization knowledge reuse
- the same dataset is used
- no run-time adaptations
Iterative compilation

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 = $10^{52}$

Learn program behavior across executions

High potential but:
- slow
- no optimization knowledge reuse
- the same dataset is used
- no run-time adaptations

Solving these problems is non-trivial
**MILEPOST framework**

- **Training**
  - $Program_1$
  - ...
  - $Program_N$

**MILEPOST GCC**
(with ICI and ML routines)

**IC Plugins**
- Recording pass sequences
- Extracting static program features
MILEPOST framework

Grigori Fursin et al.  "MILEPOST GCC: Machine learning based research compiler"  GCC Summit'08, Canada  19th of June, 2008
MILEPOST framework

Training

Program₁

... 

Programₙ

Deployment

New program

MILEPOST GCC

MILEPOST GCC
(with ICI and ML routines)

IC Plugins

Recording pass sequences

Extracting static program features

Drivers for iterative compilation and model training

Global Optimization Database

Continuous Collective Compilation Framework

CCC

MILEPOST GCC

Extracting static program features

Selecting “good” passes

Predicting “good” passes to improve exec. time, code size and comp. time

Grigori Fursin et al. “MILEPOST GCC: Machine learning based research compiler” GCC Summit’08, Canada 19th of June, 2008
Interactive Compilation Interface

Hooks and plugins to “open up” compilers (started for Open64 in 2003, moved to GCC in 2005) to enable transparent analysis and optimizations
Interactive Compilation Interface

Hooks and plugins to “open up” compilers (started for Open64 in 2003, moved to GCC in 2005) to enable transparent analysis and optimizations
Interactive Compilation Interface v2

Grigori Fursin et al.        “MILEPOST GCC: Machine learning based research compiler”  GCC Summit'08, Canada  19th of June, 2008
Interactive Compilation Interface v2

GCC with ICI

Detect optimization flags

IC Event

GCC Controller (Pass Manager)

Pass 1

Pass N

... IC Event

Interactive Compilation Interface

ICI

Garcc Data Layer

AST, CFG, CF, etc

IC Data

Grigori Fursin et al. “MILEPOST GCC: Machine learning based research compiler” GCC Summit'08, Canada 19th of June, 2008
Interactive Compilation Interface v2

GCC with ICI

Detect optimization flags

IC Event

GCC Controller (Pass Manager)

IC Event

Pass_1

... Pass_N

IC Event

GCC Data Layer

AST, CFG, CF, etc

IC Data

ICI

Interactive Compilation Interface

IC Plugins

High-level scripting (java, python, etc)

< Dynamically linked shared libraries >

Selecting pass sequences

Extracting static program features

Grigori Fursin et al. “MILEPOST GCC: Machine learning based research compiler” GCC Summit’08, Canada 19th of June, 2008
static int
load_ici (char *dynlib_file)
{
    ...
    void *ICILib;
    bool error = 0;

    ICILib = dlopen(dynlib_file, RTLD_LAZY);
    error |= check_for_dLError();

    ici_start = (func) dlsym (ICILib, "start");
    error |= check_for_dLError();
    ici_stop = (func) dlsym (ICILib, "stop");
    error |= check_for_dLError();
    ...
}

#define EXECUTE_PASS (void)

void executed_pass (void)
{
    char *pass_name;
    char *func_name;

    func_name = (char *) ici_get_feature("function_name");
    pass_name = (char *) ici_get_parameter("pass_name");
    printf("%s %s\n", func_name, pass_name);
}

char start (void)
{
    ici_register_event("pass_execution",
                        &executed_pass);
}
Iterative compilation

-freorder-blocks -frerun-loop-opt -fno-sched-interblock -fsched-spec-load-dangerous -fsched-spec-load -fno-sched2-use-superblocks
-fno-tree-vec-loop-version

“Good” random set of compiler flags
Iterative compilation

-freorder-blocks -frerun-loop-opt -fsched-spec-load-dangerous -fsched-spec-load -fno-sched2-use-superblocks
-fno-tree-vec-loop-version

“Good” random set of compiler flags

fixupcfg, init_datastructures, all_optimizations, referenced_vars, reset_cc_flags, salias, ssa, alias, retslot, copyrename, ccp, fre, dce, forwprop, copyprop, mergephi, vrp, dce, dom, phicprop, phi, alias, tailr, profile, ch, cpxlower, sra, alias, copyrename, dom, phicprop, reassoc, dce, dse, alias, forwprop, phi, objsz, store_ccp, store_copyprop, fab, alias, criteq, pre, alias, sink, loop, loopinit, copyprop, lim, unsunswitch, scp, empty, record_bounds, ivcanon, cunroll, ivopts, loopdone, reassoc, vrp, dom, phicprop, cddce, dse, forwprop, phi, tailr, copyrename, uncprop, optimized, nrv, blocks, final_cleanup, warn_function_noreturn, free_datastructure, free_cfg_annotations, expand, rest_of_compilation, init_function, sibling, locators, initvals, unshared, vregs, jump, cse, gcse1, bypass, ce1, loop2, loop2_init, loop2_invariant, loop2_unswitch, loop2_done, cse2, life1, combine, ce2, regmove, split1, mode-sw, life2, lreg, greg, postreload, postreload_cse, gcse2, flow2, csa, peephole2, ce3, mreg, bbro, leaf_regs, sched2, stack, compute_alignments, compgotos, free_cfg, mach, elnotes, barriers, eh-anges, shorten, set_nothrow_function_flags, final, clean_state

Sequence of compiler passes for -O3

fixupcfg, init_datastructures, all_optimizations, referenced_vars, reset_cc_flags, salias, ssa, alias, retslot, ccp, fre, dce, forwprop, copyprop, mergephi, vrp, dce, phi, alias, profile, ch, cpxlower, sra, alias, reassoc, dce, alias, forwprop, phi, objsz, store_ccp, store_copyprop, fab, alias, criteq, pre, alias, sink, loop, loopinit, copyprop, unsunswitch, scp, empty, record_bounds, ivcanon, cunroll, loopdone, reassoc, vrp, cddce, forwprop, phi, optimized, nrv, blocks, final_cleanup, warn_function_noreturn, free_datastructure, free_cfg_annotations, expand, rest_of_compilation, init_function, sibling, locators, initvals, unshared, vregs, jump, cse, gcse1, bypass, ce1, loop2, loop2_init, loop2_invariant, loop2_unswitch, loop2_done, cse2, life1, combine, ce2, regmove, split1, mode-sw, life2, lreg, greg, postreload, postreload_cse, gcse2, flow2, csa, peephole2, ce3, mreg, bbro, leaf_regs, sched2, stack, compute_alignments, compgotos, free_cfg, mach, elnotes, barriers, eh-anges, shorten, set_nothrow_function_flags, final, clean_state

Sequence of GCC passes for the “good” set of compiler flags (ISSUE with UNROLLING, PEELING)
ICI transparent use

Examples: http://gcc-ici.sourceforge.net

One of the goals: no project modifications during optimizations

Record GCC passes:

```bash
export ICI_PLUGIN="$ICI_PLUGIN_HOME/save-executed-passes.so"
export ICI_USE=1
make
```

Select new GCC passes and orders (global or per function):

```bash
export ICI_PLUGIN="$ICI_PLUGIN_HOME/use-pass-order.so"
export ICI_PASSES_ALL=fixupcfg,init_datastructures,all_optimizations,referenced_vars,...,mach,eelnotes,barriers,eh-ranges,shorten,set_nothrow_function_flags,final,clean_state
export ICI_USE=1
make
```
Feature extraction

We can now add new passes that are not included into default optimization heuristic but called through ICI.

**Example: program static feature extractor (thanks to IBM Haifa)**

```bash
export XSB_DIR="…"
export ICI_PLUGIN="$ICI_PLUGINS_HOME/extract-program-static-features.so"
export ICI_PROG_FEAT_PASS=fre
export ICI_PROG_FEAT_EXT_TOOL="$ICI_PLUGINS_HOME/ml-feat-proc"
export ML_ST_FEAT_PROC="$ICI_PLUGINS_HOME/featlstn.P"
export ICI_USE=1

ft1 - Number of basic blocks in the method
ft20 - Number of conditional branches in the method
ft21 - Number of assignment instructions in the method
ft22 - Number of binary integer operations in the method
ft23 - Number of binary floating point operations in the method
ft24 - Number of instructions in the method
ft25 - Average of number of instructions in basic blocks
ft54 - Number of local variables that are pointers in the method
ft55 - Number of static/extern variables that are pointers in the method
```
Continuous Collective Compilation Framework

- Unify iterative experiments and optimization knowledge reuse
- Collect and analyze data from various partners in a global database:
  - COMPILERS, DATASETS, ENVIRONMENTS, OPT_FLAGS_GLOBAL, PLATFORMS
  - PROGRAMS, PROGRAM_FEATURES, PROGRAM_PASSES,
  - STATS_COMP_GLOBAL_FLAGS, STATS_EXEC_GLOBAL
- Support iterative compilation (flags & passes) with different strategies, transparent profiling using hardware counters collection using PAPI library
- During last 6 months around 2,000,000 executions on various platforms:
  - x86, x8664, IA64
  - TMS320C6713
  - ICT GODSON2
  - ARC 700
- Build machine learning models to improve GCC performance on average across all programs or for
Experiments

Generating training set to build model

Traditional iterative search:
500 random sequences of flags and associated passes (turned on or off)
Later “focused” search

Graph showing speedup for various benchmarks on different platforms:
- AMD - a cluster with 16 AMD Athlon 64 3700+ processors running at 2.4GHz
- IA32 - a cluster with 4 Intel Xeon processors running at 2.8GHz
- IA64 - a server with an Itanium2 processor running at 1.3GHz

Grigori Fursin et al. “MILEPOST GCC: Machine learning based research compiler” GCC Summit’08, Canada 19th of June, 2008
Experiments

Building and using model to predict optimizations

Use static or dynamic (hardware counters) program features to find similarities between programs to focus search for good optimizations

Similar to feedback directed optimizations, except we reuse “global optimization knowledge” and use program features to suggest good optimizations
Experiments

Evaluating model performance
(FPGA implementation of the ARC 725D processor)
Acknowledgments

Grigori Fursin
Cupertino Miranda
Olivier Temam
INRIA Saclay, France

Mircea Namolaru
Elad Yom-Tov
Ayal Zaks
Bilha Mendelson
IBM Haifa, Israel

Edwin Bonilla
John Thomson
Hugh Leather
Chris Williams
Michael O’Boyle

University of Edinburgh, UK

Phil Barnard, Elton Ashton
ARC International, UK

Sebastian Pop
AMD, USA

Eric Courtois, Francois Bodin
CAPS Enterprise, France
Conclusions and future work

• Just finished all the infrastructure that works and is stable – most parts are either released or will be released at the end of summer 2008 …

• Continue research on selecting good passes, build dependencies between passes (and find compiler bugs), learn sequences of good passes to improve program execution time, code size or compiler speed, studying interaction between optimizations, automatically selecting sets of optimizations for –Ox levels, evaluate hardware-software co-design, etc …

• Will add support to the ICI for all available cost models (for register allocation, scheduling, vectorization, etc) – looking for help and suggestions …

• Use the infrastructure to develop adaptive optimized libraries for different datasets and architectures using GCC …

• Optimize SPEC2006 on a number of architectures during summer 2008 …
Instead of catching up other compilers – go ahead

Feedback, support, collaboration ?!

Adding to mainline GCC?

Most tools are or will be available online:

http://www.milepost.eu

Contact: grigori.fursin@inria.fr (http://unidapt.org)