A COMPILER AGNOSTIC AND ARCHITECTURE AWARE PREDICTIVE MODELING FRAMEWORK FOR KERNELS

William Killian [1,2] Ian Karlin [1]
David Beckingsale [1] John Cavazos [2]
[1] Lawrence Livermore National Laboratory [2] University of Delaware

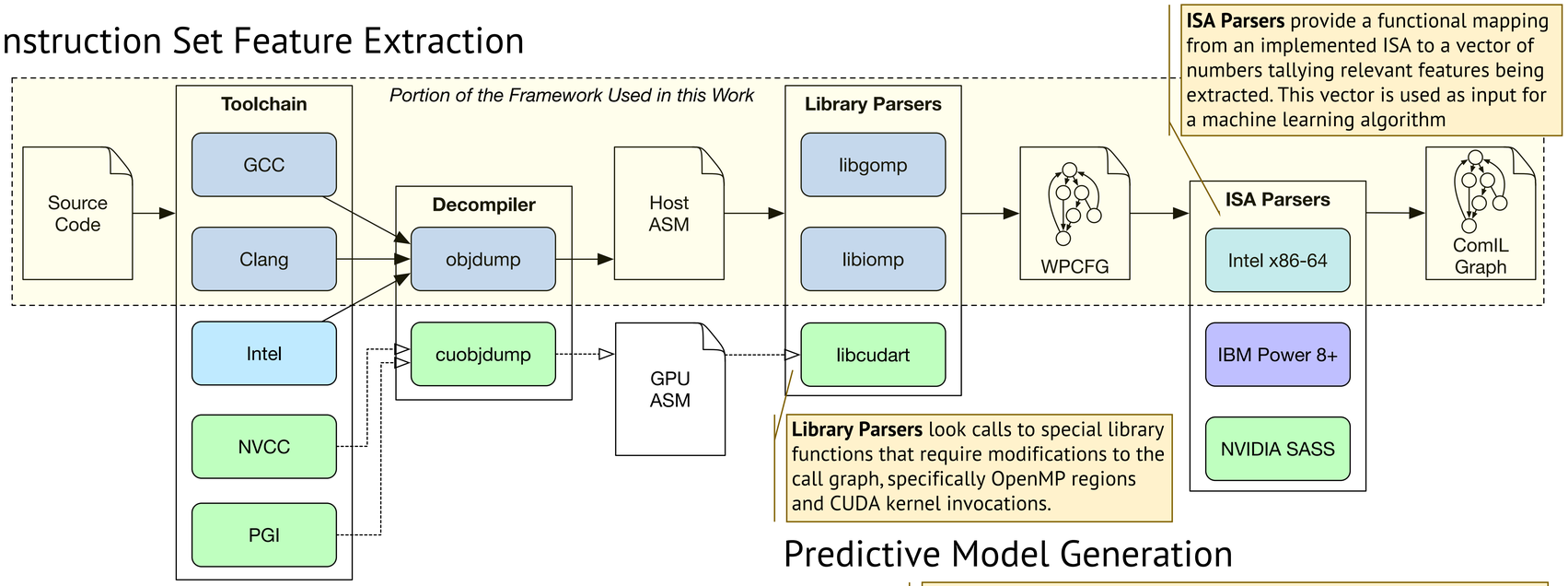
Problem Statement

- Predictive modeling becomes increasingly difficult as heterogeneous architectures are adopted and many compilers are available
- Application codes need to target multiple architectures, but it is not feasible to explore a complete optimization search space.
- A predictive modeling framework should be able to handle any binary targeting an architecture regardless of compiler.
- Optimizations and transformations applied can take one nested loop and generate thousands of variants, all of which may not be correct.

Experiment Setup

- PolyBench [1] (30) and ARES LoopSuite (4) kernels
- Used RAJA [2] as tuning framework for code transformations – some transformations applied could be unsafe (forced parallelization)
- Compiled each kernel version with GCC 7.1 and Clang 4.0
- Train models with all-but-one PolyBench kernel (cross validation)
- Evaluate with LoopSuite and the remaining PolyBench kernel
- Modeling Objectives: execution time and maximum absolute error

Instruction Set Feature Extraction



ISA Parsers provide a functional mapping from an implemented ISA to a vector of numbers tallying relevant features being extracted. This vector is used as input for a machine learning algorithm

Library Parsers look calls to special library functions that require modifications to the call graph, specifically OpenMP regions and CUDA kernel invocations.

Decouple the compiler-architecture pair problem by:

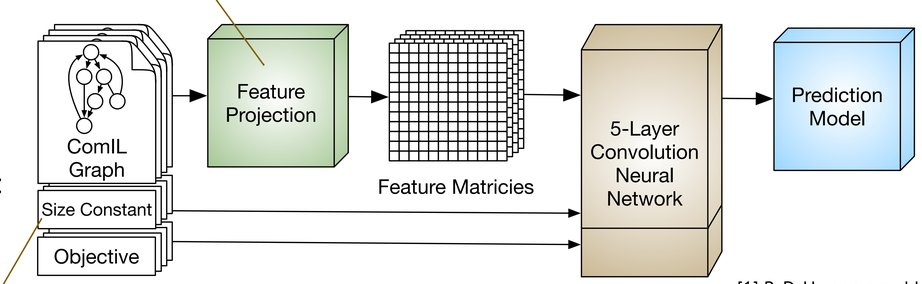
- Considering the final code emitted by the compiler after all optimizations are applied.
- Handling heterogeneous codes by specializing instruction feature mapping for different Instruction Set Architectures (ISA)

Map each instruction to a common instruction format for learning (ComIL), storing the ISA target as a feature as well as:

- **Data:** type (integer, floating point), size (in bytes), vector size
- **Memory:** # bytes read, # bytes stored, push/pop instructions
- **Control Flow:** call, return, jump, conditional jump
- **Computation:** add, sub, mul, div, bitwise, flags, special, cond

Predictive Model Generation

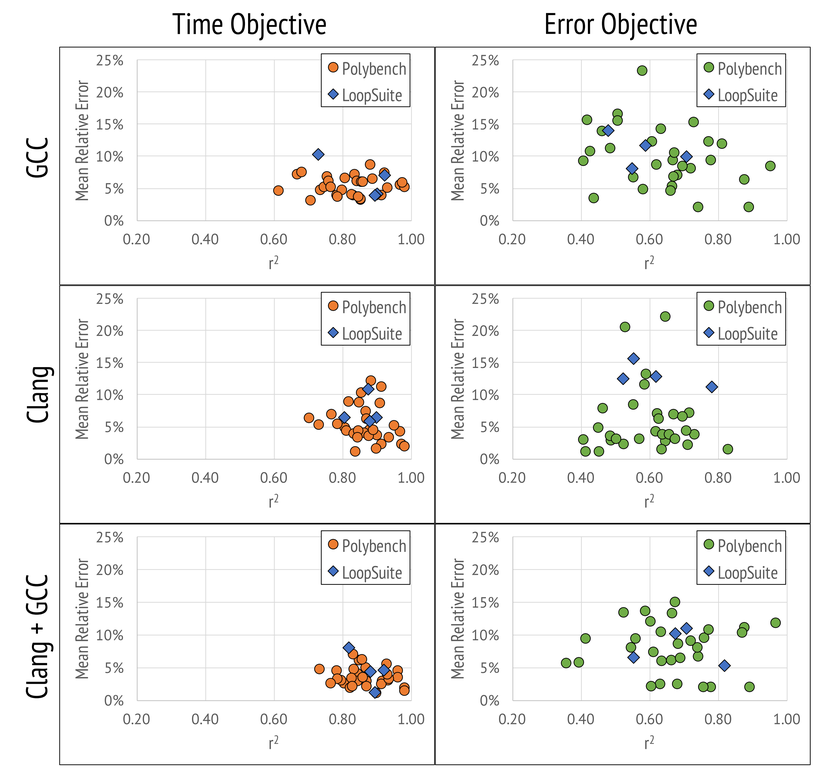
Feature Projection maps the feature space of a graph (which can be dynamic) to a fixed feature matrix. This uses eigenvector projection of the adjacency matrix to reduce and unify the feature space



The **Size Constant** is a user-provided value which represents a function of the program computation size. For this work it's the total number of times executing an innermost loop.

Evaluation

- Generate all models (90) `{gcc,clang,gcc+clang}x{PolyBench}`
- Process unseen program versions through each model
- Evaluate each model for error (mean relative error) and ordering by rank of the optimization speedup curve (r^2)



Conclusion and Future Work

- We show that we are able to decouple the compiler-architecture pair by constructing and evaluating models constructed from PolyBench kernels. Predicted objective outcomes show:
  - *Execution Time:* 4.7% average error rate and r^2 of 0.865
  - *Maximum Absolute Error:* 10% average error rate and r^2 of 0.624
- Extend work to additional target architectures and compilers
- Evaluate models with proxy applications CoMD [3] and KRIPKE [4]

[1] R. D. Hornung and J. A. Keasler, "*The RAJA Portability Layer: Overview and Status, Tech Report*", LLNL-TR-661403, Sep. 2014.
[2] L. N. Pouchet. "*PolyBench/C – the Polyhedral Benchmark Suite*", 2015 (https://sourceforge.net/projects/polybench/)
[3] M. A. Heroux, J. Mohd-Yusof, D. Richards, A. Herdman and J. Luitjens, "*Mantevo Suite 1.0*", R&D 100 Awards Submission, 2013
[4] A. J. Kunen, T. S. Bailey, P. N. Brown, "*KRIPKE - A Massively Parallel Transport Mini-App*", American Nuclear Society M&C, 2015

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344

LLNL-POST-73535