Source-based Matching and Rewriting

Guido Araujo

University of Campinas (Unicamp)
Computing Institute (IC)

Workshop • Mar 8th, 2024

Agenda

1. Goals
2. Background
3. Algorithm
4. Toolchain
5. Experiments

Goals

- ▷ Pattern-replacement description
- ▷ *Written in original language*
- ▷ Embedded in existing compilation tools
- ▷ Raising - rewrite complex patterns

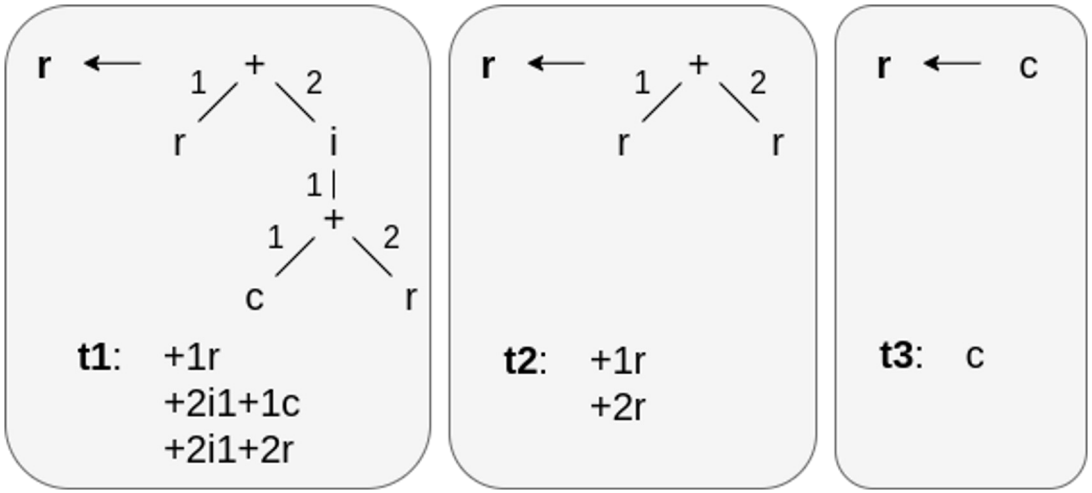
PAT Description

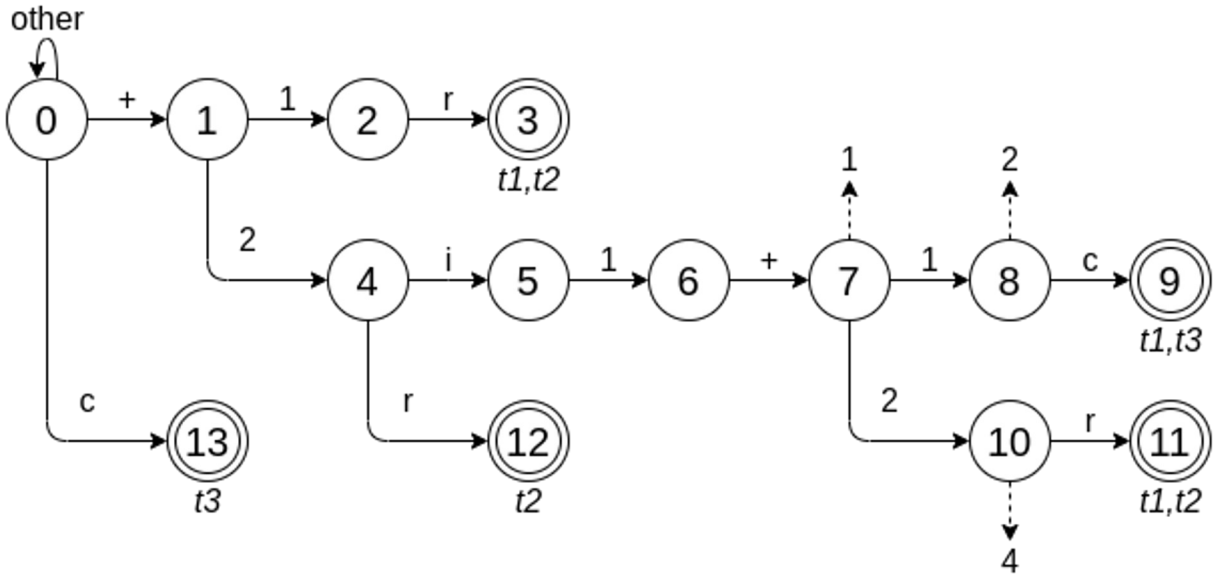
```
C {
  void dot(int N,float *x,int ix,float *y,int iy,float out){
    for (int i = 0; i  <  N;  i++)
      out += x[i*ix] * y[i*iy];
  }
} = {
  #include <cblas.h>
  void dot(int M,float *p,int ip,float *q,int iq,float out){
    out += cblas_sdot(M, p, ip, q, iq);
  }
}
```

Background - TWIG

- ▷ Pattern matching for instruction selection
- ▷ Converts trees to strings
- ▷ Use automatons to match strings
- ▷ Matching the string we match the trees
- ▷ Inspiration for the algorithm

Background - TWIG

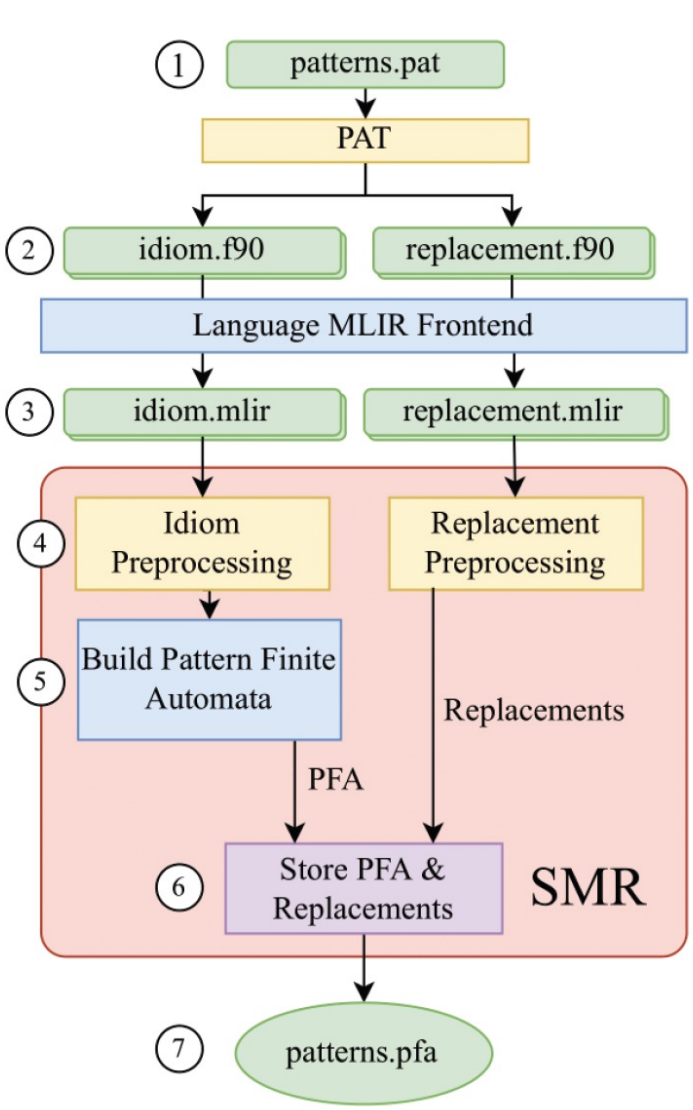


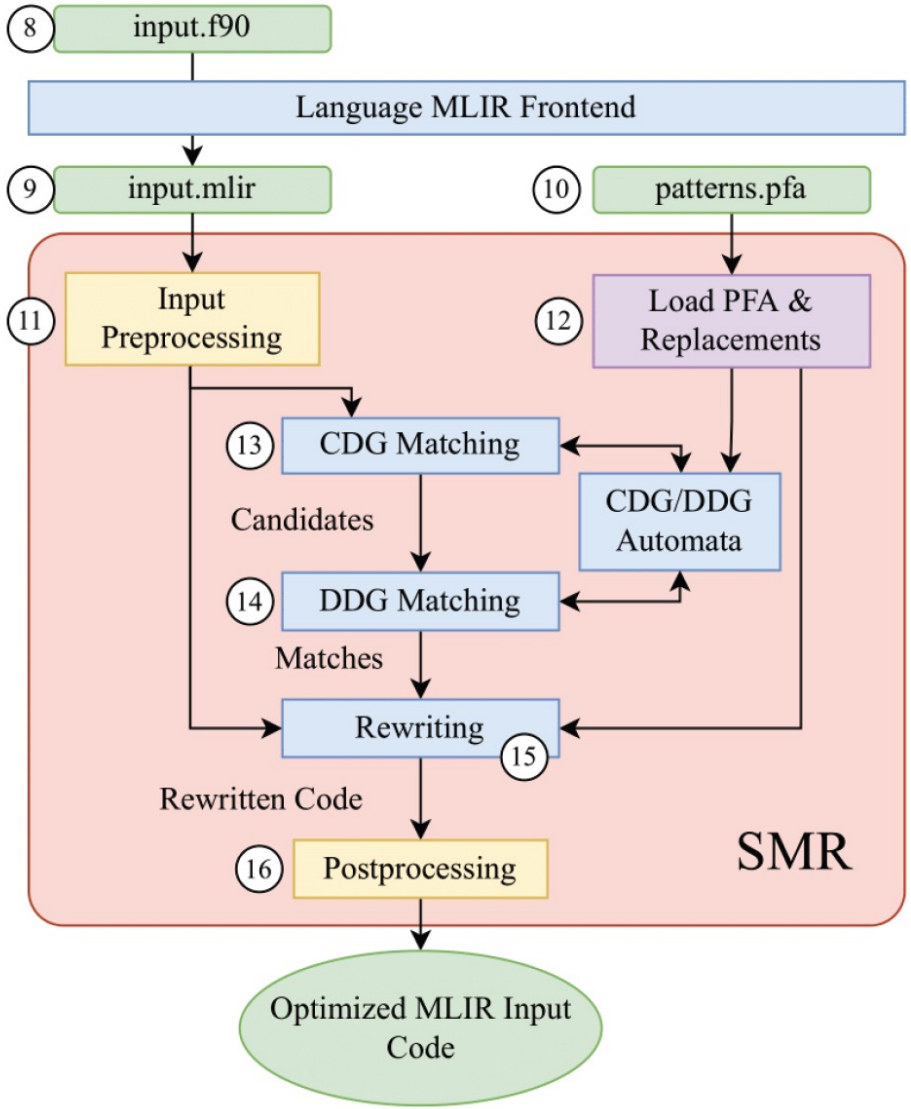


Algorithm - Matching

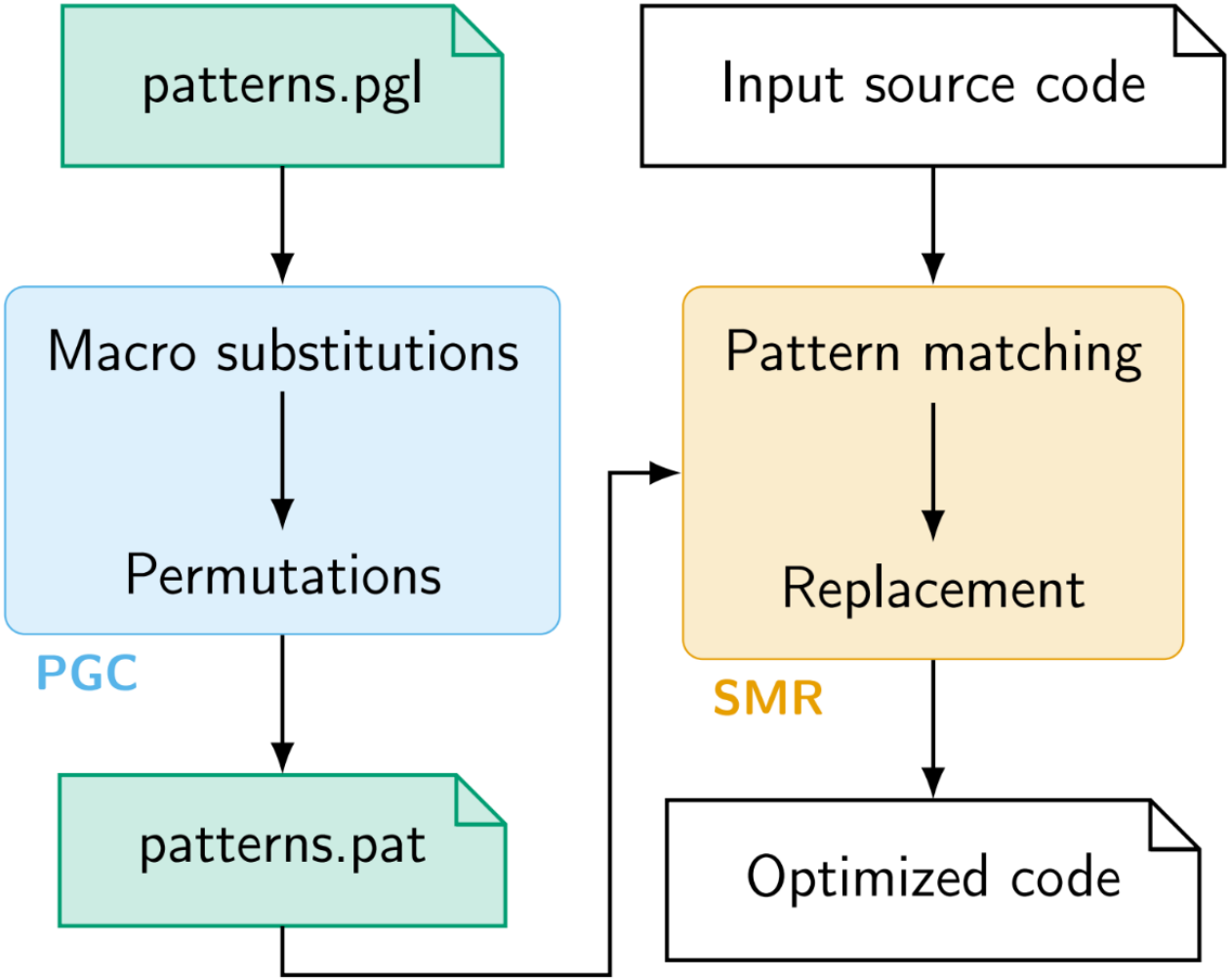
- **Compile input files to MLIR**
- **First step -** Control Dependency Graph (CDG)
  - Filter candidates by control structure
- **Second step -** Data Dependency Graph (DDG)
  - Check candidate and pattern data-flow equality
- **Is a Match?** Then rewrite the input code

Toolchain

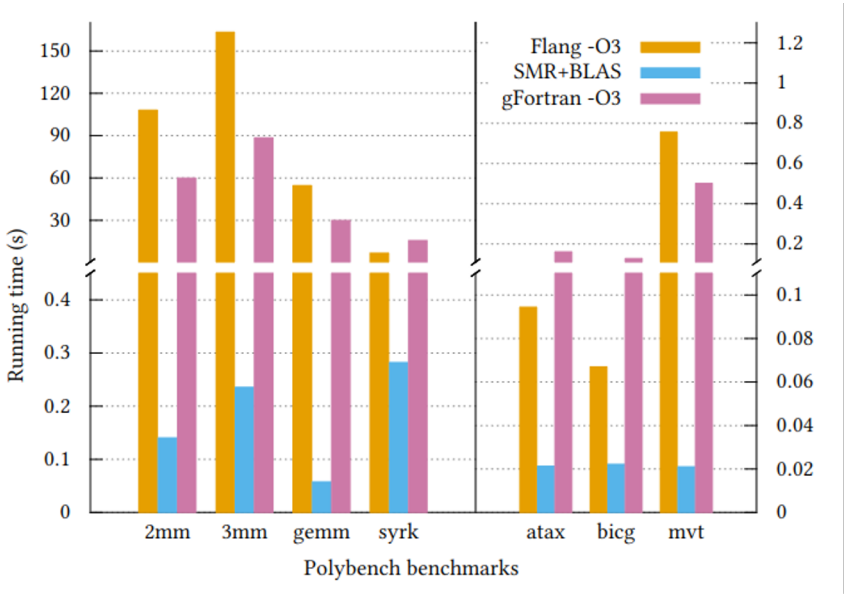

1
patterns.pat
PAT
2
idiom.f90
replacement.f90
Language MLIR Frontend
3
idiom.mlir
replacement.mlir
4
Idiom Preprocessing
Replacement Preprocessing
5
Build Pattern Finite Automata
PFA
Replacements
6
Store PFA & Replacements
SMR
7
patterns.pfa


8
input.f90
Language MLIR Frontend
9
input.mlir
10
patterns.pfa
11
Input Preprocessing
12
Load PFA & Replacements
13
CDG Matching
CDG/DDG Automata
Candidates
14
DDG Matching
Matches
Rewriting
15
Rewritten Code
16
Postprocessing
SMR
Optimized MLIR Input Code

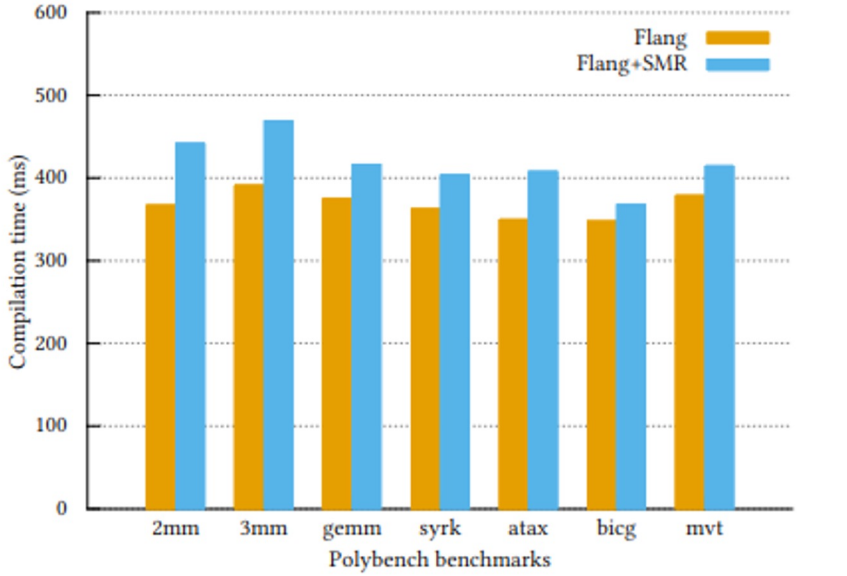
Toolchain


patterns.pgl
Input source code
Macro substitutions
Permutations
PGC
Pattern matching
Replacement
SMR
patterns.pat
Optimized code

Methodology - Usability

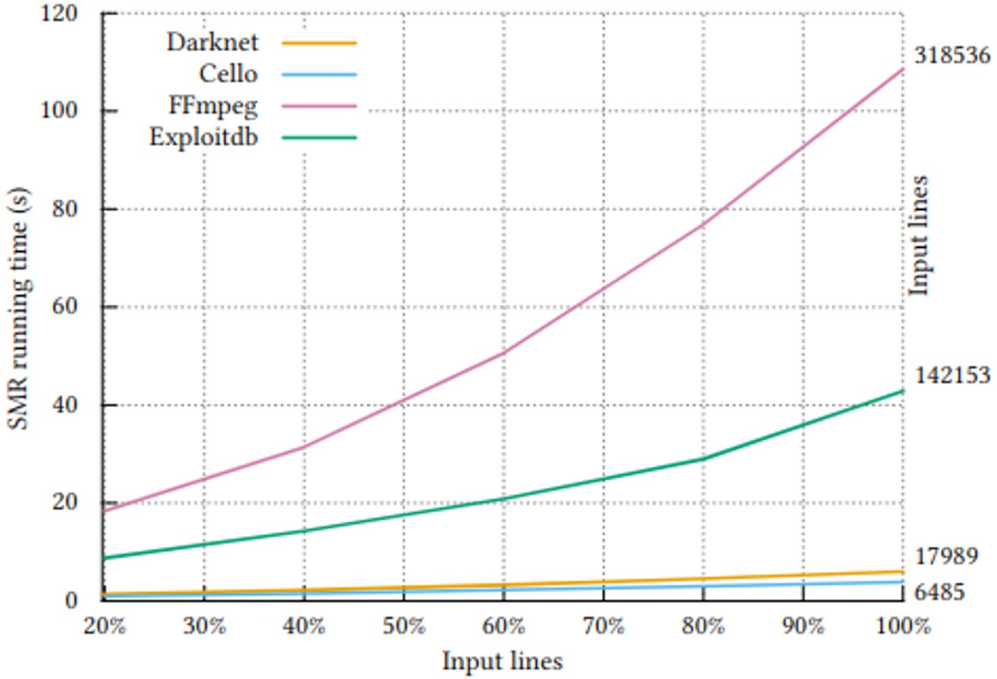


Polybench running time after blas replacement

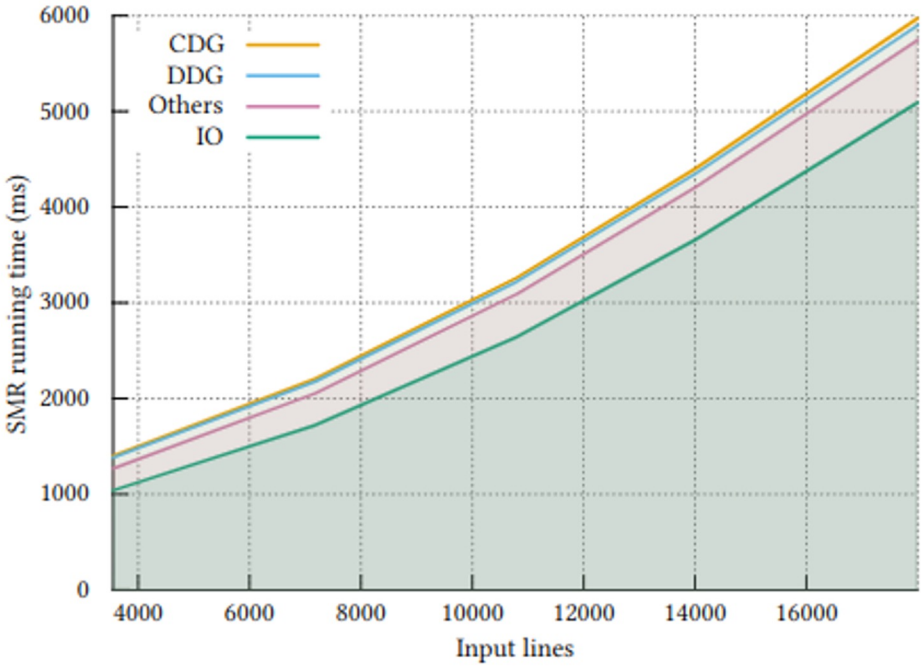


FIR compilation time with/without SMR+BLAS

Methodology - Input Scalability

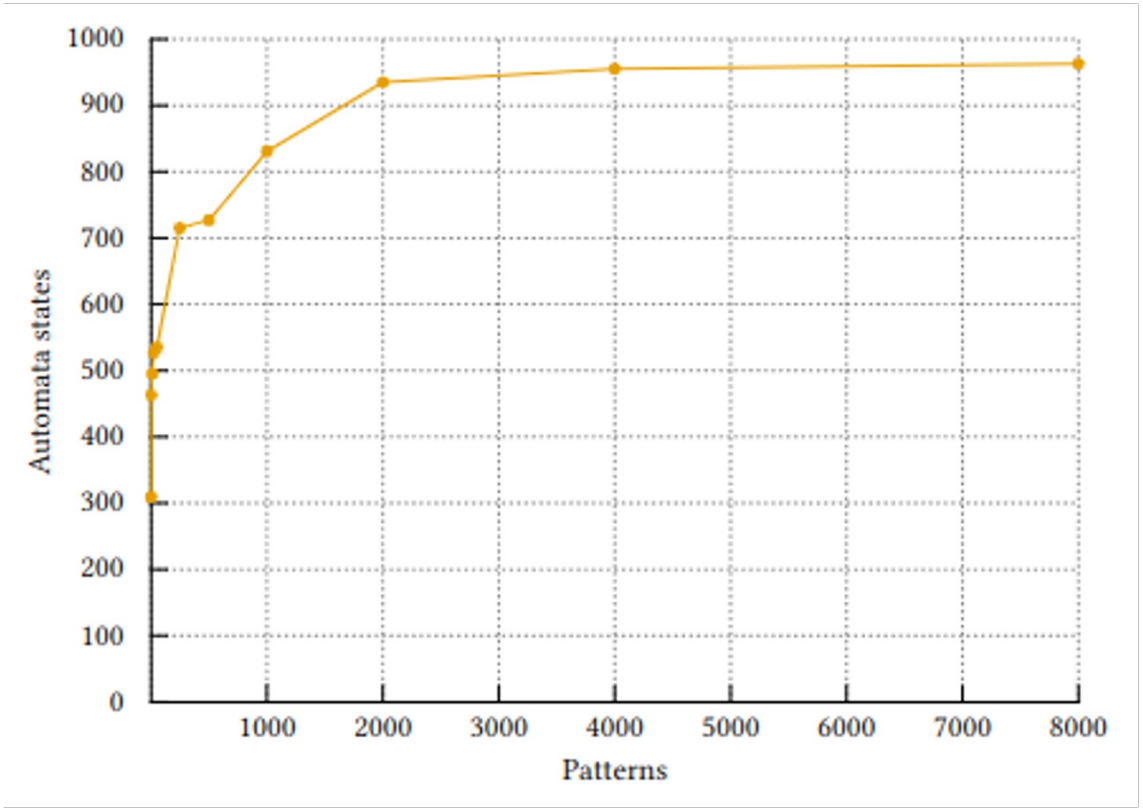


4 input programs against 95 patterns



Darknet breakout

Methodology - Pattern Scalability



SMR's automaton prefix merging

Methodology - Dialects Flexibility

Idiom	Darknet [40]	Cello [26]	Exploitdb [46]	Ffmpeg [16]	Hpgmg [1]	Nekrs [17]	**Total**
saxpy	1						1
scopy	1						1
sdot	1			1			2
sgemm	4						4
scall	2						2
ddot		1			1	2	4
dgemm			1			3	4
dgemmv						1	1
dscal						3	3
Total	9	1	1	1	1	9	22

Matching with CIL and CBLAS idioms

Source Matching and Rewriting for MLIR Using String-Based Automata

VINICIUS ESPINDOLA, Institute of Computing - UNICAMP, Brazil
LUCIANO ZAGO, Institute of Computing - UNICAMP, Brazil
HERVÉ YVIQUEL, Institute of Computing - UNICAMP, Brazil
GUIDO ARAUJO, Institute of Computing - UNICAMP, Brazil

A typical compiler flow relies on a uni-directional sequence of translation/optimization steps that *lower* the program abstract representation, making it hard to preserve higher-level program information across each transformation step. On the other hand, modern ISA extensions and hardware accelerators can benefit from the compiler's ability to detect and *raise* program idioms to acceleration instructions or optimized library calls. Although recent works based on Multi-Level IR (MLIR) have been proposed for code raising, they rely on specialized languages, compiler recompilation, or in-depth dialect knowledge. This paper presents *Source Matching and Rewriting* (SMR), a user-oriented source-code-based approach for MLIR idiom matching and rewriting that does not require a compiler expert's intervention. SMR uses a two-phase automaton-based DAG-matching algorithm inspired by early work on tree-pattern matching. First, the idiom *Control-Dependency Graph* (CDG) is matched against the program's CDG to rule out code fragments that do not have a control-flow structure similar to the desired idiom. Second, candidate code fragments from the previous phase have their *Data-Dependency Graphs* (DDGs) constructed and matched against the idiom DDG. Experimental results show that SMR can effectively match idioms from Fortran (FIR) and C (CIL) programs while raising them as BLAS calls to improve performance. Additional experiments also show performance improvements when using SMR to enable code replacement in areas like approximate computing and hardware acceleration.

CCS Concepts: • **Theory of computation** → *Grammars and context-free languages*; **Pattern matching**; • **Hardware** → Emerging languages and compilers.

Additional Key Words and Phrases: idiom recognition, automata, MLIR, rewriting, hardware accelerators

ACM Reference Format:
Vinicius Espindola, Luciano Zago, Hervé Yviquel, and Guido Araujo. 2018. Source Matching and Rewriting for MLIR Using String-Based Automata. 1, 1 (November 2018), 26 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Idiom recognition is a well-known and studied problem in computer science, which aims to identify program fragments [5, 7, 23, 24, 32, 40]. Although idiom recognition has found a niche in compiling technology, in areas like code generation (e.g., instruction selection) [2, 3, 20, 39], its broad application is considerably constrained by how modern compilers work. A typical compiler flow makes a series of translation passes that lowers the level of abstraction from source to machine code, with the goal of optimizing the code at each level. One such example is the Clang/LLVM

Authors' addresses: Vinicius Espindola, Institute of Computing - UNICAMP, Campinas, SP, Brazil, v188115@dac.unicamp.br; Luciano Zago, Institute of Computing - UNICAMP, Campinas, SP, Brazil, l182835@dac.unicamp.br; Hervé Yviquel, Institute of Computing - UNICAMP, Campinas, SP, Brazil, hyviquel@unicamp.br; Guido Araujo, Institute of Computing - UNICAMP, Campinas, SP, Brazil, guido@unicamp.br.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 Association for Computing Machinery.
XXXX-XXXX/2018/11-ART $15.00
https://doi.org/XXXXXXX.XXXXXXX

Vinicius Espindola, Luciano Zago, Hervé Yviquel, and Guido Araujo. 2023. Source Matching and Rewriting for MLIR Using String-Based Automata. ACM Trans. Archit. Code Optim. 20, 2, Article 22 (June 2023), 26 pages. https://doi.org/10.1145/3571283

Thank you!