Scalable Polyhedral Compilation in Open-Source and AI Compilers

Tobias Grosser
My first scientific publication
2011 CARP: Correct and Efficient Accelerator Programming

ENS ÉCOLE NORMALE SUPÉRIEURE

Google Research PhD Fellowship
Polyhedral today
- Good polyhedral libraries
- Good solutions to some problems (Parallelisation, Tiling, GPGPU)
- Several successful research projects
- First compiler integrations

**but still limited IMPACT.**

Can Polly help to change this?
IMPACT = Scaling?

Scaling

Algorithms
- Asymptotics
- Benchmarks
  - Size
  - Difficulty

Development
- Many Users
- Many Developers

New Research
Polybench - Likely our most widely used artifact

PolyBench/C
the Polyhedral Benchmark suite

News

- 03/19/12: Public release of PolyBench/GPU 1.0. PolyBench/GPU 1.0 was contributed by John Cavazos Scott Grauer-Gray, from U. Delaware.
- 03/28/12: Public release of PolyBench/Fortran 1.0. PolyBench/Fortran 1.0 is a Fortran port of PolyBench/C 3.2.
- 03/12/12: Public release of PolyBench/C 3.2 Download (minor cosmetic and bug fixes, now called PolyBench/C instead of PolyBench)
- 11/13/11: Public release of PolyBench 3.1 (use heap-allocated arrays by default, fix a bug for 3D arrays in 3.0)
- 10/28/11: Public release of PolyBench 3.0 (support of heap-allocating)
- 3/16/11: Public release of PolyBench 2.0 (superset of 1.0 + C99)
- 4/12/10: Public release of PolyBench 1.0

[CITATION] Polybench: The polyhedral benchmark suite
☆ Save  Cite  Cited by 518  Related articles
Polybench - A Dream Benchmark

Small Kernels
5-50 lines
3-10 loops

Well-Behaved
No Integer Wrapping
No-Unbounded Loops

Structured Control
Loops + If-Conditions
(Im)perfectly Nested

Diverse Kernels

Challenging to Optimize
Polly - A polyhedral Compiler for LLVM-IR
# Challenges for Polly

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>LLVM-IR</th>
<th>Polyhedral Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variant Loads in Control Conditions</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Aliasing Arrays</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Integer Wrapping</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Out-of-Bound Accesses</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Potentially Unbounded Loops</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
</tbody>
</table>
The COSMO Atmospheric Model

Used by 7 national weather services (DE, CH, IT, …)
Resolution: 35m

What resolution is needed?
Resolution: 35m

What resolution is needed?
Resolution: 140m

What resolution is needed?
Resolution: 560m

What resolution is needed?
Resolution: 1.1km (Today)

What resolution is needed?
Challenges

**Computation**

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>1 km²</td>
</tr>
<tr>
<td>Surface</td>
<td>40,000 km² (CH)</td>
</tr>
<tr>
<td>Duration</td>
<td>2-7 days (weather)</td>
</tr>
<tr>
<td>Time-to-Solution</td>
<td>3 months</td>
</tr>
<tr>
<td>Surface</td>
<td>→ 500,000,000 km²</td>
</tr>
<tr>
<td>Duration</td>
<td>→ 100 years</td>
</tr>
</tbody>
</table>

**Software (COSMO)**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Fortran</td>
</tr>
<tr>
<td>Size</td>
<td>300,000 LoC</td>
</tr>
<tr>
<td>Loops</td>
<td>thousands</td>
</tr>
<tr>
<td>Multi-Domain</td>
<td>Physics, Stencils, General-Purpose, MPI</td>
</tr>
</tbody>
</table>

**Hardware**

Insufficient memory bandwidth

**Community**

DSL and Non-DSL code, HPC engineer wants control, …
The Size of Deep Neural Networks

How to scale to such large networks?
mlir-meminfo: A Memory Model for MLIR

Kunwar Grover
Arjun Pitchanathan
Tobias Grosser
… and other contributors

IIIT Hyderabad
University of Edinburgh
University of Edinburgh
A Transformation on 2D Matrix Multiplication

Is the miss rate of the final program intuitive?
How long does it take to profile?
What about bigger programs? : Transformers

Is the miss rate of the final program intuitive?

How long does it take to profile?
The Unintuitive Cost of Data-Movement

Which has the best cache miss rate?
Time to understand this cache behavior on BERT

- Profiling: 18min
- mlir-meminfo: 12sec
- Instant (<1 sec)
- Responsive (<15 sec)
- Coffee Break (>15 min)
mlir-meminfo on Matmul Transform

mlir-meminfo --cache-lines 512 --associativity 8 Mul.mlir

[kunwar@node03 build]$ ./bin/mlir-meminfo -cs 512 -a 8 ../memref-examples/Mul.mlir
module attributes {torch.debug_module_name = "Mul"} {
  ml_program.global private mutable @global_seed(dense<0> : tensor<i64>) : tensor<i64>
  func.func @forward(%arg0: memref<1024x1024xf32>, %arg1: memref<1024x1024xf32>) -> memref<1024x1024xf32> { 
    %cst = arith.constant 0.000000e+00 : f32
    %cast = memref.cast %arg1 : memref<1024x1024xf32> to memref<1024x1024xf32>
    %cast_0 = memref.cast %arg0 : memref<1024x1024xf32> to memref<1024x1024xf32>
    %alloc = memref.alloc() {alignment = 64 : i64} : memref<1024x1024xf32>
    linalg.fill ins(%cst : f32) outs(%alloc : memref<1024x1024xf32>)
    | |
    | | Miss rate: 6.25% Access percentage: 0.0244081
    |
    linalg.matmul ins(%cast_0, %cast : memref<1024x1024xf32>, memref<1024x1024xf32>) outs(%alloc : memref<1024x1024xf32>)
    | |
    | | Miss rate: 25.0289% Access percentage: 99.9756
    |
    memref.dealloc %alloc : memref<1024x1024xf32>
    return %alloc : memref<1024x1024xf32>
  }
}
Miss rate: 25.0244
Total time: 140.039ms
mlir-meminfo on Matmul Transform

mlir-meminfo --cache-lines 512 --associativity 8 Mul.mlir

```
linalg.fill ins(%cst : f32) outs(%alloc : memref<1024x1024xf32>)
   Miss rate: 6.25%   Access percentage: 0.0244081

linalg.matmul ins(%cast_0, %cast : memref<1024x1024xf32>, memref<1024x1024xf32>) outs(%alloc : memref<1024x1024xf32>)
   Miss rate: 25.0289%   Access percentage: 99.9756
```
mlir-meminfo on Matmul Transform

\[
\begin{align*}
\text{llalg.fill ins}(\text{cst : f32}) & \text{ outs}(\text{alloc : memref\langle1024x1024xf32\rangle}) \\
\text{llalg.matmul ins}(\text{\%cast_0, \%cast : memref\langle1024x1024xf32\rangle, memref\langle1024x1024xf32\rangle}) & \text{ outs}(\text{alloc : memref\langle1024x1024xf32\rangle}) \\
\text{llalg.mmt4d ins}(\text{\%alloc_2, \%alloc_4 : memref\langle64x64x16x16xf32\rangle, memref\langle64x64x16x16xf32\rangle}) & \text{ outs}(\text{\%alloc_1 : memref\langle64x64x16x16xf32\rangle})
\end{align*}
\]

Miss rate: 6.25%  Access percentage: 0.0244081

Miss rate: 25.0289%  Access percentage: 99.9756

< 150 ms!

Miss rate: 1.75781%  Access percentage: 99.8051

< 150 ms!
Matmul Transform on BERT

```
linalg.fill ins(%cst_5 : f32) outs(%alloc_226 : memref<32768x768xf32>)

------> Miss rate: 6.25%  Access percentage: 0.000202098

%alloc_227 = memref.alloc() {alignment = 64 : i64} : memref<32768x768xf32>
memref.copy %alloc_226, %alloc_227 : memref<32768x768xf32> to memref<32768x768xf32>
linalg.matmul ins(%collapse_shape, %alloc_224 : memref<32768x768xf32>, memref<768x768xf32>) outs(%alloc_227 : memref<32768x768xf32>)

------> Miss rate: 26.5645%  Access percentage: 0.620844
```
Time to understand this cache behavior on BERT

Profiling
- mlir-meminfo: 0.5 sec
- mlir-meminfo transform: 12 sec

Coffee Break (> 15 min)

Instant (< 1 sec)
Responsive (< 15 sec)
mlir-meminfo Matmul Transform on BERT

- Miss rate: 0.197347
- Original miss rate: 26.5645%
- Access percentage: 0.617006
Algorithm
LRU Cache Policy

accesses

cache state

most recent
LRU Cache Policy

accesses

MISS

A

cache state

A

most recent
LRU Cache Policy

accesses

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
</table>

| MISS | MISS |

cache state

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
</table>

most recent
LRU Cache Policy

accesses

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>MISS</td>
<td>MISS</td>
<td>MISS</td>
<td></td>
</tr>
</tbody>
</table>

cache state

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

most recent
LRU Cache Policy

accesses

A  B  C  C

MISS  MISS  MISS  HIT!

cache state

A  B  C

most recent
LRU Cache Policy

accesses

cache state

most recent
LRU Cache Policy

accesses
A  B  C  C  D  D

cache state
B  C  D

most recent
LRU Cache Policy

accesses

MISS  MISS  MISS  MISS  HIT!
A     B     C     C     D

cache state

B     C     D
most recent
LRU Cache Policy

accesses

A  B  C  C  D  D  D  D

MISS  MISS  MISS  MISS  HIT!

cache state

B  C  D

most recent
LRU Cache Policy

accesses:

MISS MISS MISS MISS
A B C C

cache state:

C D B
most recent

HIT!
LRU Cache Policy

accesses

MISS MISS MISS MISS HIT! MISS
A B C C D D D D B A

cache state

D B A

most recent
LRU Cache Policy

accesses
A B C C D D D D C A

cache state
D B A = last N unique locations accessed
LRU Cache Policy

accesses

A  B  C  C  D  D  D  D  B  A

miss if…
LRU Cache Policy

- access order: A B C C D D D D B A

- Misses:
  - A, B, C, C, D, D, D, D, B, A

- Hit: D

- Miss if...
  - first access to location
LRU Cache Policy

miss if…

● first access to location
● previous access to same location was “long” ago
Most recent access: how long ago for miss?

2 unique memory accesses ago; hit
Most recent access: how long ago for miss?

3 unique memory accesses ago; miss
Earlier Cache Modelling Algorithms

\[ A = \text{access map} \quad L_\leq = \{ (i_0, \ldots, i_n) \rightarrow (j_0, \ldots, j_n) : \]
\[ (i_0, \ldots, i_n) \leq (j_0, \ldots, j_n) \land \]
\[ (i_0, \ldots, i_n), (j_0, \ldots, j_n) \in S_{ran} \} \]

\[ I = \text{iteration domain} \]

\[ S = \text{schedule map} \]

\[ L_\leq = \{ (i_0, \ldots, i_n) \rightarrow (j_0, \ldots, j_n) : \]
\[ (i_0, \ldots, i_n) \leq (j_0, \ldots, j_n) \land \]
\[ (i_0, \ldots, i_n), (j_0, \ldots, j_n) \in S_{ran} \} \]

\[ F = (S^{-1} \circ L_\leq \circ S) \circ N^{-1} \]

\[ E = S \circ A^{-1} \circ A \circ S^{-1} \]

\[ B = S^{-1} \circ L_{\leq}^{-1} \circ S \]

\[ N = S^{-1} \circ \text{lexmin}(L_\leq \cap E) \circ S \]

\[ D = \{|A \circ (F \cap B)|\} \]

\[ F = S^{-1} \circ \text{lexmin}(S \circ A^{-1}) \quad \text{(different F)} \]

compulsory misses = \(|F_{dom}|\)
Most recent access = dependence analysis
Most recent access = dependence analysis

for i = 0 to 499:
    load A[i]
Most recent access = dependence analysis

for i = 0 to 499:
    load A[i]

for i = 0 to 499:
    load B[i]
Most recent access = dependence analysis

for i = 0 to 499:
    load A[i]

for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]
Most recent access = dependence analysis

for i = 0 to 499:
    load A[i]

for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]

for i = 0 to 499:
    load C[i]
Most recent access = dependence analysis

for i = 0 to 499:
    load A[i]

for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]

for i = 0 to 499:
    load C[i]

load C[0]
Most recent access = dependence analysis

for i = 0 to 499:
    load A[i]

for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]

for i = 0 to 499:
    load C[i]

load C[0]
load B[400]
Most recent access = dependence analysis

for i = 0 to 499:
    load A[i]

for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]

for i = 0 to 499:
    load C[i]

load C[0]
load B[400]
load A[x]  (in bounds)
Most recent access = dependence analysis

for i = 0 to 499:
    load A[i]

for i = 0 to 499:
    load B[i]
    no dependence; first access; miss

for i = 0 to 499:
    load C[i]

for i = 0 to 499:
    load C[i]

load C[0]
load B[400]
load A[x]
Most recent access = dependence analysis

for i = 0 to 499:
    load A[i]

for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]

for i = 0 to 499:
    load C[i]

load C[0]
load B[400]
load A[x]
Most recent access = dependence analysis

for i = 0 to 499:
    load A[i]

for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]

for i = 0 to 499:
    load C[i]

load C[0]
load B[400]
load A[x]
Most recent access = dependence analysis

for i = 0 to 499:
    load A[i]

for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]

for i = 0 to 499:
    load C[i]

load C[0]
load B[400]
load A[x]
How far is the most recent access?

for i = 0 to 499:
    load A[i]

for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]

for i = 0 to 499:
    load C[i]

load C[0]  499 unique memory accesses; hit
load B[400]
load A[x]
load C[1..499]
How far is the most recent access?

```
for i = 0 to 499:
    load A[i]

for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]

for i = 0 to 499:
    load C[i]

load C[0]
load B[400]
load A[x]
```

500 + 99 unique memory accesses; miss C[0..499], B[401..499]
How far is the most recent access?

```plaintext
for i = 0 to 499:
    load A[i]

for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]

load C[0]
load B[400]
load A[x]
```

500 + 500 + ??? ≥ 512; miss
C[0..499], B[0..499], A[???]

Cache size = 512
Loop nests beyond 512 have no relevance

for i = 499 to 0:
    load A[i]

for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]

for i = 0 to 499:
    load C[i]

load C[0]
load B[400]
load A[x]

\[500 + 500 + \text{???} \geq 512; \text{miss}\]
C[0..499], B[0..499], A[???]

cache size = 512
Loop nests beyond 512 have no relevance

```plaintext
for i = 499 to 0:
    load A[i]
    load A[i + 1]
for i = 0 to 499:
    load B[i]
for i = 0 to 499:
    load C[i]
for i = 0 to 499:
    load C[i]
load C[0]
load B[400]
load A[x]
```

500 + 500 + ??? ≥ 512; miss
C[0..499], B[0..499], A[???]

Cache size = 512
Loop nests beyond 512 have no relevance

for $i = 499$ to 0:
    load $B[i]$

for $i = 0$ to 499:
    load $B[i]$

for $i = 0$ to 499:
    load $C[i]$

for $i = 0$ to 499:
    load $C[i]$

load $C[0]$
load $B[400]$
load $A[x]$

$500 + 500 + ??? \geq 512$; miss $C[0..499], B[0..499], ???$

cache size = 512
Loop nests beyond 512 have no relevance

for $i = 0$ to 499:
  load $B[i]$

for $i = 0$ to 499:
  load $C[i]$

for $i = 0$ to 499:
  load $C[i]$

load $C[0]$
load $B[400]$
load $A[x]$

500 + 500 + ??? ≥ 512; miss $C[0..499], B[0..499], ???$
Asymptotic improvement on practical programs

for i = 0 to 499:
    load A[i]

for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]

for i = 0 to 499:
    load C[i]

load C[0]
load B[400]
load A[x]

checking all pairs of statements for dependences: $O(n^2)$
Asymptotic improvement on practical programs

for i = 0 to 499:
  load B[i]

for i = 0 to 499:
  load C[i]

for i = 0 to 499:
  load C[i]

load C[0]
load B[400]
load A[x]

checking a few loop nests for each statement: $O(n)$
Incremental recomputation

for i = 0 to 499:
  load B[i]

for i = 0 to 499:
  load C[i]

for i = 0 to 499:
  load C[i]

load C[0]
load B[400]
load A[x] ← this always misses
Incremental recomputation

for i = 499 to 0:
    load A[i]  
teel this

for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]

for i = 0 to 499:
    load C[i]

load C[0]
load B[400]
load A[x]  
teel this always misses


cache size = 512
Incremental recomputation

```plaintext
for i = 499 to 0:
    load A[i]
    load A[i + 1]
for i = 0 to 499:
    load B[i]
for i = 0 to 499:
    load C[i]
load C[0]
load B[400]
load A[x]
```

change this

this always misses

cache size = 512
Incremental recomputation

for i = 499 to 0:
    load B[i]  
    change this

for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]

for i = 0 to 499:
    load C[i]

load C[0]
load B[400]
load A[x]  
this always misses
Incremental recomputation

for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]

for i = 0 to 499:
    load C[i]

load C[0]
load B[400]
load A[x] ← this always misses
Incremental recomputation

for i = 0 to 499:
  load B[i]

for i = 0 to 499:
  load C[i]

for i = 0 to 499:
  load C[i]

load C[0]
load B[400]
load A[x]

change this
Incremental recomputation

```
for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]

load C[0]
load B[400]
load A[x]
```
Incremental recomputation

for $i = 0$ to 499:
  load $B[i]$

for $i = 0$ to 499:
  load $C[i]$

for $i = 0$ to 499:
  load $C[i]$

load $C[0]$
load $B[400]$
load $A[x]$

change this

$500$

$1000 \geq 512$

cache size = 512
Incremental recomputation

```python
for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]

for i = 0 to 499:
    load C[i]

load C[0]
load B[400]
load A[x]
```

1000 ≥ 512

same cache performance!
Incremental recomputation

```plaintext
for i = 0 to 499:
  load B[i]

for i = 0 to 499:
  load C[i]

don't involve the changed region

for i = 0 to 499:
  load C[i]

load C[0]
load B[400]
load A[x]
```

cache size = 512
Incremental recomputation

```plaintext
for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]

for i = 0 to 499:
    load C[i]

load C[0]
load B[400]
load A[x]
```

Involves changed region, but sure miss anyway
Incremental recomputation

for $i = 0$ to 499:
    load $B[i]$

for $i = 0$ to 499:
    load $C[i]$

possibly changed cache performance

1000 ≥ 512

load $C[0]$
load $B[400]$
load $A[x]$
Incremental recomputation

```plaintext
for i = 0 to 499:
    load B[i]

for i = 0 to 499:
    load C[i]

for i = 0 to 499:
    load C[i]

load C[0]
load B[400]
load A[x]
```

change this

constant amount of recomputation

cache size = 512
Fast Polyhedral Library
Applicability of Transprecision

- **74.8%** int16_t, 32 cols
- **24.7%** int64_t
- **<1%** int128_t

Presburger Arithmetic in MLIR

FlatAffineConstraints
Integer Polyhedron

FlatAffineRelation
Affine Relations

PresburgerSet
Union of Integer Polyhedra

PresburgerRelation
Union of Affine Relations
FPL: library-level transprecision

```cpp
void copy(const Vec16x32 &src, Vec16x32 &dst) {
    dst = src;
}
```

isl optimization: element-TP

```cpp
void copy(unsigned n, tpint **src, tpint **dst) {
    for (unsigned i = 0; i < n; ++i) {
        if (is_int32(src[i]))
            set_int32(dst[i], get_int32(src[i]));
        else
            set_gmp(dst[i], get_gmp(src[i]));
    }
}
```
isl

- Full Presburger Arithmetic
- Scalar Arithmetic
- Standalone C Library

FPL

- Full Presburger Arithmetic
- Exploits SIMD Parallelism
- Modern C++ Library, Integrated into MLIR
  - Currently Upstreaming
The internal representation

Internal Representation

<table>
<thead>
<tr>
<th>Integer Polyhedron</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2x + 2y \leq 13)</td>
</tr>
<tr>
<td>(-x + 3y \leq 9)</td>
</tr>
<tr>
<td>(2x + 2y \geq 1)</td>
</tr>
<tr>
<td>(-x + 10y \geq 0)</td>
</tr>
</tbody>
</table>
Presburger Sets: Unions of Integer Polyhedra

\[ \begin{align*}
  x - y & \geq 0 \\
  x - y & \leq 2 \\
  x + y & \geq 2 \\
  x + y & \leq 4 \\
  x - y & \geq -4 \\
  x - y & \leq -2 \\
  x + y & \geq 4 \\
  x + y & \leq 8
\end{align*} \]
The internal representation

\[ \begin{align*}
2x + 2y & \geq 1 \\
2x + 2y & \leq 13 \\
3y & \leq x + 9 \\
10y & \geq x
\end{align*} \]

Internal Representation

<table>
<thead>
<tr>
<th>Integer Polyhedron</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2x + 2y \leq 13)</td>
</tr>
<tr>
<td>(-x + 3y \leq 9)</td>
</tr>
<tr>
<td>(2x + 2y \geq 1)</td>
</tr>
<tr>
<td>(-x + 10y \geq 0)</td>
</tr>
</tbody>
</table>
Presburger Sets: Unions of Integer Polyhedra

\[ x - y \geq 0 \]
\[ x - y \leq 2 \]
\[ x + y \geq 2 \]
\[ x + y \leq 4 \]

\[ x - y \geq -4 \]
\[ x - y \leq -2 \]
\[ x + y \geq 4 \]
\[ x + y \leq 8 \]
The internal representation

\[\begin{align*}
(2x + 2y) &\geq 1 \\
(2x + 2y) &\leq 13 \\
3y &\leq x + 9 \\
10y &\geq x
\end{align*}\]
Presburger Sets: Unions of Integer Polyhedra

\[ U \]

<table>
<thead>
<tr>
<th>Presburger Set</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Integer Polyhedron</strong></td>
</tr>
<tr>
<td>( x - y \geq 0 )</td>
</tr>
<tr>
<td>( x - y \leq 2 )</td>
</tr>
<tr>
<td>( x + y \geq 2 )</td>
</tr>
<tr>
<td>( x + y \leq 4 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Integer Polyhedron</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>( x - y \geq -4 )</td>
</tr>
<tr>
<td>( x - y \leq -2 )</td>
</tr>
<tr>
<td>( x + y \geq 4 )</td>
</tr>
<tr>
<td>( x + y \leq 8 )</td>
</tr>
</tbody>
</table>
The internal representation

2x + 2y ≥ 1
2x + 2y ≤ 13
3y ≤ x + 9
10y ≥ x

Internal Representation

<table>
<thead>
<tr>
<th>Integer Polyhedron</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x + 2y ≤ 13</td>
</tr>
<tr>
<td>-x + 3y ≤ 9</td>
</tr>
<tr>
<td>2x + 2y ≥ 1</td>
</tr>
<tr>
<td>-x + 10y ≥ 0</td>
</tr>
</tbody>
</table>
Presburger Sets: Unions of Integer Polyhedra

Presburger Set

<table>
<thead>
<tr>
<th>Integer Polyhedron</th>
<th>x - y ≥ 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x - y ≤ 2</td>
</tr>
<tr>
<td></td>
<td>x + y ≥ 2</td>
</tr>
<tr>
<td></td>
<td>x + y ≤ 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Integer Polyhedron</th>
<th>x - y ≥ -4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x - y ≤ -2</td>
</tr>
<tr>
<td></td>
<td>x + y ≥ 4</td>
</tr>
<tr>
<td></td>
<td>x + y ≤ 8</td>
</tr>
</tbody>
</table>
A Transprecision Presburger Library

TransprecisionSet

Precision Dispatcher

Increase precision
Retry

intersect()

isEmpty()

Genaralized Basis Reduction <int16_t>

int16_t

isEqual()

subtract()

Simplex<int16_t>

complement()

coalesce()

Parametric Integer Programming <int16_t>

union()

evacuate Existentials()

PresburgerSet<int16_t>

Core Algorithms <int16_t>

int64_t

PresburgerSet<int64_t>

Core Algorithms <int64_t>

BigInt

PresburgerSet<BigInt>

Core Algorithms <BigInt>

int64_t

Overflow!

Overflow!
Algorithmic Design

intersect() → isEmpty() → Generalized Basis Reduction

isEqual() → subtract() → Simplex

complement() → complement() → Parametric Integer Programming

coaalesce() → union() → eliminate Existentials()
Our speedup comes from the long-running cases
Library-level transprecision has low overhead

Overall* speedup over FPL (transprecision)

- FPL (128-bit, unchecked): 0.64
- FPL (64-bit, unchecked): 0.86
- FPL (16-bit, unchecked): 1.08
Now in MLIR — used in Affine Loop Fusion, CIRCT ...

Currently uses unchecked 64-bit arithmetic; patch for introducing fast arbitrary precision is under review.
Now in MLIR

```
$ arjun@haley ~/llvm/llvm-project-patch/mlir/include/mlir/Analysis % tree

  AffineAnalysis.h
  AffineStructures.h
  AliasAnalysis
    LocalAliasAnalysis.h
  AliasAnalysis.h
  BufferAliasAnalysis.h
  CallGraph.h
  LinearTransform.h
  Liveness.h
  LoopAnalysis.h
  NestedMatcher.h
  NumberOfExecutions.h
  Presburger
    Fraction.h
    Matrix.h
    Simplex.h
  PresburgerSet.h
  SliceAnalysis.h
  Utils.h
```

```
$ arjun@haley ~/llvm/llvm-project-patch/mlir/unittests/Analysis % tree

  AffineStructuresTest.cpp
  CMakeLists.txt
  LinearTransformTest.cpp
  Presburger
    CMakeLists.txt
    MatrixTest.cpp
    SimplexTest.cpp
  PresburgerSetTest.cpp
```
Subview fusion through equality checks

%0 = memref.alloc() : memref<3x512xbf16, 1>
%1 = memrefSubview %0[0, 0][3, 256][1, 1] : ...
%2 = memrefSubview %0[0, 256][3, 256][1, 1] : ...
// write to %1
// write to %2

Stop reimplementing operations!
Using FPL in MLIR
Loop Fusion: Checking for inter-loop dependencies

affine.for i = 0 to 4 {
    %c = affine.load %C[i] : memref<4xf32>
S0: affine.store %c, %B[i] : memref<4xf32>
}

affine.for j = 0 to 4 {
S1: %c = affine.load %B[3 - j] : memref<4xf32>
    affine.store %c, %B[j] : memref<4xf32>
}
Loop Fusion: Checking for inter-loop dependencies

```cpp
// Create access relation from each MemRefAccess.
FlatAffineRelation srcRel, dstRel;

MemRefAccess srcAccess(iStoreOp);
srcAccess.getAccessRelation(srcRel);
(i) -> (x): (i >= 0 and i < 4 and i = x)

MemRefAccess dstAccess(jLoadOp);
dstAccess.getAccessRelation(dstRel);
(j) -> (y): (j >= 0 and j < 4 and y = 3 - j)

affine.for i = 0 to 4 {
    %c = affine.load %C[%i] : memref<4xf32>
    S0: affine.store %c, %B[%i] : memref<4xf32>
}

affine.for j = 0 to 4 {
    %c = affine.load %B[3 - %j] : memref<4xf32>
    affine.store %c, %B[%j] : memref<4xf32>
}
```
Loop Fusion: Checking for inter-loop dependencies

// Create access relation from each MemRefAccess.
FlatAffineRelation srcRel, dstRel;

affine.for i = 0 to 4 {
  %c = affine.load %C[%i] : memref<4xf32>
  S0: affine.store %c, %B[%i] : memref<4xf32>
}

affine.for j = 0 to 4 {
  S1: %c = affine.load %B[3 - %j] : memref<4xf32>
  affine.store %c, %B[%j] : memref<4xf32>
}

MemRefAccess srcAccess(iStoreOp);
srcAccess.getAccessRelation(srcRel);

(i) -> (x): (i >= 0 and i < 4 and i = x)

MemRefAccess dstAccess(jLoadOp);
dstAccess.getAccessRelation(dstRel);

(j) -> (y): (j >= 0 and j < 4 and y = 3 - j)
Loop Fusion: Checking for inter-loop dependencies

```cpp
// Create access relation from each MemRefAccess.
FlatAffineRelation srcRel, dstRel;

MemRefAccess srcAccess(iStoreOp);
srcAccess.getAccessRelation(srcRel);

// Compute the dependence relation by composing
// `srcRel` with the inverse of `dstRel`.
dstRel.inverse();

affine.for i = 0 to 4 {
    %c = affine.load %C[%i] : memref<4xf32>
S0:  affine.store %c, %B[%i] : memref<4xf32>
}

affine.for j = 0 to 4 {
S1:  %c = affine.load %B[3 - %j] : memref<4xf32>
    affine.store %c, %B[%j] : memref<4xf32>
}
```

```
(i) -> (x): (i >= 0 and i < 4 and i = x)
(j) -> (y): (j >= 0 and j < 4 and y = 3 - j)
(y) -> (j): (j >= 0 and j < 4 and y = 3 - j)
```
Loop Fusion: Checking for inter-loop dependencies

```cpp
// Create access relation from each MemRefAccess.
FlatAffineRelation srcRel, dstRel;

MemRefAccess srcAccess(iStoreOp);
srcAccess.getAccessRelation(srcRel);

MemRefAccess dstAccess(jLoadOp);
dstAccess.getAccessRelation(dstRel);

// Compute the dependence relation by composing `srcRel` with the inverse of `dstRel`.
dstRel.inverse();
dstRel.compose(srcRel);
```

```cpp
affine.for i = 0 to 4 {
    %c = affine.load %C[%i] : memref<4xf32>
    S0: affine.store %c, %B[%i] : memref<4xf32>
}

affine.for j = 0 to 4 {
    S1: %c = affine.load %B[3 - %j] : memref<4xf32>
    affine.store %c, %B[%j] : memref<4xf32>
}
```

- (i) -> (x): (i >= 0 and i < 4 and i = x)
- (j) -> (y): (j >= 0 and j < 4 and y = 3 - j)
- (i) -> (j): (i >= 0 and i < 4 and j >= 0 and j < 4 and i = 3 - j)
Loop Fusion: Checking for inter-loop dependencies

// Create access relation from each MemRefAccess.
FlatAffineRelation srcRel, dstRel;

MemRefAccess srcAccess(iStoreOp);
srcAccess.getAccessRelation(srcRel);
(i) -> (x): (i >= 0 and i < 4 and i = x)

MemRefAccess dstAccess(jLoadOp);
dstAccess.getAccessRelation(dstRel);
(j) -> (y): (j >= 0 and j < 4 and y = 3 - j)

// Compute the dependence relation by composing // `srcRel` with the inverse of `dstRel`.
dstRel.inverse();
(y) -> (j): (j >= 0 and j < 4 and y = 3 - j)
dstRel.compose(srcRel);
(i) ->(j): (i >= 0 and i < 4 and
j >= 0 and j < 4 and i = 3 - j)

bool hasDependency = dstRel.isIntegerEmpty();
Scaling the Community
Top 1.3% of LLVM Contributors (2,929 overall)

32015  Chris Lattner  3095  Nico Weber
9508  Craig Topper  3009  Fariborz Jahanian
8456  Simon Pilgrim  3006  Fangrui Song
7376  Rafael Espindola  2924  Greg Clayton
6108  Ted Kremenek  2886  NAKAMURA Takumi
5572  Sanjay Patel  2793  Eli Friedman
5441  Daniel Dunbar  2776  Devang Patel
5371  Evan Cheng  2523  Jim Grosbach
5362  Douglas Gregor  2452  Argyrios Kyratzidis
5311  Dan Gohman  2448  Jakob Stoklund Olesen
4859  Matt Arsenault  2381  Pavel Labath
4735  Benjamin Kramer  2309  Tobias Grosser
4591  Rui Ueyama  2269  Owen Anderson
4521  Richard Smith  2219  Anders Carlsson
3939  Chandler Carruth  2202  Eric Fiselier
3704  Reid Spencer  2170  Johnny Chen
3672  Bill Wendling  2148  Lang Hames
3511  Eric Christopher  2143  Zachary Turner
3150  Reid Kleckner
3104  David Blaikie

32. Most Contributions

Building a Career on Open-Source Research
Fast Polyhedral Core Library

- 52 patches committed to MLIR
- Arjun and Kunwar have commit rights
- Weekly public video calls

Objectives

- Open Source
- Open Development
- Active Community

Distributed Developer Team

Arjun IIIT Hyderabad
Christian ETH Zurich
Michel ETH Zurich
Kunwar IIIT Hyderabad

Proposal
Acceptance
Open Development

Jul 2020
Sep 2020
Sep 2021
Mar 2022

https://grosser.science/FPL

HiPEAC Technology Transfer Award 2022

Presentation @ LLVM Developer Meeting 2021

Distinguished Paper @ OOPSLA 21

External Users: 唐适之 @ Tsinghua, Intel, PolyMage, …

HiPEAC Technology Transfer Award 2022

External Users: 唐适之 @ Tsinghua, Intel, PolyMage, …

Objectives

- Open Source
- Open Development
- Active Community

Distributed Developer Team

Arjun IIIT Hyderabad
Christian ETH Zurich
Michel ETH Zurich
Kunwar IIIT Hyderabad

Proposal
Acceptance
Open Development

Jul 2020
Sep 2020
Sep 2021
Mar 2022

https://grosser.science/FPL

HiPEAC Technology Transfer Award 2022

Presentation @ LLVM Developer Meeting 2021

Distinguished Paper @ OOPSLA 21

External Users: 唐适之 @ Tsinghua, Intel, PolyMage, …
Building Open Communities

**Isolated Research**

Inhouse Research & Development → Release Code → Feedback & Upstreaming → No Community Trust -> Large Code Drop

**Open Research**

Announce → Public Code Reviews → Commit Code Directly in LLVM

External Developers Join → Students Graduate

Uni...
FPL’s growing upstream development community

Number of landed patches

Jul 20  Sep 20  Nov 20  Jan 21  Mar 21  May 21  Jul 21  Sep 21  Nov 21  Jan 22  Mar 22
**13 Public Code Reviews in 13 Days ~ 1 discussion / day**

### FPL Review History

<table>
<thead>
<tr>
<th>Pull Request</th>
<th>Title</th>
<th>Author</th>
<th>Reviewers</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>D116636</td>
<td>Move PresburgerSet to Presburger/ directory</td>
<td>Groverkss</td>
<td>arjung, bondhugula, fynse, aarvik</td>
<td>Sat, Jan 8, 10:13 AM</td>
</tr>
<tr>
<td>D116681</td>
<td>Move presburger functionality from FlatAffineConstraints to L...</td>
<td>Groverkss</td>
<td>arjung, bondhugula, fynse</td>
<td>Fri, Jan 7, 7:59 PM</td>
</tr>
<tr>
<td>D80860</td>
<td>Exact integer emptiness checks for FlatAffineConstraints</td>
<td>Groverkss</td>
<td>arjung, andydavis1, chelini, Kayjukh, grosser, bondhugula</td>
<td>Thu, Jan 6, 1:50 PM</td>
</tr>
<tr>
<td>D115595</td>
<td>Add division normalization by GCD in <code>getDivRepr</code> fn.</td>
<td>Groverkss</td>
<td>arjung</td>
<td>Thu, Jan 6, 10:52 AM</td>
</tr>
<tr>
<td>D116672</td>
<td>Simplex:normalizeRow: early exit when gcd is one</td>
<td>Groverkss</td>
<td>arjung</td>
<td>Thu, Jan 6, 3:28 AM</td>
</tr>
<tr>
<td>D116593</td>
<td>Add clearAndCopyFrom to IntegerPolyhedron</td>
<td>Groverkss</td>
<td>arjung, bondhugula, fynse</td>
<td>Wed, Jan 5, 6:09 PM</td>
</tr>
<tr>
<td>D116590</td>
<td>Remove dependency on IR for Simplex</td>
<td>Groverkss</td>
<td>arjung, bondhugula, fynse</td>
<td>Mon, Jan 3, 11:00 AM</td>
</tr>
<tr>
<td>D116311</td>
<td>Move LinearTransform to Presburger/</td>
<td>Groverkss</td>
<td>arjung, bondhugula, fynse, vinayakapolygonal</td>
<td>Mon, Jan 3, 5:50 AM</td>
</tr>
<tr>
<td>D116287</td>
<td>Use IntegerPolyhedron in Simplex instead of FlatAffineConstraints</td>
<td>Groverkss</td>
<td>arjung, bondhugula, fynse, vinayakapolygonal</td>
<td>Mon, Dec 27, 1:39 PM</td>
</tr>
<tr>
<td>D116269</td>
<td>Move <code>print()</code> and <code>dump()</code> from FlatAffineConstraints to IntegerP...</td>
<td>Groverkss</td>
<td>arjung, Bondhugula, fynse, vinayakapolygonal</td>
<td>Mon, Dec 27, 1:18 PM</td>
</tr>
<tr>
<td>D116627</td>
<td>Add forgotten directory Support to unittests cmake</td>
<td>Groverkss</td>
<td>ipienaar</td>
<td>Mon, Dec 27, 9:13 AM</td>
</tr>
<tr>
<td>D116285</td>
<td>Move presburger math from FlatAffineConstraints to Presburger/dir...</td>
<td>Groverkss</td>
<td>arjung, bondhugula, fynse, mehdi amini</td>
<td>Sun, Dec 26, 7:09 PM</td>
</tr>
<tr>
<td>D114674</td>
<td>Move Presburger Math from FlatAffineConstraints to Presburger/dir...</td>
<td>Groverkss</td>
<td>arjung, bondhugula, fynse, mehdi amini</td>
<td>Sun, Dec 26, 1:31 PM</td>
</tr>
</tbody>
</table>

Source: https://reviews.llvm.org/differential/query/cP8ij2thxgW/
FPL - A Growing Developer Community

Arjun Pitchanthan  Kunwar Grover  Abhinav Menon  Bharathi Ramana Joshi
Discussion, where are we heading?
Polyhedral Algorithms are too expensive to Scale?
Is Polyhedral Compilation More Than Polyhedral Loop Scheduling?
How Can We Mix Polyhedral Compilation and Real-World Compilers?
Conclusion