Polly
Polyhedral Optimizations for LLVM

Tobias Grosser - Hongbin Zheng - Raghesh Aloor
Andreas Simbürger - Armin Grösslinger - Louis-Noël Pouchet

April 03, 2011
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?
LLVM

- Compiler Infrastructure
- Low Level Intermediate Language
  - SSA, Register Machine
  - Language and Target Independent
  - Integrated SIMD Support
- Large Set of Analysis and Optimization
- Optimizations Compile, Link, and Run Time
- JIT Infrastructure
- Very convenient to work with
Classical Compilers:
  ▶ clang → C/C++/Objective-C
  ▶ Mono → .Net
  ▶ OpenJDK → Java
  ▶ drakonegg → C/C++/Fortran/ADA/Go
  ▶ Others → Ruby/Python/Lua

GPGPU: PTX backend
OpenCL (NVIDIA, AMD, INTEL, Apple, Qualcomm, ...)

Graphics Rendering
(VMWare Gallium3D/LLVMPipe/LunarGlass/Adobe Hydra)

Web
  ▶ ActionScript (Adobe)
  ▶ Google Native Client

HLS (C-To-Verilog, LegUp, UCLA - autoESL)

Source to Source: LLVM C-Backend
The Architecture

Transformations

* Classical loop transformations (Blocking, Interchange, Fusion, ...)
* Expose parallelism
* Dead instruction elimination / Constant propagation

Dependency Analysis

OpenMP Backend

SIMD Backend

Code Generation

PTX Backend

LLVM IR

LLVM IR

SCoP Detection & LLVM to Poly

PSCoP

JSCoP Import/Export

Manual Optimization / LooPo / Pluto / PoCC+Pluto / ...
The SCoP - Classical Definition

\begin{align*}
\text{for } i &= 1 \text{ to } (5n + 3) \\
&\quad \text{for } j = n \text{ to } (4i + 3n + 4) \\
&\quad \quad A[i-j] = A[i] \\
&\quad \quad \text{if } i < (n - 20) \\
&\quad \quad \quad A[i+20] = j
\end{align*}

- Structured control flow
  - Regular for loops
  - Conditions

- Affine expressions in:
  - Loop bounds, conditions, access functions

- Side effect free
AST based frameworks

What about:

- Goto-based loops
- C++ iterators
- C++0x foreach loop

Common restrictions

- Limited to subset of C/C++
- Require explicit annotations
- Only canonical code
- Correct? (Integer overflow, Operator overloading, ...)

Polly - Polyhedral Optimizations for LLVM
Semantic SCoP

Thanks to LLVM Analysis and Optimization Passes:

**SCoP - The Polly way**

- **Structured control flow**
  - Regular for-loops → Anything that acts like a regular for loop
  - Conditions
- **Affine expressions** → Expressions that calculate an affine result
- **Side effect free known**
- **Memory accesses through arrays** → Arrays + Pointers
Valid SCoPs

**do..while loop**

```c
i = 0;
do {
    int b = 2 * i;
    int c = b * 3 + 5 * i;
    A[c] = i;
    i += 2;
} while (i < N);
```

**pointer loop**

```c
int A[1024];

void pointer_loop () {
    int *B = A;
    while (B < &A[1024]) {
        *B = 1;
        ++B;
    }
}
```
Polyhedral Representation - SCoP

- SCoP = (Context, [Statement])
- Statement = (Domain, Schedule, [Access])
- Access = ("read" | "write" | "may_write", Relation)

Interesting:
- Data structures are integer sets/maps
- Domain is read-only
- Schedule can be partially affine
- Access is a relation
- Access can be may_write
Applying transformations

\[ \mathcal{D} = \{ \text{Stmt}[i,j] : 0 \leq i < 32 \land 0 \leq j < 1000 \} \]
\[ \mathcal{S} = \{ \text{Stmt}[i,j] \rightarrow [i,j] \} \]

\[ \mathcal{S}' = \mathcal{S} \]

for (i = 0; i < 32; i++)
    for (j = 0; j < 1000; j++)
        A[i][j] += 1;
Applying transformations

- \( \mathcal{D} = \{ Stmt[i,j] : 0 \leq i < 32 \land 0 \leq j < 1000 \} \)
- \( S = \{ Stmt[i,j] \rightarrow [i,j] \} \)
- \( \mathcal{T}_{\text{Interchange}} = \{ [i,j] \rightarrow [j,i] \} \)
- \( S' = S \circ \mathcal{T}_{\text{Interchange}} \)

for (j = 0; j < 1000; j++)
  for (i = 0; i < 32; i++)
    A[i][j] += 1;
Applying transformations

- $D = \{Stmt[i, j] : 0 \leq i < 32 \land 0 \leq j < 1000\}$
- $S = \{Stmt[i, j] \rightarrow [i, j]\}$
- $T_{Interchange} = \{[i, j] \rightarrow [j, i]\}$
- $T_{StripMine} = \{[i, j] \rightarrow [i, jj, j] : jj \mod 4 = 0 \land jj \leq j < jj + 4\}$
- $S' = S \circ T_{Interchange} \circ T_{StripMine}$

```c
for (j = 0; j < 1000; j++)
    for (ii = 0; ii < 32; ii+=4)
        for (i = ii; i < ii+4; i++)
            A[i][j] += 1;
```

Polly - Polyhedral Optimizations for LLVM

April 03, 2011 14 / 27
JSCoP - Exchange format

Specification:
- Representation of a SCoP
- Stored as JSON text file
- Integer Sets/Maps use ISL Representation

Benefits:
- Can express modern polyhedral representation
- Can be imported easily (JSON bindings readily available)
- Is already valid Python
JSCoP - Example

{
    "name": "body => loop.end",
    "context": "[N] -> { []: N >= 0 }",
    "statements": [{
        "name": "Stmt",
        "domain": "[N] -> { Stmt[i0, i1] : 0 <= i0, i1 <= N }",
        "schedule": "[N] -> { Stmt[i0, i1] -> scattering[i0, i1] }",
        "accesses": [{
            "kind": "read",
            "relation": "[N] -> { Stmt[i0, i1] -> A[o0] }",
        },
        {
            "kind": "write",
            "relation": "[N] -> { Stmt[i0, i1] -> C[i0][i1] }",
        }]
    }]
}
Optimized Code Generation

- Automatically detect parallelism,
- after code generation
- Automatically transform it to:
  - OpenMP, if loop
    - is parallel
    - is not surrounded by any other parallel loop
  - Efficient SIMD instructions, if loop
    - is innermost
    - is parallel
    - has constant number of iterations
Generation of Parallel Code

```
for (i = 0; i < N; i++)
  for (j = 0; j < N; j++)
    for (kk = 0; kk < 1024; kk++)
      for (k = kk; k < kk+4; k++)
        A[j][k] += 9;
  for (j = 0; j < M; j++)
    B[i] = B[i] * i;
```
Generation of Parallel Code

```c
for (i = 0; i < N; i++)
    S = {[i, 0, j, ...]: 0 <= i, j < N}
for (j = 0; j < N; j++)
    for (kk = 0; kk < 1024; kk++)
        for (k = kk; k < kk+4; k++)
            A[j][k] += 9;
    S = {[i, 1, j, ...]: 0 <= i, j < N}
for (j = 0; j < M; j++)
    B[i] = B[i] * i;
```
for (i = 0; i < N; i++)
  #pragma omp parallel
  for (j = 0; j < N; j++)
    for (kk = 0; kk < 1024; kk++)
      for (k = kk; k < kk+4; k++)
        A[j][k] += 9;
  for (j = 0; j < M; j++)
    B[i] = B[i] * i;
for (i = 0; i < N; i++)
  #pragma omp parallel
  for (j = 0; j < N; j++)
    for (kk = 0; kk < 1024; kk++)
      for (k = kk; k < kk+4; k++)
        A[j][k] += 9;
  for (j = 0; j < M; j++)
    B[i] = B[i] * i;
Generation of Parallel Code

for (i = 0; i < N; i++)
  #pragma omp parallel
  for (j = 0; j < N; j++)
    for (kk = 0; kk < 1024; kk++)
      A[j][kk:kk+3] += [9,9,9,9];
  for (j = 0; j < M; j++)
    B[i] = B[i] * i;
Optimizing of Matrix Multiply

32x32 double, Transposed matrix Multiply, C[i][j] += A[k][i] * B[j][k];

Intel® Core® i5 @ 2.40GH, polly and clang from 23. March 2011
Pluto Tiling on Polybench

Polybench 2.0 (large data set), Intel® Xeon® X5670 @ 2.93GHz
polly and clang from 23. March 2011
Current Status

Transformations

- Classical loop transformations (Blocking, Interchange, Fusion, ...)
- Expose parallelism
- Dead instruction elimination / Constant propagation

Dependency Analysis

SCoP Detection & LLVM to Poly

PSCoP

OpenMP Backend
SIMD Backend
Code Generation
PTX Backend

LLVM IR

Usable for experiments
Planned
Under Construction

Manual Optimization / LooPo / Pluto / PoCC+Pluto / ...
Future Work

- Increase general coverage
- Expose more SIMDization opportunities
- Modifiable Memory Access Functions
- GPU code generation
Polly - Conclusion

- Automatic SCoP Extraction
- Non canonical SCoPs
- Modern Polyhedral Representation
- JSCoP - Connect External Optimizers
- OpenMP/SIMD/PTX backends

What features do we miss to apply YOUR optimizations?

http://wiki.llvm.org/Polly