A Framework for Automatic OpenMP Code Generation

Raghesh A (CS09M032)

Guide: **Dr. Shankar Balachandran**

May 2nd, 2011

Outline

- Introduction
- The Polyhedral Model
- LLVM
- Polly
- OpenMP Code Generation in Polly
- Testing with PolyBench
- Conclusion and Future Work
- Setting up the environment
- Various Tools Used in Polyhedral Community

An Example

Source code

```
float A[1024];

int main()
{
  int i, j;
  for (i = 0; i < 1024; i++)
    for (j = 0; j < 5000000; j++)
      A[i] += j;
}
```

An Example

LLVM-IR Sequential

```
define i32 @main() nounwind {
entry:
  %retval = alloca i32, align 4
  %i = alloca i32, align 4
  %j = alloca i32, align 4
   store i32 0, i32* %retval
   store i32 0, i32* %i, align 4
   br label %for.cond

for.cond:
  %tmp = load i32* %i, align 4
  %cmp = icmp slt i32 %tmp, 1024
   br i1 %cmp, label %for.body,
               label %for.end12

for.body:
   store i32 0, i32* %j, align 4
   br label %for.cond1

for.cond1:
  %tmp2 = load i32* %j, align 4
  %cmp3 = icmp slt i32 %tmp2, 5000000
   br i1 %cmp3, label %for.body4,
                label %for.end
```

LLVM-IR Sequential

```
for.body4:
  %tmp5 = load i32* %j, align 4
  %conv = sitofp i32 %tmp5 to float
  %tmp6 = load i32* %i, align 4
  %arrayidx = getelementptr inbounds
               [1024 x float]*
               @A, i32 0, i32 %tmp6
  %tmp7 = load float* %arrayidx
  %add = fadd float %tmp7, %conv
   store float %add, float* %arrayidx
   br label %for.inc
for.inc:
  %tmp8 = load i32* %j, align 4
  %inc = add nsw i32 %tmp8, 1
   store i32 %inc, i32* %j, align 4
   br label %for.cond1
for.end:
   br label %for.inc9
for.inc9:
  %tmp10 = load i32* %i, align 4
  %inc11 = add nsw i32 %tmp10, 1
   store i32 %inc11, i32* %i, align 4
   br label %for.cond
for.end12:
  %0 = load i32* %retval
   ret i32 %0
}
```

An Example

Source code with OpenMP pragmas

```
float A[1024];

int main()
{
  int i, j;
  #pragma omp parallel for \
  schedule(runtime) private(j)
  for (i = 0; i < 1024; i++)
    for (j = 0; j < 5000000; j++)
      A[i] += j;
}
```

An Example

LLVM-IR Manual

```
define i32 @main() nounwind {
entry:
  %retval = alloca i32
  %i = alloca i32
  %j = alloca i32
  %"alloca point" = bitcast i32 0 to i32
  call void
       @GOMP_parallel_loop_runtime_start(
       void (i8*)* @main.omp_fn.0,
       i8* null, i32 0, i32 0,
       i32 1024, i32 1) nounwind
  call void
       @main.omp_fn.0(i8* null) nounwind
  call void
       @GOMP_parallel_end() nounwind
  br label %return
'
return:
; preds = %entry
  %retval1 = load i32* %retval
; <i32> [#uses=1]
  ret i32 %retval1
}
```

LLVM-IR Manual

```
define internal void @main.omp_fn.0(
          i8* %.omp_data_i) nounwind {
entry:

  <some initializations here>

  store i8* %.omp_data_i,
          i8** %.omp_data_i_addr
  br label %bb
bb:
  %1 = call zeroext i8
        @GOMP_loop_runtime_next(
         i32* %.istart0.3,
         i32* %.iend0.4) nounwind
  %toBool = icmp ne i8 %1, 0
  br i1 %toBool, label %bb2, label %bb1
bb1:
  call void @GOMP_loop_end_nowait()
  nounwind
  br label %return
bb2:
  <body of the loop>
}
```

An Example

LLVM-IR Automatic

```
@A = common global
     [1024 x float]
     zeroinitializer, align 4
define i32 @main() nounwind {
<some initialization>
pollyBB:
  %insertInst = zext i1 true to i16
  %omp.userContext = alloca
     %main.omp_subfn.omp.userContext
  %0 = getelementptr inbounds
       %main.omp_subfn.omp.userContext*
       %omp.userContext, i32 0, i32 0
  store [1024 x float]*
        @A, [1024 x float]** %0
  call void
      @GOMP_parallel_loop_runtime_start(
      void (i8*)* @main.omp_subfn,
      i8* %omp_data, i32 0, i32 0,
      i32 1024, i32 1)
  call void @main.omp_subfn(i8*%omp_data)
  call void @GOMP_parallel_end()
  br label %for.end12.region
}
```

LLVM-IR Automatic

```
define internal void
       @main.omp_subfn(
       i8* %omp.userContext) {
omp.setup:

  <some initialization>

omp.exit:
   call void @GOMP_loop_end_nowait()
   ret void
omp.checkNext:
  %2 = call i8
       @GOMP_loop_runtime_next(
       i32* %omp.lowerBoundPtr,
       i32* %omp.upperBoundPtr)
omp.loadIVBounds:

<body of the loop>

}
```

Necessary Background

- Parallelism in programs
  - Parallelism and locality
  - Realizing parallelism
- Auto parallelization
- The polyhedral model
- LLVM
- Polly

Necessary Background

- Parallelism in programs
    - Parallelism and locality
    - Realizing parallelism
- Auto parallelization
- The polyhedral model
- LLVM
- Polly

Workdone: "OpenMP Code Generation in Polly"

The Polyhedral Model

- Examples for transformations with polyhedral model
    - Transformation for improving data locality

The Polyhedral Model

- Examples for transformations with polyhedral model
    - Transformation for improving data locality

```
for(i = 1; i <= 10; i++)
  A[i] = 10;
for(j = 6; j <= 15; j++)
  A[j] = 15;
```

The Polyhedral Model

- Examples for transformations with polyhedral model
    - Transformation for improving data locality

```
for(i = 1; i <= 10; i++)
  A[i] = 10;
for(j = 6; j <= 15; j++)
  A[j] = 15;
```

```
for(i = 1; i <= 5; i++)
  A[i] = 10;
for(j = 6; j <= 15; j++)
  A[j] = 15;
```

The Polyhedral Model

- Examples for transformations with polyhedral model
    - Transformation for improving data locality

```
for(i = 1; i <= 10; i++)
  A[i] = 10;
for(j = 6; j <= 15; j++)
  A[j] = 15;
```

```
for(i = 1; i <= 5; i++)
  A[i] = 10;
for(j = 6; j <= 15; j++)
  A[j] = 15;
```

    - Scalar expansion

The Polyhedral Model

- Examples for transformations with polyhedral model
    - Transformation for improving data locality

```
for(i = 1; i <= 10; i++)
  A[i] = 10;
for(j = 6; j <= 15; j++)
  A[j] = 15;
```

```
for(i = 1; i <= 5; i++)
  A[i] = 10;
for(j = 6; j <= 15; j++)
  A[j] = 15;
```

- Scalar expansion

```
for (i = 0; i < 8; i++)
  sum += A[i];
```

The Polyhedral Model

- Examples for transformations with polyhedral model
  - Transformation for improving data locality

```
for(i = 1; i <= 10; i++)
  A[i] = 10;
for(j = 6; j <= 15; j++)
  A[j] = 15;
```

```
for(i = 1; i <= 5; i++)
  A[i] = 10;
for(j = 6; j <= 15; j++)
  A[j] = 15;
```

  - Scalar expansion

```
for (i = 0; i < 8; i++)
  sum += A[i];
```

```
<create and initialize an array 'tmp'>
for (i = 0; i < 8; i++)
    tmp[i % 4] += A[i];
sum = tmp[0] + tmp[1] + tmp[2] + tmp[3];
```

The Polyhedral Model

- Examples for transformations with polyhedral model
  - Transformation for improving data locality

```
for(i = 1; i <= 10; i++)
  A[i] = 10;
for(j = 6; j <= 15; j++)
  A[j] = 15;
```

```
for(i = 1; i <= 5; i++)
  A[i] = 10;
for(j = 6; j <= 15; j++)
  A[j] = 15;
```

  - Scalar expansion

```
for (i = 0; i < 8; i++)
  sum += A[i];
```

```
<create and initialize an array 'tmp'>
for (i = 0; i < 8; i++)
    tmp[i % 4] += A[i];
sum = tmp[0] + tmp[1] + tmp[2] + tmp[3];
```

```
parfor (ii = 0; ii < 4; ii++)
  tmp[ii] = 0;
  for (i = ii * 2; i < (ii+1) * 2; i++)
    tmp[ii] += A[i];
sum = tmp[0] + tmp[1] + tmp[2] + tmp[3];
```

Polyhedral representation of programs

- Iteration domain
- Schedule
- Access function

Dynamic instances of each statement is represented as an integer point in statement's polyhedron

- Why not AST?
    - Dynamic instances of statements not captured
    - Rigid data structure
    - Less expressive than polyhedral model

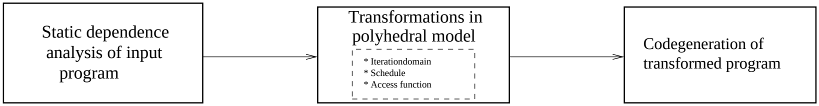

Static dependence analysis of input program
Transformations in polyhedral model
* Iterationdomain
* Schedule
* Access function
Codegeneration of transformed program

Figure: Transformation in polyhedral model

Iteration domain

```
for (int i = 2; i <= N; i++)
  for (int j = 2; j <= N; j++)
    A[i] = 10; // S1
```

```
for (int i = 2; i <= 6; i++)
  for (int j = 2; j <= 6; j++)
    if(i <= j)
      A[i] = 10; // S2
```

Iteration domain

```
for (int i = 2; i <= N; i++)
  for (int j = 2; j <= N; j++)
    A[i] = 10; // S1
```

```
for (int i = 2; i <= 6; i++)
  for (int j = 2; j <= 6; j++)
    if(i <= j)
      A[i] = 10; // S2
```

Iteration domain for **S**1 is
$D_{S1} = \{(i,j) \epsilon Z^2 \mid 2 \leq i \leq N \wedge 2 \leq j \leq N\}$

Iteration domain for **S**2 is
$D_{S2} = \{(i,j) \epsilon Z^2 \mid 2 \leq i \leq 6 \wedge 2 \leq j \leq 6 \wedge i \leq j\}$

Iteration domain

```
for (int i = 2; i <= N; i++)
  for (int j = 2; j <= N; j++)
    A[i] = 10; // S1
```

```
for (int i = 2; i <= 6; i++)
  for (int j = 2; j <= 6; j++)
    if(i <= j)
      A[i] = 10; // S2
```

Iteration domain for **S**1 is
$D_{S1} = \{(i,j) \epsilon Z^2 \mid 2 \leq i \leq N \wedge 2 \leq j \leq N\}$

Iteration domain for **S**2 is
$D_{S2} = \{(i,j) \epsilon Z^2 \mid 2 \leq i \leq 6 \wedge 2 \leq j \leq 6 \wedge i \leq j\}$

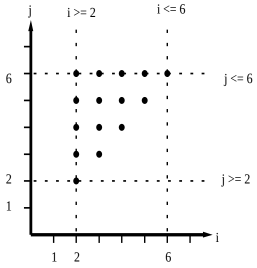


Figure: Graphical representation of iteration domain(S2)

Schedule

- Scattering function

Schedule

- Scattering function

```
for (int i = 2; i <= 4; i++)
  for (int j = 2; j <= 4; j++)
    P[i][j] = A[i] * B[j] ; // S3
```

Examples:

$\phi_{S3}(i,j) = (i,j)$

$\phi_{S3}(i,j) = (j,i)$

Schedule

- Scattering function

```
for (int i = 2; i <= 4; i++)
  for (int j = 2; j <= 4; j++)
    P[i][j] = A[i] * B[j] ; // S3
```

Examples:

$\phi_{S3}(i,j) = (i,j)$

$\phi_{S3}(i,j) = (j,i)$

Code generated by Cloog for $\phi_{S3}(i,j) = (j,i)$

```
for (t1 = 2; t1 <= 4; t1++) {
  for (t2 = 2; t2 <= 4; t2++) {
    i = t2; j = t1;
    P[i+j] += A[i] + B[j];
  }
}
```

Loops are **interchanged** here by applying this transformation

Access function

A[i+j][i+N]

Array access function: $F_A(i, j) = (i + j, i + N)$

Change array access function for better locality

SCoP - Static Control Part

Example for SCoP

```
for (i = 0; i < 5*N; i++)
  for (j = N; j < 3*i + 5*N + 6; j++)
    A[i-j] = A[i];
  if (i < N - 10)
    A[i + 20] = j;
```

- Structured control flow
    - Regular for loops
    - Conditions
- Affine expressions in:
    - Loop bounds
    - Conditions
    - Access functions
- Side effect free(Pure functions)

LLVM

- LLVM (Low Level Virtual Machine)
    - Framework for implementing compilers
    - Common low level code repersentation
    - Lifelong analysis and transformation of programs
- LLVM relaxes SCoP constraints -> more SCoPs are detected
    - ~~Regular for loops~~ -> Anything that acts like a regular for loop
    - ~~Affine experssions~~ -> Expressions that calculates an affine result
    - Side effect ~~free~~ known
    - Memory access through ~~arrays~~ arrays + pointers
- Independent of programming language

Polly I

- Polly (Polyhedral Optimization in LLVM)
  - Implementing Polyhedral Optimization in LLVM
  - Effort towards Auto Parallelism in programs.
- Implementation
  - LLVM-IR to polyhedral model
    - Region-based SCoP detection
    - Semantic SCoPs
  - Polyhedral model
    - The integer set library
    - Composable polyhedral transformations
    - Export/Import
  - Polyhedral model to LLVM-IR
- Related work
  - gcc Graphite

Polly II

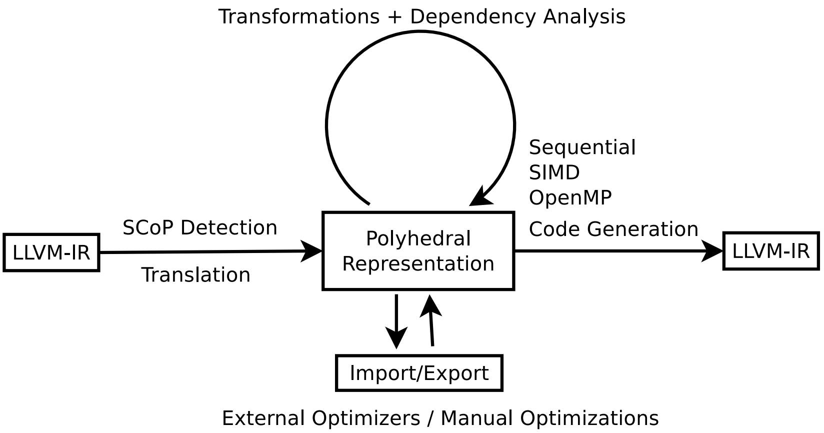

Transformations + Dependency Analysis
Sequential
SIMD
OpenMP
Code Generation
LLVM-IR
SCoP Detection
Translation
Polyhedral Representation
LLVM-IR
Import/Export
External Optimizers / Manual Optimizations

Figure: Architecture of Polly

OpenMP Code Generation in Polly

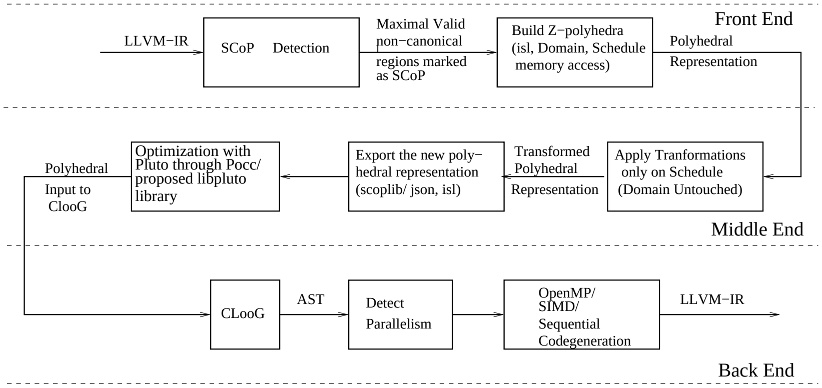

Front End
LLVM−IR
SCoP Detection
Maximal Valid non−canonical regions marked as SCoP
Build Z−polyhedra (isl, Domain, Schedule memory access)
Polyhedral Representation
Apply Tranformations only on Schedule (Domain Untouched)
Transformed Polyhedral Representation
Export the new poly−hedral representation (scoplib/ json, isl)
Optimization with Pluto through Pocc/ proposed libpluto library
Polyhedral Input to ClooG
Middle End
CLooG
AST
Detect Parallelism
OpenMP/ SIMD/ Sequential Codegeneration
LLVM−IR
Back End

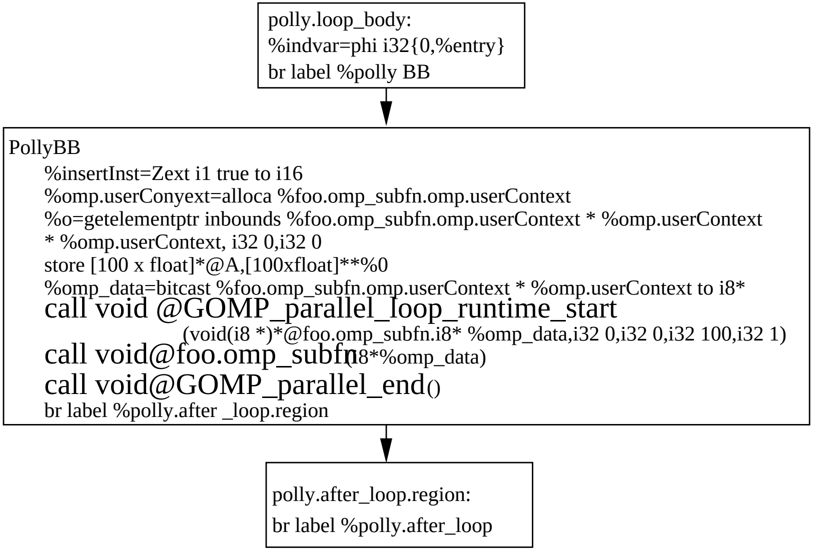
Figure: Detailed control flow in Polly

OpenMP Code Generation in Polly

- Code generation pass in Polly
- Detecting parallelism in Polly
- Generating OpenMP library calls

```
for (int i = 0; i <= N; i++)
  A[i] = 1 ;
```

OpenMP Code Generation in Polly



```
polly.loop_body:
%indvar=phi i32{0,%entry}
br label %polly BB
```

```
PollyBB
      %insertInst=Zext i1 true to i16
      %omp.userConyext=alloca %foo.omp_subfn.omp.userContext
      %o=getelementptr inbounds %foo.omp_subfn.omp.userContext * %omp.userContext
      * %omp.userContext, i32 0,i32 0
      store [100 x float]*@A,[100xfloat]**%0
      %omp_data=bitcast %foo.omp_subfn.omp.userContext * %omp.userContext to i8*
      call void @GOMP_parallel_loop_runtime_start
                  (void(i8 *)*@foo.omp_subfn.i8* %omp_data,i32 0,i32 0,i32 100,i32 1)
      call void@foo.omp_subfn(i8*%omp_data)
      call void@GOMP_parallel_end()
      br label %polly.after _loop.region
```

```
polly.after_loop.region:
br label %polly.after_loop
```

Figure: CFG showing sequence of OpenMP library calls

OpenMP Code Generation in Polly

- Support for inner loops

```
for (int i = 0; i < M; i++)
  for (int j = 0; j < N; j++)
    A[i][j] = A[i-1][j] + B[i-1][j];
```

Surrounding induction variables and parameters need to be passed to the subfunction

OpenMP Code Generation in Polly

- Support for inner loops

```
for (int i = 0; i < M; i++)
  for (int j = 0; j < N; j++)
    A[i][j] = A[i-1][j] + B[i-1][j];
```

  Surrounding induction variables and parameters need to be passed to the subfunction
- Dealing with memory references

```
#define N 10
void foo() {
  float A[N];
  for (int i=0; i < N; i++)
    A[i] = 10;
  return;
}
```

  - Adding and extracting memory references

OpenMP Code Generation in Polly

- Enabling OpenMP code generation in Polly

```
export LIBPOLLY=<path to cmake>/lib/LLVMPolly.so
pollycc -fpolly -fparallel a.c

OR

# Generate the LLVM-IR files from source code.
clang -S -emit-llvm a.c
alias opt="opt -load $LIBPOLLY
# Apply optimizations to prepare code for polly
opt -S -mem2reg -loop-simplify -indvars a.c -o a.preopt.ll
# Generate OpenMP code with Polly
opt -S -polly-codegen -enable-polly-openmp a.preopt.ll -o a.ll
# Link with libgomp
llc a.ll -o a.s
llvm-gcc a.s -lgomp
```

- OpenMP testcases
  - Polly follows LLVM testing infrastrcutre

Testing with PolyBench

- PolyBench

  Benchmarks from

  - linear algebra
  - datamining
  - stencil computation
  - solver and manipulation algorithms operating on matrices

Experimental results

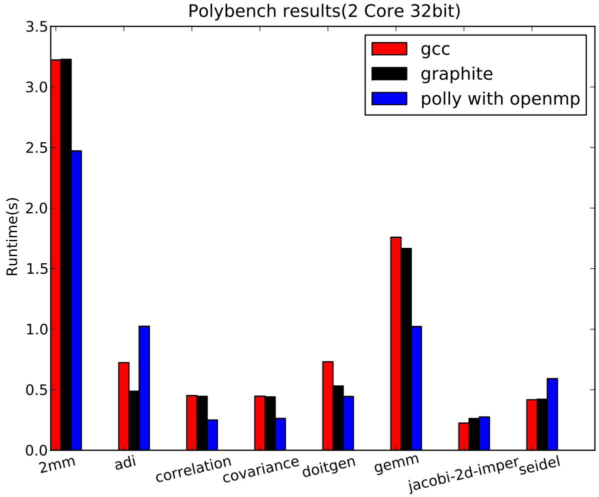


Figure: Performance comparison(2 core 32 bit)

Experimental results

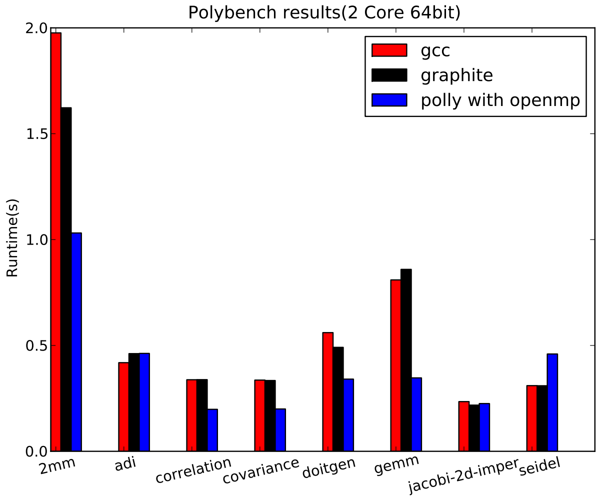


Figure: Performance comparison(2 core 64bit)

Experimental results

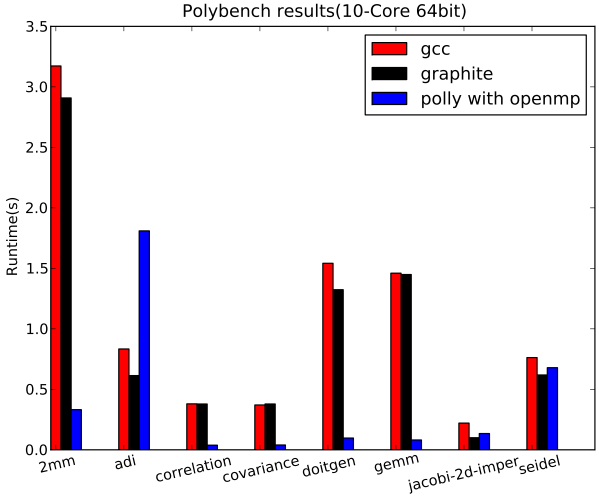


Figure: Performance comparison(10-core 64 bit)

Conclusion and Future Work

- Conclusion
- Support for memory access transformations in Polly
- Increasing coverage of Polly
  - Increasing SCoP coverage
  - Increasing the system coverage
- Integrating profile guided optimization into Polly

Setting up the environment

- CLooG
- PoCC
- Scoplib
- Building LLVM with Polly

Various Tools Used in Polyhedral Community

- ClooG
- PLUTO
- VisualPolylib