A Framework for Automatic OpenMP Code Generation

Raghes A (CS09M032)

Guide: Dr. Shankar Balachandran

May 2nd, 2011
Outline

• The Framework
• An Example
• Necessary Background
• Polyhedral Model
• SCoP - Static Control Part
• LLVM in Polly
• Polly
• OpenMP Code Generation in Polly
• Experimental Results
• Conclusion
• Future Work
The Framework

- Source code
  - Clang
    - Sequential LLVM IR
      - Execution 1
  - Polly OpenMP (Automatic)
    - Parallel LLVM IR1
      - Execution 2
  - Manually Annotated Source code
    - Parallel LLVM IR2
      - Execution 3
    - LLVM-gcc (Manual)
An Example

Source code

```c
float A[1024];

int main()
{
    int i;
    for (i = 0; i < 1024; i++)
        A[i] += 10;
}
```
An Example

LLVM-IR Sequential

define i32 @main() nounwind {

entry:

%retval = alloca i32, align 4
%i = 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.end

for.body:

%tmp1 = load i32* %i, align 4
%arrayidx = getelementptr inbounds [1024 x float]* @A, i32 0, i32 %tmp1
%tmp2 = load float* %arrayidx
%add = fadd float %tmp2, 1.000000e+01
store float %add, float* %arrayidx
br label %for.inc

for.end:

%0 = load i32* %retval
ret i32 %0
}
An Example

Source code with OpenMP pragmas

```c
float A[1024];

int main()
{
    int i;
    #pragma omp parallel for \ schedule(runtime)
    for (i = 0; i < 1024; i++)
        A[i] += 10;
}
```
define i32 @main() nounwind {
  entry:
  %retval = alloca i32
  %i = 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
}

define internal void @main.omp_fn.0(i8* %.omp_data_i) nounwind {
  entry:
    <some initialization>
  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>
}
define i32 @main() nounwind {
  entry:
    br label %for.cond
  for.cond:
    %i.0 = phi i32 [ 0, %entry ]
    br label %pollyBB
  pollyBB:
    <some initialization>
    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.end.region
}

define internal void @main.omp_subfn(
    i8* %omp.userContext) {
  omp.setup:
    <some initialization>
    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

```c
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

```c
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

```c
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

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

- Scalar expansion

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

```plaintext
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

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

- Scalar expansion

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

```c
create and initialize an array 'tmp'
for (i = 0; i < 8; i++)
    tmp[i % 4] += A[i];
```
The Polyhedral Model

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

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

- Scalar expansion

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

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

```c
parfor (ii = 0; ii < 4; ii++)
    tmp[ii] = 0;
for (i = ii * 2; i < (ii+1) * 2; i++)
    tmp[ii] += A[i];
```
Polyhedral Representation of Programs

- Iteration domain
- Schedule
- Access function
Polyhedral Representation of Programs

- Iteration domain
- Schedule
- Access function

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

- Transformation in polyhedral model
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

Iteration domain for S1 is
\[
D_{S1} = \{(i, j) \in \mathbb{Z}^2 | 2 \leq i \leq N \land 2 \leq j \leq N\}
\]

Iteration domain for S2 is
\[
D_{S2} = \{(i, j) \in \mathbb{Z}^2 | 2 \leq i \leq 6 \land 2 \leq j \leq 6 \land i \leq j\}
\]
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 S1 is
\[ D_{S1} = \{(i,j) \in \mathbb{Z}^2 \mid 2 \leq i \leq N \land 2 \leq j \leq N\} \]

Iteration domain for S2 is
\[ D_{S2} = \{(i,j) \in \mathbb{Z}^2 \mid 2 \leq i \leq 6 \land 2 \leq j \leq 6 \land i \leq j\} \]

Figure: Graphical representation of iteration domain(S2)
• Scattering function

```c
for (i = 0; i < 32; i++)
    for (j = 0; j < 1000; j++)
        A[i][j] += 1; // S3
```

Assigning execution date for each statement instance. Instances with same execution dates can be run in parallel.
Schedule

- Scattering function

```cpp
for (i = 0; i < 32; i++)
    for (j = 0; j < 1000; j++)
        A[i][j] += 1; // S3
```

Assigning execution date for each statement instance. Instances with same execution dates can be run in parallel.

Examples:

\[
\begin{align*}
\phi_{S3}(i, j) &= (i, j) \\
\phi'_{S3}(i, j) &= (j, i) \\
\phi''_{S3}(i, j) &= \{(i, jj, j) : jj \mod 4 = 0 \land jj \leq j < jj + 4\}
\end{align*}
\]
• Scattering function

```c
for (i = 0; i < 32; i++)
    for (j = 0; j < 1000; j++)
        A[i][j] += 1;  // S3
```

Assigning execution date for each statement instance. Instances with same execution dates can be run in parallel

Examples:
- $\phi_{S3}(i, j) = (i, j)$
- $\phi'_{S3}(i, j) = (j, i)$
- $\phi''_{S3}(i, j) = \{(i, jj, j) : jj \mod 4 = 0 \land jj \leq j < jj + 4\}$

Code generated for $\phi'_{S3}$

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

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

- Scattering function

```c
for (i = 0; i < 32; i++)
    for (j = 0; j < 1000; j++)
        A[i][j] += 1; // S3
```

Assigning execution date for each statement instance. Instances with same execution dates can be run in parallel.

Examples:

\[
\phi_{S3}(i,j) = (i,j) \quad \phi_{S3}'(i,j) = (j,i) \\
\phi_{S3}''(i,j) = \{(i,jj,j) : jj \mod 4 = 0 \land jj \leq j < jj + 4\}
\]

Code generated for \(\phi_{S3}' \circ \phi_{S3}''\)

```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;
```

Loops are **stripmined** here by applying this transformation.
A[i+j][i+N]

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

**Change array access function for better locality**
Example for SCoP

```c
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 in Polly

- LLVM (Low Level Virtual Machine)
  - Framework for implementing compilers
  - Common low level code representation
  - 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 expressions -> Expressions that calculates an affine result
  - Side effect free known
  - Memory access through arrays + pointers
- Independent of programming language
• 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
    • Polyhedral transformations
    • Export/Import
  • Polyhedral model to LLVM-IR

• Related work
  • gcc Graphite
Figure: Architecture of Polly
OpenMP Code Generation in Polly

**Figure:** Detailed control flow in Polly
OpenMP Code Generation in Polly

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

```c
for (int i = 0; i <= N; i++)
    A[i] = 1;
```
OpenMP Code Generation in Polly

```assembly
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

```c
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.
• Support for inner loops

```c
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

```c
#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

```bash
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 infrastructure
Testing

- GCC Compile farm
• GCC Compile farm

A simple test case

```c
float A[1024];

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

- GCC Compile farm

### A simple test case

```c
float A[1024];

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

<table>
<thead>
<tr>
<th></th>
<th>Serial Execution</th>
<th>Automatic Parallelization(Polly)</th>
<th>Manual Parallelization(GCC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Core 2 Duo(32 Bit OS)</td>
<td>9.509s</td>
<td>4.852s</td>
<td>4.835s</td>
</tr>
<tr>
<td>Intel Core 2 Duo(64 Bit OS)</td>
<td>6.40s</td>
<td>3.32s</td>
<td>3.50s</td>
</tr>
<tr>
<td>Intel Core i5(64 Bit OS)</td>
<td>6.96s</td>
<td>3.78s</td>
<td>3.75s</td>
</tr>
<tr>
<td>AMD Engineering Sample(24 Core)(64 Bit OS)</td>
<td>17.039s</td>
<td>0.757s</td>
<td>0.796s</td>
</tr>
</tbody>
</table>

**Table:** Performance Comparison

Automatic OpenMP code generation in Polly gives similar results as GCC with OpenMP pragmas
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)
Figure: Performance comparison (10-core 64 bit)
Improving seidel’s performance

- Polly + Pluto + OpenMP + setting OMP_SCHEDULE=guided

<table>
<thead>
<tr>
<th></th>
<th>Serial Execution</th>
<th>Polly + OpenMP</th>
<th>Polly PLUTO + OpenMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Core 32 Bit</td>
<td>0.417174s</td>
<td>0.591673s</td>
<td>0.348909s</td>
</tr>
<tr>
<td>2 Core 64 Bit</td>
<td>0.310160s</td>
<td>0.459641s</td>
<td>0.254605s</td>
</tr>
</tbody>
</table>

**Table:** Performance improvement of seidel

<table>
<thead>
<tr>
<th>No of threads</th>
<th>Chunk size</th>
<th>512</th>
<th>256</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>12.930170s</td>
<td>11.254353s</td>
<td>37.003882s</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>15.433336s</td>
<td>14.657253s</td>
<td>14.518356s</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>14.002886s</td>
<td>12.283284s</td>
<td>14.018281s</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>16.649145s</td>
<td>18.778266s</td>
<td>18.013177s</td>
<td></td>
</tr>
</tbody>
</table>

**Table:** Performance of seidel with different OpenMP parameters
Conclusion

• Burden of manual annotation eliminated
• More SCoP coverage
• LLVM’s pre-optimization passes helps a lot
• Enough space for further improvement -> all are welcome to contribute
Future Work

• Support for memory access transformations in Polly (Planned for GSOC 2011)
  • Transformation on access function also
• Increasing coverage of Polly
  • Increasing SCoP coverage
  • Increasing the system coverage
• Integrating profile guided optimization into Polly

Not a valid SCoP

```c
scanf("%d", &b);
for (i = 0; i < N; i += b) {
    body;
}
```
Future Work

- Support for memory access transformations in Polly (Planned for GSOC 2011)
  - Transformation on access function also
- Increasing coverage of Polly
  - Increasing SCoP coverage
  - Increasing the system coverage
- Integrating profile guided optimization into Polly

Not a valid SCoP

```c
scanf("%d", &b);
for (i = 0; i < N; i += b) {
    body;
}
```

With profiling observed $b = 2$ most of the times

```c
scanf("%d", &b);
if (b == 2) {
    for (i = 0; i < N; i += 2) {
        body;
    }
} else {
    for (i = 0; i < N; i += b) {
        body;
    }
}
Future Work

- Support for memory access transformations in Polly *(Planned for GSOC 2011)*
  - Transformation on access function also
- Increasing coverage of Polly
  - Increasing SCoP coverage
  - Increasing the system coverage
- Integrating profile guided optimization into Polly

If N is small no need to detect this as a SCoP

```cpp
for (i = 0; i < N; i++) {
    body;
}
```
Tobias Grosser, Hongbin Zheng, **Raghes Aloor**, Andreas Simbürger, Armin Größlinger and Louis-Noël Pouchet Polly - Polyhedral optimization in LLVM *IMPACT 2011 (First International workshop on PolyhedrAl Compilation Techniques as part of CGO 2011)*, Chamonix, France.
What Did I Achieve?

- An interesting area to work
- A platform to strengthen my knowledge
- An opportunity to improve my mathematical skills
- Skill to work in a collaborative environment, especially in free software way
Questions????