A Framework for Automatic OpenMP Code Generation

Raghesh A (CS09M032)

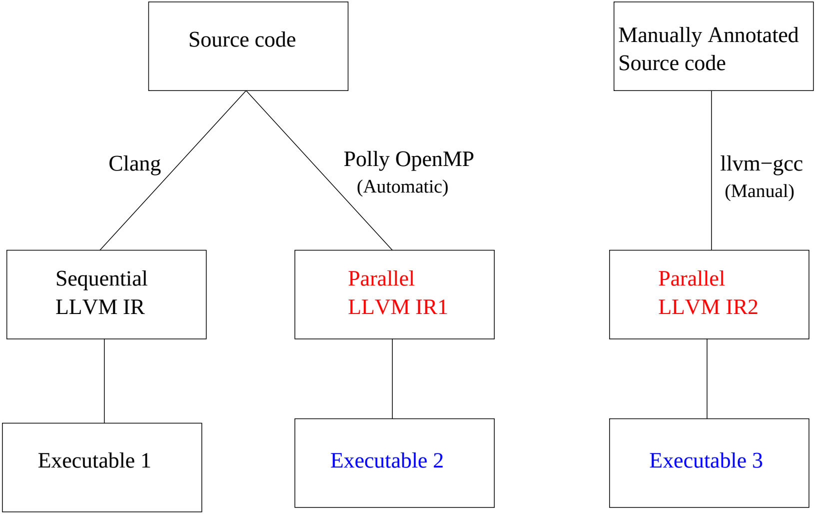
Guide: **Dr. Shankar Balachandran**

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



An Example

Source code

```
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
```

LLVM-IR Sequential

for.inc:

```
  %tmp3 = load i32* %i, align 4
  %inc = add nsw i32 %tmp3, 1
  store i32 %inc, i32* %i, align 4
  br label %for.cond
```

for.end:

```
  %0 = load i32* %retval
  ret i32 %0
}
```

An Example

Source code with OpenMP pragmas

```
float A[1024];

int main()
{
   int i;
  #pragma omp parallel for \
   schedule(runtime)
   for (i = 0; i < 1024; i++)
        A[i] += 10;
}
```

An Example

LLVM-IR Manual

```
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
return:
  %retval1 = load i32* %retval
  ret i32 %retval1
}
```

LLVM-IR Manual

```
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>
}
```

An Example

LLVM-IR Automatic

```
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


}
```

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;
```

- 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

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

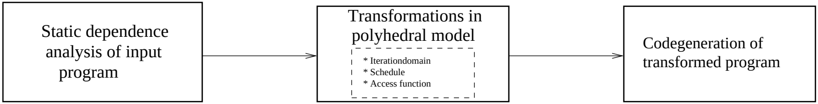
- Why not AST?
    - Dynamic instances of statements not captured
    - Rigid data structure
    - Less expressive than polyhedral model

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



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
$D_{S1} = \{(i,j) \epsilon Z^2 \mid 2 \le i \le N \wedge 2 \le j \le N\}$

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

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
$D_{S1} = \{(i,j) \in Z^2 \mid 2 \leq i \leq N \wedge 2 \leq j \leq N\}$

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

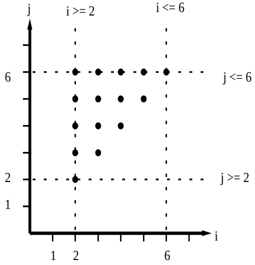


Figure: Graphical representation of iteration domain(S2)

Schedule

- Scattering function

```
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

```
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:

$$\phi_{S3}(i,j) = (i,j) \quad \phi'_{S3}(i,j) = (j,i)$$

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

Schedule

- Scattering function

```
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:

$$\phi_{S3}(i,j) = (i,j) \quad \phi'_{S3}(i,j) = (j,i)$$

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

Code generated for ϕ'_{S3}

```
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

```
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:

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

Code generated for ϕ'_{S3} o ϕ''_{S3}

```
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

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 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~~ arrays + pointers
- Independent of programming language

Polly

- 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

Polly

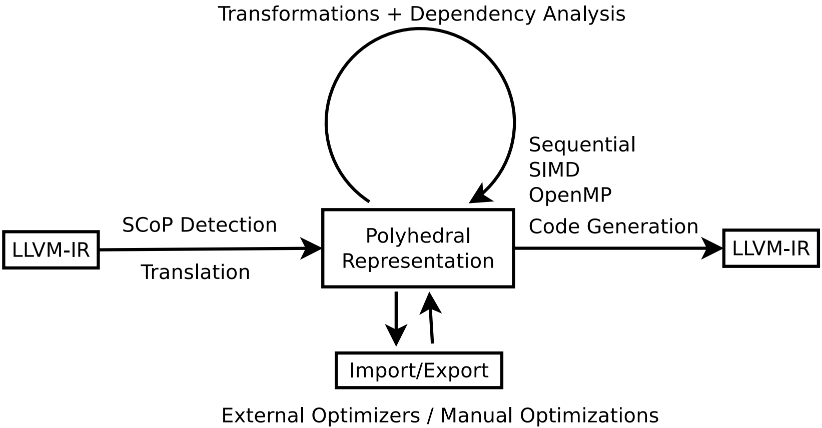


Figure: Architecture of Polly

OpenMP Code Generation in Polly

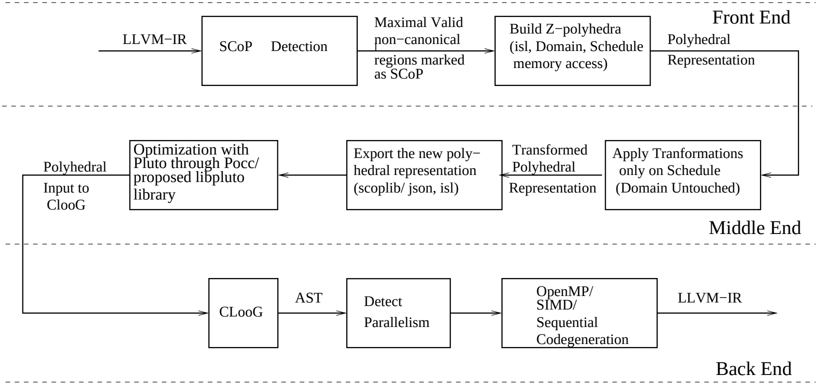


Figure: Detailed control flow in Polly

OpenMP Code Generation in Polly

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

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

OpenMP Code Generation in Polly

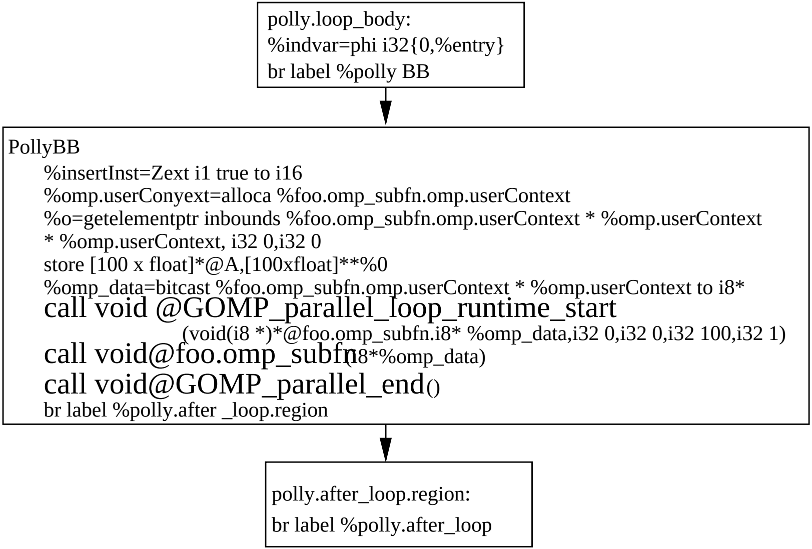


Figure: CFG showing sequence of OpenMP library calls

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 infrastructure

Testing

- GCC Compile farm

Testing

- GCC Compile farm

A simple test case

```
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

```
float A[1024];

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

	Serial Execution	Automatic Parallelization(Polly)	Manual Parallelization(GCC)
Intel Core 2 Duo(32 Bit OS)	9.509s	4.852s	4.835s
Intel Core 2 Duo(64 Bit OS)	6.40s	3.32s	3.50s
Intel Core i5(64 Bit OS)	6.96s	3.78s	3.75s
AMD Engineering Sample(24 Core)(64 Bit OS)	17.039s	0.757s	0.796s

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

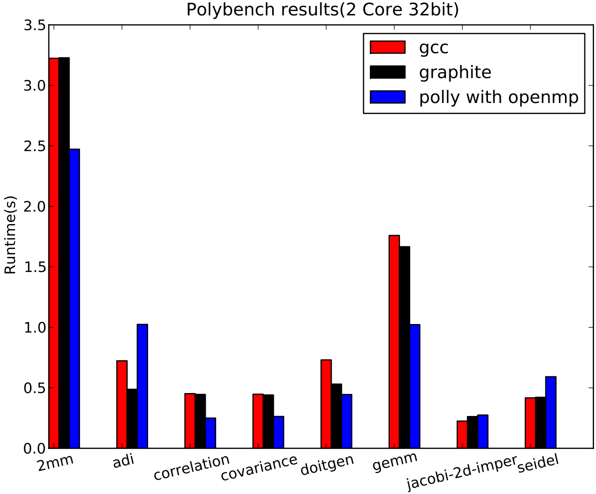


Figure: Performance comparison(2 core 32 bit)

Experimental Results

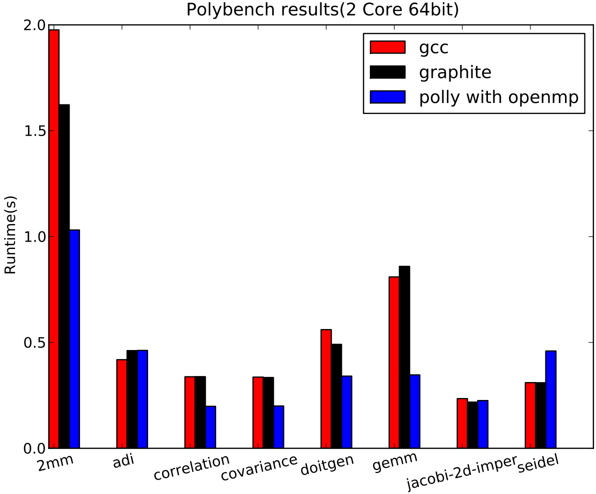


Figure: Performance comparison(2 core 64bit)

Experimental Results

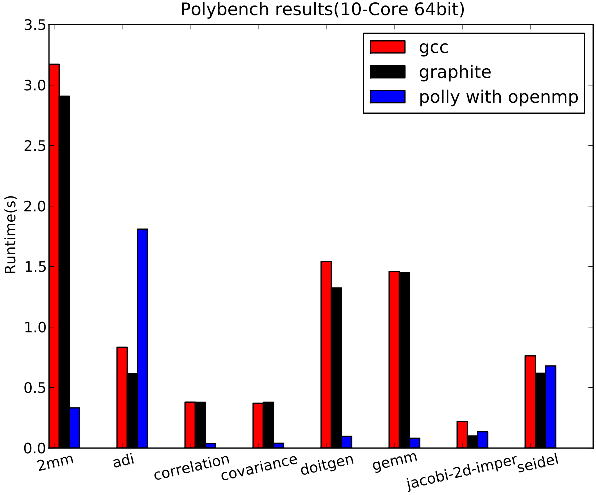


Figure: Performance comparison(10-core 64 bit)

Improving seidel's performance

- **Polly + Pluto + OpenMP + setting OMP_SCHEDULE=guided**

	Serial Execution	Polly + OpenMP	Polly + PLUTO + OpenMP
2 Core 32 Bit	0.417174s	0.591673s	0.348909s
2 Core 64 Bit	0.310160s	0.459641s	0.254605s

Table: Performance improvement of seidel

No of threads \ Chunk size	**512**	**256**	**128**
default	12.930170s	11.254353s	37.003882s
10	15.433336s	14.657253s	14.518356s
5	14.002886s	12.283284s	14.018281s
2	16.649145s	18.778266s	18.013177s

Table: Performance of seidel with different OpenMP parameters

- Conclusion
  - Burden of manual annotation eliminated
  - More SCoP coverage
  - LLVM's pre-optimzation 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

```
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

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

With profiling observed b = 2 most of the times

```
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

```
for(i = 0; i < N; i++) {
  body;
}
```

Publication

1. Tobias Grosser, Hongbin Zheng, **Raghesh 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????

THANK YOU