Advanced code optimization by LLVM Polly

Dominik Adamski

4Developers Łódź 2018

23.10.2018

Scope of the presentation

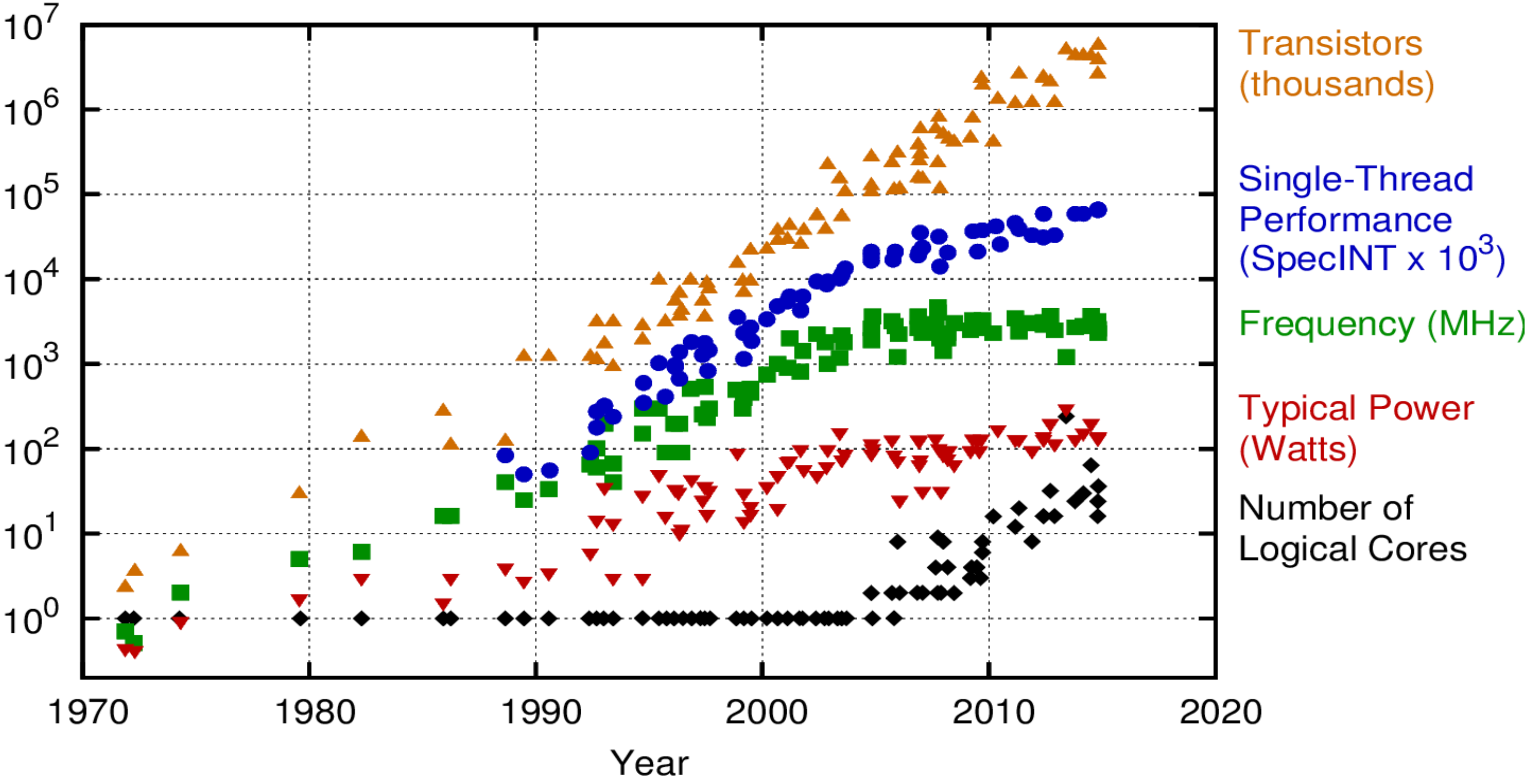
Topics included:

- Loop optimization
- Automatic parallelizable code detection and optimization
- Cost of optimization
- Scope of application
- Hints for developers

Topics excluded:

- Source code level transformations
- Techniques for improvement of data locality
- Target specific optimizations

Motivation



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

clang/LLVM

- Main open source competitor for gcc
- BSD-like licensed
- Support for C,C++, Objective-C, Objective-C++, OpenMP, OpenCL and CUDA
- Library based design:

  clang – compiler front end

  LLVM – compiler back end
- Entirely written in C++

Standard optimization options 1/2

```
clang++ -Olevel test.cpp
```

- `-O0` no optimization.
- `-O1` optimize for code size and speed without time consuming optimizations
- `-O2` optimize for code size and speed on intermediate level
- `-O3` optimize mainly for code speed, use the most resource consuming techniques

Standard optimization options 2/2

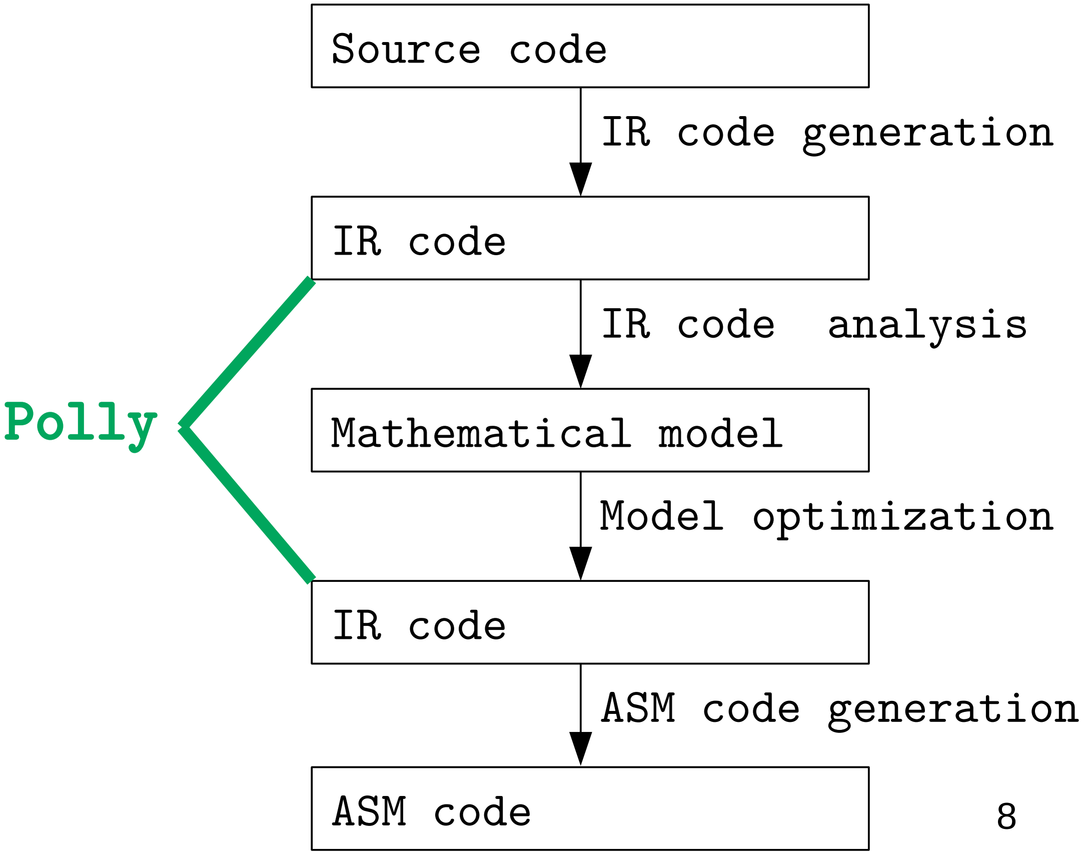
- `-Os` optimize for code size. Turn on all `-O2` optimizations that do not increase code size. Add more optimizations for code size reduction
- `-Oz` advanced optimization for code size
- `-Og` similar to `-O1`. In the future optimize code for good debugging experience
- `-Ofast` optimize for code speed. May break standard compliance

Difficulties with code parallelization

- Data dependencies
- Data races
- Need for code refactoring
- Additional tests

LLVM Polly

- High level loop and data locality optimizer
- Applied mathematical model for code analysis and optimization
- Based on LLVM infrastructure
- Integrated with clang command line arguments



Polly in details 1/2

- Independent from input source code language
- Detects loops where order of execution does not matter – Static Control Parts (SCoPs)
- It optimizes loops when there are no side effects
- It can be used as code analyzer

```
for (i = m; i < 4000; i++)
    x1[i] += A[3*i+5];
```

Polly in details 2/2

- It operates only on well defined loops:
  - loop borders need to be known at compilation time
  - loop borders can be parametric
  - indexes need to be described by linear function

```
for (i = m; i < 4000; i++)
      x1[i] += A[3*i+5];
```

How to launch Polly?

- Polly optimizer runs only if `-O3` optimization level is set
- Enable Polly optimizer without code parallelization:

```
clang -O3 -mllvm -polly test.c

clang++ -O3 -mllvm -polly test.cpp
```

- Enable Polly optimizer with code parallelization:

```
clang -O3 -mllvm -polly -mllvm -polly-parallel test.c -lgomp

clang++ -O3 -mllvm -polly -mllvm -polly-parallel test.cpp -lgomp
```

- Polly use internally OpenMP function calls for code parallelization

When standard optimization is not enough 1/2

```
for (i = 0; i < 4000; i++)
  for (j = 0; j < 4000; j++)
    x1[i] = x1[i] + A[i][j] * y_1[j];
for (i = 0; i < 4000; i++)
  for (j = 0; j < 4000; j++)
    x2[i] = x2[i] + A[j][i] * y_2[j];
```

Test specs:

- Code from Polybench 4.2.1
- Ubuntu 18.04
- AMD Ryzen 5 1600
- DDR4, 16GB, 2400MHz, CL17
- x1, A,x2,y_1,y_2 continuous arrays

When standard optimization is not enough 2/2

```
for (i = 0; i < 4000; i++)
   for (j = 0; j < 4000; j++)
      x1[i] = x1[i] + A[i][j] * y_1[j];
for (i = 0; i < 4000; i++)
   for (j = 0; j < 4000; j++)
      x2[i] = x2[i] + A[j][i] * y_2[j];
```

Time of execution:

- `clang-6.0 -O3`

  0.110448s
- `gcc-7.30 -O3`

  0.103513s
- `clang -6.0 -O3 -mllvm -polly`

  0.038668s **+185%** > `clang -O3`
- `clang-6.0 -O3 -mllvm -polly`

  `-mllvm -polly-parallel`

  `-lgomp`

  0.022625s **+388%** > `clang -O3`

What makes the difference?

- Improved data locality
  - loop tiling
  - loop fusion
  - loop splitting
  - loop interchange
- Code parallelization
- Optional code vectorization

Cost of optimization

```
for (i = 0; i < 4000; i++)
  for (j = 0; j < 4000; j++)
    x1[i] = x1[i] + A[i][j] * y_1[j];
for (i = 0; i < 4000; i++)
  for (j = 0; j < 4000; j++)
    x2[i] = x2[i] + A[j][i] * y_2[j];
```

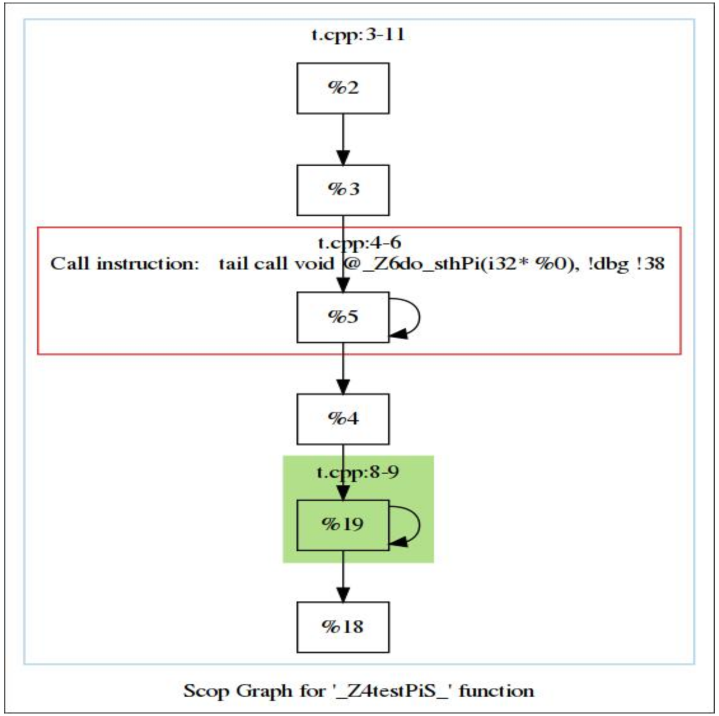
Time of compilation:

- Documentation: GEMM benchmark **+1.6%** overhead to `clang -O3`
- Measured:
  - `clang-6.0 -O3`
    0.184s
  - `gcc-7.30 -O3`
    0.163s
  - `clang -6.0 -O3 -mllvm -polly`
    0.277s – **+50%** to `clang -O3`
  - `clang-6.0 -O3 -mllvm -polly -mllvm -polly-parallel -lgomp`
    0.242s – **+31%** to `clang -O3`

Polly as code analyzer

```
clang++ t.cpp -O3 -mllvm -polly-show-only \
-mllvm -polly-process-unprofitable  -g -c
```

```
1 void do_sth(int *A);
2
3 void test(int *A, int *B) {
4   for (int i = 1; i < 100; ++i) {
5     B[i] = B[i - 1] + A[i * i];
6     do_sth(A);
7   }
8   for (int i = 0; i < 100; i++) {
9     A[i] += B[i];
10   }
11 }
```

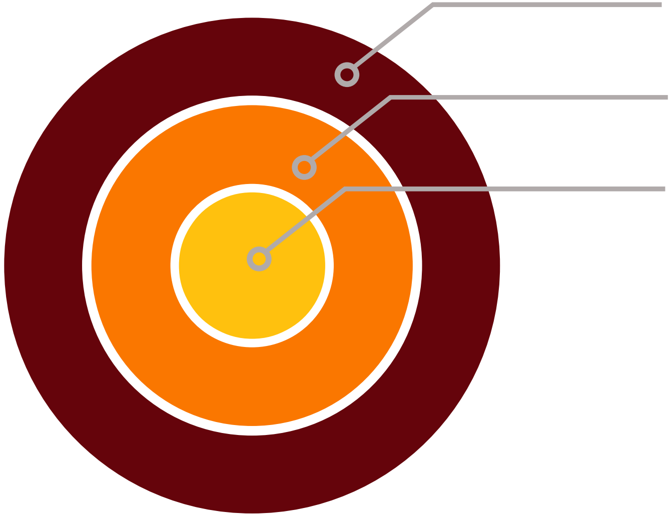


Call instruction: tail call void @_Z6do_sthPi(i32* %0), !dbg !38

Scop Graph for '_Z4testPiS_' function

Polly domains

- Data mining
- Scientific applications
- Artificial intelligence
- High performance computing



Polly optimization – threats

- Misleading heuristic
- Variety of hardware architecture
- Not optimal loop schedule
- Not optimal tile size
- Too long threads initialization and synchronization in comparison to operation time

How to use Polly efficiently

- Profile your application
- Determine functions which can be optimized
- Try to optimize them with Polly
  - use flag `-polly-only-func=<regex_string>`
- Check performance
- Optionally you can adjust Polly by setting flags:
  - adjusting Polly tile size: `-polly-tile-sizes=<number>`
  - Enable 2nd level tiling: `-polly-2nd-level-tiling`

Further reading

- LLVM project: http://llvm.org/
- Polly project: https://polly.llvm.org/
- Polly user’s manual: https://polly.llvm.org/docs/
- Detailed information about Polly’s internal structure: https://polly.llvm.org/publications/grosser-diploma-thesis.pdf
- Benchmark for Polly evaluation: http://web.cse.ohio-state.edu/~pouchet.2/software/polybench/
- Scott Meyers’s presentation about data locality: https://www.youtube.com/watch?v=WDIkqP4JbkE

Thank you!

Any questions?