Lab 3
Experimenting with the polyhedral model

We have already studied about dependence analysis and loop transformations in class. In this assignment, we will look at the potential of loop optimization through the polyhedral model. Compilation using polyhedral model involves representation of programs (especially those involving nested loops and arrays) to parametric polyhedra and exploiting combinatorial and geometrical optimizations on these objects to analyze and optimize the programs. The interest of using polyhedral representations is that they can be manipulated or optimized with algorithms whose complexity depends on their structure and not on the number of elements they represent. Furthermore, generic and compact solutions can be designed that depend on program parameters (e.g., loop bounds, tile sizes, array bounds).

Tool

We will use ***Polly***, a high-level loop and data-locality optimizer for LLVM IR for performing experiments in this assignment. There exists other tools also like Graphite (for GCC), PLUTO which support polyhedral compilation.

Benchmark

General matrix-matrix multiplication (GEMM) from polybench test suite. You can download the benchmark and study it in detail using below links:

1) https://sourceforge.net/projects/polybench/
2) www.cs.colostate.edu/AlphaZsvn/.../trunk/...polybench/polybench.../polybench.pdf

Use the following flags while compilation of gemm in polybench

- -DPOLYBENCH_TIME
- -DEXTRALARGE_DATASET
- -DDATA_TYPE_IS_INT

Part(a)

Study the "Polly" optimization framework and how gemm is optimized using it.

Note that, polly is not by default enabled in **Clang** to automatically optimize C/C++ code during compilation. It needs to be enabled by passing explicit flags. Please refer the below mentioned documents for further details.

Reference documents:

1) https://media.readthedocs.org/pdf/polly-/latest/polly-.pdf
2) http://www.grosser.es/publications/grosser-2012-Polly-Performing-polyhedral-optimizations-on-a-low-level-intermediate-representation.pdf
3) https://polly.llvm.org/publications/grosser-diploma-thesis.pdf
4) https://polly.llvm.org/

Report the run times for gemm.c, when compiled with

- GCC (-O3)
- ICC (-O3)
- Clang (-O3) without polly extension
- Clang (-O3) with polly extension

Part(b)

Analyze the x86 assembly code generated in the above four cases.
Write (if possible) a faster x86 implementation using intrinsics or inline assembly than the above code generated by clang for the gemm program. You can use any of the supported instruction set extensions for your manual optimization (SSE, FMA, AVX).

Report the assembly generated in all four cases, your analysis for these assembly codes, your manually optimized source code and x86 assembly. Report the run times for your implementation.

1) https://software.intel.com/sites/landingpage/IntrinsicsGuide/
2) https://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html

Part(c)

What are the key learnings about the strengths of the polly framework.

Part(d)

What are the limitations of the current polly framework implementation. You can use your own micro programs to highlight the limitations.