Clang -mllvm xxx：将选项传递给llvm。

clang -S -emit-llvm xxx:生成llvm IR表示。

clang -c -emit-llvm xxx：生成bytecode文件。

lli hello.ll/hello.bc：执行LLVM IR文件/bytecode文件。

llc hello.ll/hello.bc：生成汇编文件。

opt -O3 hello.bc/hello.ll -o xx.bc：优化代码。

opt -load ./mycustom\_pass.so -test\_llvm < hello.bc > /dev/null：执行指定优化遍。

as hello.s -o hello.o：生成目标文件。

ld -dynamic-linker /lib64/ld-linux-x86-64.so.2 /usr/lib/x86\_64-linux-gnu/crt1.o /usr/lib/x86\_64-linux-gnu/crti.o /usr/lib/x86\_64-linux-gnu/crtn.o -lc t2.o -o hello：链接生成可执行文件。

Breakpoint:

1. gdb opt
2. B llvm::legacy::PassManager::run;
3. Run --load ./mycustom\_pass.so -test\_llvm < hello.bc > /dev/null

llvm::PassManager::run.

llvm::legacy::PassManager::run

Llvm选项：

-print-before-all:对每个优化遍，在执行前打印出IR。

-print-after-all：对每个优化遍，在执行后打印出IR。

-debug-pass=Details：打印出遍在执行过程中的详细信息。

LLVM IR说明：

@xxx：全局变量

%xxx：局部变量

;xxxx：注释

LLVM IR Analysis

1. Print alue in IR:

------------> P val->dump()

几乎所有的类型，如module, function等都支持dump()函数，来打印出相应信息。

IR中相关类型的定义几乎都在llvm-3.9.1.src/include/llvm/IR目录下。

-------------> p llvm::Function .getName(): print function name.

1. 优化遍：

每个优化遍都包含了runonfunction函数，和runonmodule函数。

Dump pass info and IR：

-debug-pass=Details，Executions,Structure，Arguments等可以打印出执行的遍信息。

-print-after-all，-print-before-all：打印IR到屏幕，在每个遍的执行后/前。

Clang -mllvm xxx: 将xxx选项传递给 llvm。

Clang -c -emit-llvm:生成bc文件.bc。

Clang -S -emit-llvm：生成llvm IR文件.ll。

Pass manager：成员函数：

dumpPassStructure（0）:打印出该pass管理器中所有的遍。

getPassName()

Pass遍：

成员：

PassID：打印出遍的名称。

操作：

getPassName()

LoopVectorizePass过程分析：

#pragma clang loop vectorize(enable)

-Rpass=loop-vectorize：打印出进行了向量化的循环。

-Rpass-missed=loop-vectorize：打印出没有进行向量化的循环。

-Rpass-analysis=loop-vectorize：分析没有进行向量化的循环问题出在哪条语句。

#pragma unroll/nounroll：控制循环是否展开。

processLoop函数：

1. 断言仅仅处理最内层循环。
2. 提取出pragma信息到LoopVectorizeHints对象中。
3. 根据hints、循环次数是否过小等判断是否进行向量化。包括判断向量化是否合法。
4. 使用代价模型来计算出向量宽度，interleave count（根据结果看上去好像是循环展开次数）
5. 生成InnerLoopVectorizer类的对象，进行向量化。先是生成一个新的空白循环（包含循环index变量，cmp判断语句），然后对于旧循环中的每个语句，进行加宽形成向量。