How to run program

1. In Makefile change LLVM_CONFIG to your llvm-config's path
2. In directory type “make hw1.so” -> get hw1.so
3. If want test case 1 & 2's IR, type “make test1” or “make test2”
4. If want result, change Makefile's run's depend object to test case's IR

The cases can handle

1. Only one dimension loop (like test_case)
2. Induction variable increase by one (like test_case)
3. Can handle more than one array access in RHS (ex: A[i] = B[i] + C[i])
4. Opcode only can handle +, -, * (like test_case)
5. Test1, Test2 required

Experiment report

1. In the “entry” block get the induction variable's name & lower_bound.
2. In the “for.cond” (name may different) block get the induction variable's upper_bound.
3. In the “for.body” (name may different) block get each array's access pattern, put info in SmallVector data structure.
4. Use Brute force method to find data dependence rather than method in class, traverse every iteration to find all possible data dependence.

Bonus

Mixin pattern

1. While the llvm pass usually writes in the anonymous namespace, the common function(to get info) has to be implemented in each pass’s class, which is a tedious task.
   Mixin pattern helps us to let new llvm pass inherit base pass’s class, it will provide informational APIs needed for pass, function will follow template’s typename to return corresponding result.

Utilize ADT

1. I chose SmallVector to store array access info.
   In llvm’s document, it said that it is much better to use SmallVector than vector, because it avoids relatively expensive malloc/free calls, when SmallVector is smaller than N(programmer wants to allocate), no malloc is performed.
2. Another benefit is that when SmallVector is small, it is most useful when on the stack. SmallVector will allocate on the stack rather than heap, making it avoid expensive malloc/free calls.