Assignment 2: Dependency Analysis

15-745: Optimizing Compilers

Due: 10:30am, Thursday, February 14

In this assignment you will implement a memory dependency analysis for memory references inside simple, perfectly nested loops. You will determine the direction and, if possible, the distance of every memory dependence inside of the loop and create the direction matrix for the loop.

1 Code

We provide a file `LoopDependenceAnalysis.cpp` that identifies perfectly nested loops where the inner loop is a single basic block that only references memory through load and store instructions. You should add this file to your 15745 llvm directory and modify your makefile so that LIBRARYNAME is 15745. Your task is to implement the `computeDependencies` function which determines the type of dependency between every pair of memory operations where at least one member of the pair is a store (since there are no real dependencies between load instructions). If the pointers being referenced by memory operations do not alias, then they are independent. If the pointers do alias and the same pointer or array is being indexed by an affine function of loop induction variables and integer constants, you should determine the direction and distance vectors. If the pointers alias, but are not analyzable, then all possible direction vectors should be conservatively reported. In addition to the individual direction vectors, you should compute the overall direction matrix of the loop.

When analyzing affine references you need only analyze the fully separable SIV (single induction variable) and ZIV (zero induction variable) cases. You do not need to handle symbolic constants. You do not need to implement any analyses that require knowledge about the loop bounds.

Using the Statistics package, you should keep track of four different counts:

- **NumIndependentMemOps** The number of memory operation pairs that are provably independent.
- **NumDirectionalDependentMemOps** The number of memory operation pairs that are dependent and for which you can determine a precise direction vector for.
- **NumAmbiguousDependentMemOps** The number of memory operation pairs that are not provably independent and for which you cannot determine a precise direction vector.
- **NumInterestingLoops** The number of loops whose direction matrix is neither empty (no loop carried dependencies) nor full (all loop carried dependencies).

For each pair of memory operations you should print out a line to the `DEPINFO` file using the format shown in Figure 1. For both memory operations you should specify whether the memory operation is a load (L) or store (S), provide the internal unique pointer value of the memory operation, provide the name of the base pointer of the memory operation, and then print out the direction vector. Note that independent

```
#define N 512
int A[N][N][N];
int global;
void test0(int *ptr)
{
  unsigned i,j,k;
  for(i = 0; i < N; i++)
    for(j = 0; j < N; j++)
      for(k = 0; k < N; k++)
      {
        global += i*j*k;
        A[i+1][j+1][k] = A[i][j][k] + A[i][j+1][k+3];
      }
}

Loop Nest 1
L0xd039a0(global)        S0xd02560(global)        [***]
S0xd02560(global)        L0xd039a0(global)        [***]
L0xd028f0(A)     S0xd041e0(A)     [>(-1)>(-1)=]
L0xd02be0(A)     S0xd041e0(A)     [>(-1)=<(3)]
S0xd041e0(A)     L0xd028f0(A)     [<(1)<(1)=]
S0xd041e0(A)     L0xd02be0(A)     [<(1)=>(-3)]
Matrix 1
[***]
```

Figure 1: An example loop and the information output by the dependence analysis.

memory operations are not printed. If a precise direction vector exists, then both the direction and distance information should printed as shown in Figure 1. If it is not possible to determine a precise direction vector, all vectors, using the wildcard notation, should be conservatively reported.

In addition to reporting the individual direction vectors, you should compute the direction matrix for the loop nest. This is the union of all the legal direction vectors within the loop. A direction vector is legal if its leftmost non-'=' direction is a '<'. If the loop nest contains dependent memory operations that do not have precise direction vectors, then wildcard notation should be used when printing out the matrix, as shown in Figure 1.

2 Testing

We provide a tarball, `bench2.tgz`, of prebuilt llvm binaries that you can unpack in your versabench directory. We also provide a script, `optbench`, that, after you update it to point to your llvm working directory, will run your pass on these prebuilt binaries. We also provide `loops.c` which contains several simple test cases and their solutions. Note that two benchmarks, `101.tomcatv` and `vpenta` are compiled from a slightly different version of source than previously provided. This source code is provided in a `nsrc` directory.

3 Writeup and Handin

You should determine the total count across all the provided benchmarks for the four statistics described in Section 1 and report this in your writeup.

Our analysis is fairly simple and misses out on several opportunities. In fact, most benchmarks do not have nontrivial direction matrixes. Identify one loop nest in the provided benchmarks where we are unnecessarily conservative in our analysis. Describe how you could extend your code to determine more precise dependence information for this loop. You must include the source code of the loop, your current analysis output for the loop, a description of how you would improve the analysis (you do not have to provide source code) and the improved analysis output you would generate if you implemented the described improvement. Some possible ideas for improvement are:

- coupled groups
- handle control flow
- multiple induction variables
- more sophisticated tests in general
- bounds checking.

For **extra credit** implement and submit your improvement.

You should handin your writeup (in pdf form) and your version of `LoopDependenceAnalysis.cpp` into your course handin directory.[1]

[1] If you have trouble, just email your submission to `dkoes`. If you're on andrew and getting permissions problems, try running `aklog` *`userid`*`@cs.cmu.edu`