5130 Project 3: Spatial Reuse Detection

Due: Monday, Nov. 7 @ 5:00 PM

We will continue basing our project on the Low Level Virtual Machine (LLVM) compiler infrastructure. In this project you will increment the dependence analysis pass to detect self-spatial reuses. A memory reference in a loop nest has self-spatial reuse if the distance (stride) between two consecutive accesses by this reference in the innermost loop is smaller than the cache line size.

1 Project Summary

LLVM has implemented a loop dependence analysis pass
(see `$llvm/lib/Analysis/DependenceAnalysis.cpp`) which is based on Goff et al. PLDI'91. Our lecture notes are also based on this paper. Your work would be an extension to this pass. You will find most information you need to detect spatial reuse is also used by dependence testing.

2 Implementation

You only need to consider perfectly nested loops which only contains one basic block in the loop body (all test cases are attached). You can assume all loop bounds are known constants and loop steps are also constant. You only consider array references whose subscripts are affine expressions. You can assume a default cache line size of 64 bytes or introduce a compiler option to specify the cache line size.

2.1 Output

Direct your output to stderr. For each loop nest, print "Loop Nest" followed by a loop nest number that is based on your analysis processing order. Then in this loop nest, for each array reference for which you detect self-spatial reuse, print the reference, the loop level where the spatial reuses is carried, and the stride of the reuse. A stride is defined as the address distance between two accesses by the reference in two consecutive innermost loop iterations. The stride is measured in terms of 4-byte words.

```
for (i = 1;  i < N; i++)
  for (j = 1; j < N; j++)
  {
     a[i][j] = a[i][2*j] + 10;
  }
```

For example, the style of the expected output for the above loop nest (see test.c and test.ll) is as follows.

```
Loop Nest 1
  %1 = load i32, i32* %arrayidx5, align 8, !tbaa !3
level: 2
stride: 2
  store i32 %add, i32* %arrayidx9, align 4, !tbaa !3
level: 2
stride: 1
```

The command to genetate this output can be as follows.

```
opt -enable-new-pm=0 -da test.ll -o test.bc
```

2.2 Testing

You can use the attached loops.c and loops.ll as your testing input. Both loops.ll and test.ll are generated using “clang -O1”.

3 Submission

Submit your source code for this new analysis pass and other source code if you make changes. Write a short description of your implementation and attach the output for loops.c.