Michgan Tech

Spatial Reuse Dectection

Project III

Writen by: liang YAN

Computer Science

Last modified: April 2015

0.1 Introduction

1. compile the source file

clang -O1 -emit-llvm -c loops.c

2. run tests

opt -analyze -ds –ssr=true –cls=64 loops.bc

0.2 Implementation

0.2.1 Basic Algorithm

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.

I only think about the self spatial reuse at innermost level , for example I J K order, I only think about A[I][J][K], A[I][K][K], A[K][K][K]. If no K at all, I take it as a self-temp and pass over the dectect. For example,

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

the stride of a[i][j][k] is 1, the stride of a[i][k2][k3] is upperbound of L3 * stride of k2 + stride of k3, k2 means k in level 2 and k3 means k in level 3. Same to a[k1][j][k3]. But when meets a[i][j][j], I will take it as a self-temp.

At last we will make sure the stride count will be lower the cachelinecount, which means two consecutive value store in a same cache line.

0.2.2 Loop Nest Number

Find the outermost loop (no parent) of each loopnest, save it to a currentLoop, then , find another outermost, check if it is same as the currentLoop, if same, do nothing, not same, let loopNest++, and make currentLoop = current outermost loop.

0.2.3 Get the variable type

Need to know the value type of the array, because different type comes with different layout size. Int would be 4, and doulbe is 8, which means a cacheline with size 64 can hold 16 integers but 8 doulbe variable. This is one thing we need to notice, if we get a pointer value, we need to find the element it points to.

0.2.4 Get the subscripts

Maily using the Pair struct offered by this DA file. For example

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

we will get [i] of a[i][j] from Pair[1], and could get the i of a[i] from Loop 1, also we could get the coeffient 1, which we could use it as the stride Same idea with [j], we know its coeff is 1 and loop level is 2, we need its stride is lower or equal to cachelinecount

0.2.5 Get the upperbound

Upperbound is necessary for a[i][k][k] and a[k][j][k], this kind of situations. We also need to get all level upperbound in advance.

0.2.6 MIV

For array A[i][j+k][i], we need do a loop for all j and k in j+k, and check if it is same as k, if same, we need to use upperbound to caculate the stride count.

0.3 The difficulties

For A[i][j][j], there is a situation that, cacheline is very big, and upperbound of lever 2 and 3 is small, then although level 3 has a self-temp, level 2 still could have a self-reuse, which means a[i][j][j] and a[i][j+1][j+1] could be in a same cache line. Considering its big possibily accroding to dimmentions, I did not handle this case this time.

0.4 Results

```
test1

test2

Loop Nest  1
  %1 = load i32* %arrayidx, align 8, !tbaa !1
level : 1
stride : 2

  store i32 %1, i32* %arrayidx2, align 4, !tbaa !1
level : 1
stride : 1

test3

Loop Nest  1
  %1 = load i32* %arrayidx, align 8, !tbaa !1
level : 1
stride : 4

  store i32 %1, i32* %arrayidx2, align 4, !tbaa !1
level : 1
stride : 2

test4

Loop Nest  1
  %2 = load i32* %arrayidx, align 4, !tbaa !1
level : 1
stride : 8

  %4 = load i32* %arrayidx3, align 16, !tbaa !1
level : 1
stride : 12

test5

test6

Loop Nest  1
  %3 = load i32* %arrayidx5, align 4, !tbaa !1
level : 2
stride : 1

  store i32 %add10, i32* %arrayidx14, align 4, !tbaa !1
level : 2
stride : 1

test7

Loop Nest  1
  %4 = load i32* %arrayidx6, align 8, !tbaa !1
```

```
level : 2
stride : 2

  %6 = load i32* %arrayidx11, align 4, !tbaa !1
level : 2
stride : 3

  store i32 %add12, i32* %arrayidx17, align 4, !tbaa !1
level : 2
stride : 1

test8

Loop Nest  1
  store i32 %add, i32* %arrayidx13, align 4, !tbaa !1
level : 2
stride : 3

test9

Loop Nest  1
  %0 = load double* %arrayidx10, align 8, !tbaa !1
level : 3
stride : 1

  %3 = load double* %arrayidx17, align 8, !tbaa !1
level : 3
stride : 2

  store double %add18, double* %arrayidx25, align 8, !tbaa !1
level : 3
stride : 1

test10

Loop Nest  1
  %4 = load double* %arrayidx17, align 8, !tbaa !1
level : 3
stride : 4

  store double %add18, double* %arrayidx24, align 16, !tbaa !1
level : 3
stride : 2

test11

test12

Loop Nest  1
  %1 = load double* %arrayidx10, align 8, !tbaa !1
level : 3
stride : 5
```