CS314 Spring 2017

April 25

Due Monday, May 1, 11:59pm

submission: pdf file through sakai.rutgers.edu

Problem 1 – Dependence Analysis

List all all dependencies in the following loop nests. If a dependence exists, how far apart in term of iterations are the accesses to the same memory location? This is called the *distance* of a dependence. Also, state explicitly whether a dependence is a true, anti, or output dependence.

1.
```
do i = 3, 100
  a(i) = a (i-1) + a(i+1) + a(i-2)
enddo
```

2.
```
do i = 4, 5
  a(i) = a (i+2) + a(i-3)
enddo
```

3.
```
do i = 1, 100
  a(2*i) = a(2*i-1) + a(2*i+1)
enddo
```

4.
```
do i = 1, 10
  a(i) = a(5) + 1
enddo
```

5.
```
do i = 1, 9
  a(10-i) = a(i) + 5
enddo
```

Problem 2 – Vectorization

A statement-level dependence graph represents the dependences between statements in a loop nest. Nodes represent single statements, and edges dependences between statements.

An edge is generated by a pair of array references that have a dependence. Edges are directed from the source of the dependence to its sink. For example, for a true dependence, the source is a write reference, and the sink is a read reference. There may be multiple edges (i.e., dependences) between two nodes in the graph.

```
        for i = 2, 99
S1:        a(i) = b(i-1) + c(i+1);
S2:        b(i) = c(i) + 3;
S3:        c(i) = c(i-1) + a(i);
        endfor;
```

Here is a basic vectorization algorithm based on a statement-level dependence graph as discussed in class (lecture 26):

1. Construct statement-level dependence graph considering true, anti, and output dependences;
2. Detect strongly connected components (SCC) over the dependence graph (note: a single node may be an SCC by itself); represent SCC as summary nodes; walk resulting graph in topological order; For each visited node do
   (a) If SCC has more than one statement in it, distribute loop with statements of SCC as its body, and keep the code sequential.
   (b) If SCC is a single statement and has no loop-carried output or true dependencies, distribute loop around it and "collapse" loop into a vector instruction. For example, the loop

```
for i=1, 100
  a(i) = b(i) + 1;
endfor
```

   can be "collapsed" into a single vector instruction

```
a(1:100) = b(1:100) + 1;
```

   . If there are loop-carried true or output dependencies on the single statement, distribute the loop around the statement and keep loop sequential.

1. Show the statement-level dependence graph for the loop with its strongly connected components. Show every dependence by a pair of array references. Note: There may be multiple depencencies between two statements.
2. Show the generated code by the vectorization algorithm described above.