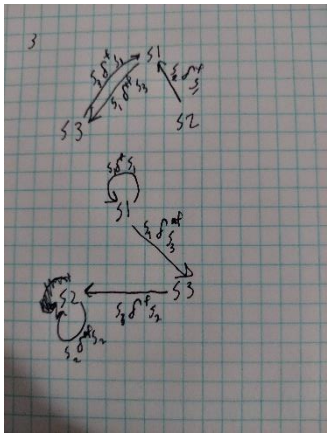
Jared Hogan HW 4

1. For the first code block there is no output dependency. There is are dependencies of an anti S3(I,j,k) => S1(i+1,j-1,k+1), anti S2(I,j,k) => S1(I,j,k+1) anti S3(I,j,k) => S2(I,j+1,k). Level 2 carries the dependency for the c array and levels 2 and 3 carry the dependency for the a and b array. Level 1 carries the dependency for the a array as such there are no parallelizable loops. Thus, loop permutation will not help increase the number of parallel loops.

   For the second code block there is no output dependency. There is a dependency of anti S1(I,j-1,k) => S2(I,j,k), anti S2(I,j,k)=>S3(I,j-1,k) and anti S3(I,j+1,k)=>S1(I,j,k). As such level 1 carries the dependencies for the a array, level 2 carries the dependency for the c array, and level three carries the dependency for the b array. As there a dependency carried by every level loop permutation can not help because on solution to one array will break another dependency.

2. The first code sample has a flow dependency with itself carried by the I loop. As such it cannot be vectorized as later values depend on previous values.
   The second code block has a benign flow dependence and as such is safe to vectorize into:
   a[1:255] = a[2:256]+b[0:254]
3. For the first code block the maximized vector code is
   for(int i=1;i<N-1;i++){a[i+1]=c[i-1]+1;c[i]=a[i]+1;}
   b[0:N-2]=a[2:N]+1;



   For the second code block it can't be vectorized because of many flow and anti dependencies. Thus the code provided in the best. for (i=2; i<N-2; i++){ a[i] = a[i+1]+c[i-1];b[i] = b[i-1]-1;c[i] = b[i-2]+1;}

4. The first code section has a flow dependence vector of (2,-1,-2). As such it is not vectorizable no matter the permutation because on value in the vector relies on a previous value to be computed first.
   The second code block has an anti-dependence vector of (0,0,1). As such it is vectorizable which results in: a[1:255][1:255][1:255] = a[1:255][1:255][0:244]