```
for (int i = 0; i<max; i++){
float tmpA = 0.0;
 for (int j = 0; j<max2; j++){
    tmpA += 2*B[j];
    ...
    }
 ...
}
```

Figure 1: A code snippet with tmpA initialized for every iteration.

```
float tmpA[max] = {0.0, ...};
for (int i = 0; i<max; i++){
 for (int j = 0; j<max2; j++){
    tmpA[i] += 2*B[j];
    ...
    }
 ...
}
```

Figure 2: The same code with tmpA hoisted.

0.1 CPU Parallelization Preparation

0.1.1 The Idea

Before even beginning to worry about `tridag`, we needed to prepare the CPU code. The first step is to conglomerate every secondary function, which allowed us to hoist the initialization of the globs (global variables) into glob arrays. That is, instead of initializing a temp variable or array for every iteration of a loop, we instead initialize an array one dimension larger, with size equal to the number of iterations of the loop, before the looping code. See figures 1 and 2 for an example.

The purpose of this is twofold. First, this allows us to serialize a computation which would otherwise be repeated by every thread. By computing it beforehand, the threads can be spared this extra work. Second, this allows us to easily parallelize the inner loops. Since each iteration of the inner `j` loop requires access to a single `tmpA` per iteration of the `i` loop, we would have to compute it, then pass it on as a variable to each of the threads. In the hoisted version, `tempA[]` is copied to device memory, so that the inner loop can just access the appropriate version without much trouble.

0.1.2 The Work

The first step for us was to begin preparing the CPU for kernel parallelization. Before distributing the outer loops in ProjCoreOrig.cpp, we began by moving all of the secondary functions (`void updateParams`, `void setPayoff`, `REAL value`, and `void rollback`) into `void run_OrigCPU`. This allowed us to easily see the globs and their relations to one another.

The work at this step is temporary. More time is spent on initializing arrays, memory usage is larger, and the code runs slower. This is to help us prepare for parallel distribution.

The `REAL strike` variable has been removed, being placed instead inside of one of the kernels. The variables which have been hoisted at this step are:

```
void  run_OrigCPU(...
{
    initGrid(s0, alpha, nu, t, numX, numY, numT, globs);
    initOperator(globs.myX,globs.myDxx);
    initOperator(globs.myY,globs.myDyy);
    vector<PrivGlobs> globArr (outer, globs);
    ...
```

0.2 Loop

0.2.1 notes while writing

mem consumptin bigger, startup time increase threads, lot more threads, more overhead. It is not meant to be fancy, just as precursor to parallelization.

0.3 Simple CUDA

0.4 TRIDAG