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.

```
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.

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, `tmpA[]` 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. The code is objectively de-optimized, since more time is spent on initializing arrays, memory usage is larger, and the code runs slower. The purpose of this is to prepare the code for optimal loop 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 listed in figure 3.

```
void  run_OrigCPU(...
{
   ...
   // Generate vector of globs. Initialize grid and operators onces
    // and make default element of vector
    // Hoisted from "value"
    PrivGlobs   globs(numX, numY, numT);
    initGrid(s0, alpha, nu, t, numX, numY, numT, globs);
    initOperator(globs.myX,globs.myDxx);
    initOperator(globs.myY,globs.myDyy);
    vector<PrivGlobs> globArr (outer, globs);
    ...
    //Rollback globs
    vector<vector<vector<REAL> > > u(outer,vector<vector<REAL> >(numY,
        vector<REAL>(numX))); // [outer][numY][numX]
    vector<vector<vector<REAL> > > v(outer,vector<vector<REAL> >(numX,
        vector<REAL>(numY))); // [outer][numX][numY]
    vector<vector<REAL> > a(outer,vector<REAL>(numZ)),
        b(outer,vector<REAL>(numZ)), c(outer,vector<REAL>(numZ)),
        y(outer,vector<REAL>(numZ)); // [outer][max(numX,numY)]
    vector<vector<REAL> > yy(outer,vector<REAL>(numZ)); // temporary
        used in tridag // [outer][max(numX,numY)]
    ...
```

Figure 3: Hoisted variables in ProjCoreOrig.cpp.

0.1.3 The result

After this work was done, ProjCoreOrig.cpp was is a state best for OpenMP parallelization, adding `#pragma omp parallel for` before each parallelizable

for loop and compiling with options `-Xcompiler` and `-fopenmp`. This is the version referred to in the results table, figure **??**.

0.2 Distributing Loops

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

0.3.1 writing notes

0.4 TRIDAG