DFT GPU operators for 5D interpolation

Kai Zhuang*, and Daniel Trad
kzhuang@ucalgary.ca

Abstract

- Standard 5D interpolation suffers from both long offset accuracy and speed problems.
- Using DFT for exact location instead of binning fixes issues at wide azmuth/long offsets.
- Use of GPU processing to reduce runtime of DFT algorithm for large datasets.

DFT Kernel for parallelization

Algorithm 2 gpu shared MVM pseudo code based on Nvidia guide (Cook, 2022)

```
__global__
function MVM
    blockRow = blockIdx.y;
    blockCol = blockIdx.x;
    Csub = GetSubMatrix(C, blockRow, blockCol);
    row = threadIdx.y;
    col = threadIdx.x;
    for m < (A.width/BLOCK_SIZE); do
        Asub = GetSubMatrix(A, blockRow, m);
        Bsub = GetSubMatrix(B, m, blockCol);
        __shared__As[BLOCK_SIZE][BLOCK_SIZE];
        __shared__Bs[BLOCK_SIZE][BLOCK_SIZE];
        As[row][col] = GetElement(Asub, row, col);
        Bs[row][col] = GetElement(Bsub, row, col);
        sync
        for e < BLOCK_SIZE do
            Cvalue+ = As[row][e] * Bs[e][col];
        end for
    end for
    SetElement(Csub, row, col, Cvalue);
end function
```

As we are applying a direct assessment of the Fourier transform, the main operation for the transform being applied is a simple matrix-vector multiplication (mvm). The matrix represents the mapping from the spatial plane to the wave-number domain and the vector represents each frequency slice of the data after FFT is applied in time to the dataset. For the multiplication algorithm, we compute the single precision matrix-vector multiplication. We make a comparison of general CPU and GPU approaches to mvm as well as standard library packages that are commonly used.

Runtime comparison

Something interesting to note is that at very small matrix sizes the CPU implementations lead the runtimes, while the GPU code is seen to be slower. This speed difference at small matrices can be attributed to the lack of a full population of threads on the GPU.

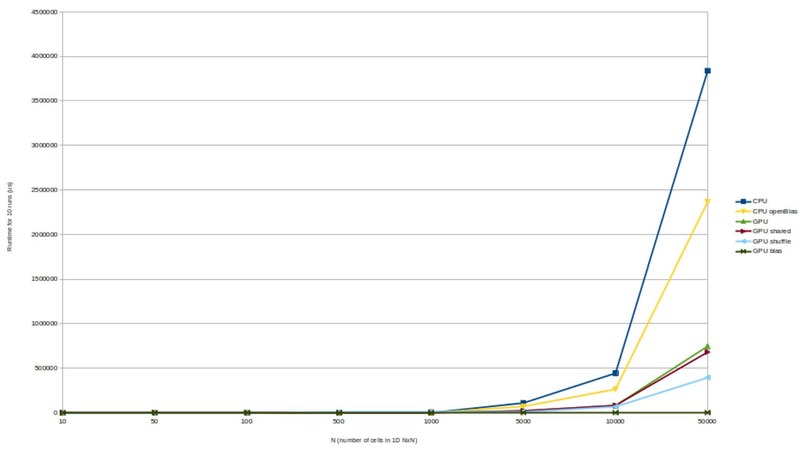

Figure 1: Compairson of GPU and CPU runtimes for matrix-vector multiplication, CPU is an intel 8 core 8 thread 9000 series CPU, GPU is an Nvidia Geforce RTX 3060.

In general, in these cases of small calculations, the sheer speed difference in processor threads pushes the advantage toward CPU threads. As the GPU threads populate with data, it can be seen that the GPU kernels start to take the lead while the CPU begins to lag significantly as the graph diverges.

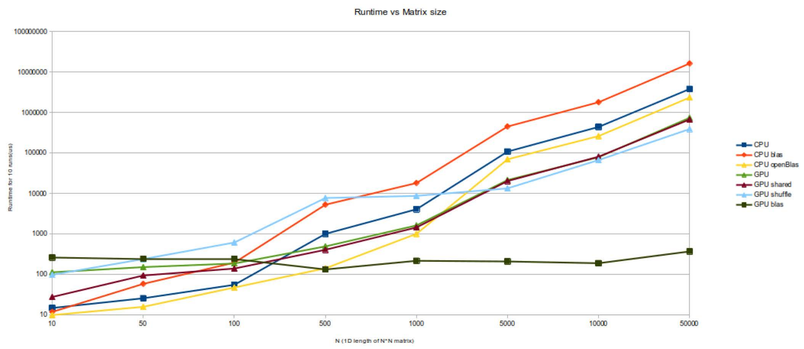

Figure 2: Compairson of GPU and CPU runtimes for matrix-vector multiplication, CPU is an intel 8 core 8 thread 9000 series CPU, GPU is an Nvidia Geforce RTX 3060.

Interpolation

he general 2D interpolation has been implemented, the reconstruction can be seen in figure 4 and the decimation in figure 3. The decimated shot gather has two traces between every live trace removed for the interpolation. We noticed when comparing CPU and GPU results is that although the results are the same there is are float-rounding difference between GPU and CPU results causing random noise.

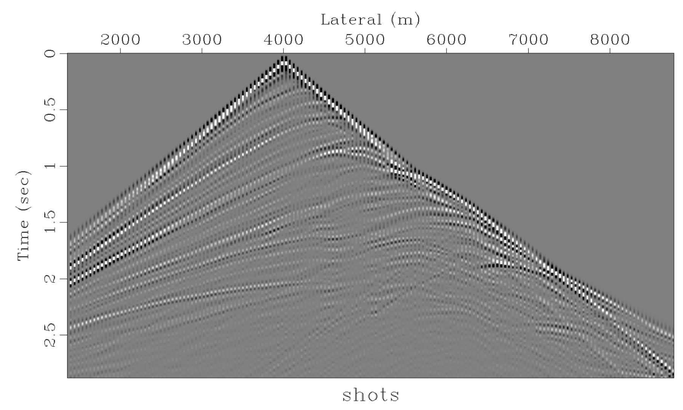

Figure 3: Decimated shot.

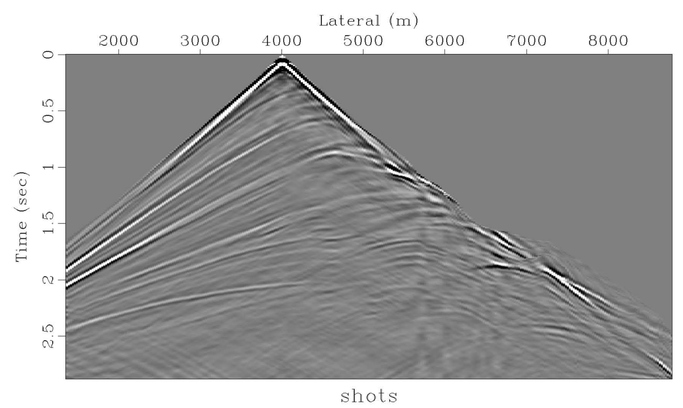

Figure 4: Shot reconstruction using DFT interpolation.

Conclusions

- GPU implementations are far faster than CPU implementation at large matrices for this use case.
- Results from GPU kernel and CPU kernel for Fourier interpolation are the same barring float rounding differences.

Acknowledgments

We thank the sponsors of CREWES for continued support. This work was funded by CREWES industrial sponsors, and NSERC (Natural Science and Engineering Research Council of Canada) through the grant CRDPJ 461179-13.