cusFFT: A High-Performance *Sparse* Fast Fourier Transform Algorithm on GPUs

Cheng Wang*, Sunita Chandrasekaran† and Barbara Chapman‡
*Department of Computer Science, University of Houston, Houston, TX, USA
†Department of Computer and Information Sciences, University of Delaware, Newark, DE, USA
‡Department of Applies Mathematics & Statistics and Computer Science, Stony Brook University, NY, USA

Introduction

Fast Fourier Transform (FFT) is a widely used numerical algorithm in scientific and engineering applications. When the input data is *sparse*, i.e., the *n*-points input data lead to only $k \ll n$ non-zero coefficients in the transformed domain, the FFT is clearly inefficient: the algorithm performs *O(nlogn)* operations on *n-points* data in order to calculate only a small number of *k* large coefficients, while the rest of $n - k$ numbers are zero or negligibly small. The *sparse* FFT (sFFT) algorithm provides a solution to this problem.

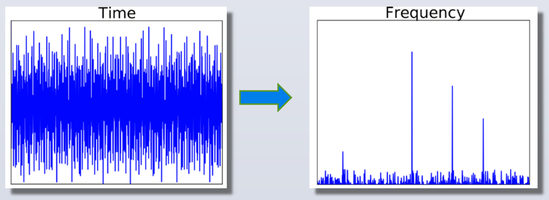
In this poster, we explore the challenges and propose effective solutions to efficiently port sFFT to massively parallel processors, such as GPUs, using CUDA. We present some of the optimization strategies such as index coalescing, loop splitting, asynchronous data layout transformation, linear time selection algorithm that are required to compute sFFT on such massively parallel architectures.

Our CUDA-based sFFT, cusfft, performs over 10x faster than the state-of-the-art cuFFT library on GPUs and over 28x faster than the parallel FFTW on multicore CPUs.

Motivation

Fourier Transform is one of the most fundamental tools widely used in :

- signal processing
- compression (audio, image, video)
- data analysis
- FFT: *O(nlogn)* time

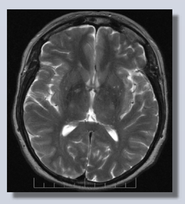


Often the Fourier transform is dominated by a small number of "peaks"

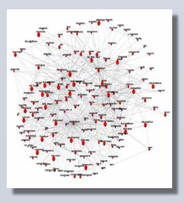
- Precisely the reason to use for data/image/video compression.
- If most of mass in *k* locations, can we compute FFT faster?

***Sparsity* is common:**

(a) Seismic data



(b) Biomedical images

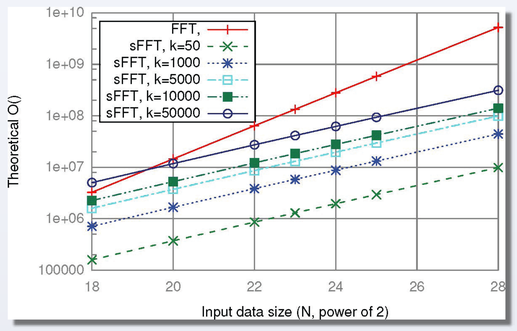


(c) Social graph data

Sparse FFT

***Sparse* FFT (sFFT) computes the Discrete Fourier Transform (DFT) on *sparse* data:**

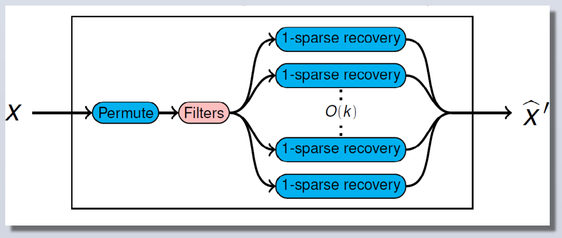
- Achieved the runtime of $O(logn\sqrt{nklogn})$, which is *sublinear* in the signal size *n*
- Faster than FFT ($O(nlogn)$) for *k* is up to $O(n/logn)$



Overview of the order of complexity of both FFT and sFFT algorithms

At a high level, *sparse* FFT can be decomposed into the following steps:

- **Step 1: Random Spectrum Permutation.** The input vector *x* is *permuted* with random parameters.
- **Step 2: Flat Window Functions.** *Window functions* are used as filters to extract a subset of the *n* elements of the signal. This step is crucial to achieve a sublinear runtime: it allows extracting information out of the input vector without touching all *n* elements.
- **Step 3: Subsampled FFT.** Using *subsampling* and a low dimensional FFT, the signal's Fourier coefficients can be binned into a small number of bins.
- **Step 4: Reverse Hash Function for Location Recovery and Magnitude Reconstruction.** By repeating the above steps multiple times and combining the results, the *k* nonzero Fourier coefficients can be found with high probability.



Implementation notes:

- Implements the sequential sFFT using C, more friendly for accelerators and embedded systems
- Uses array-based data structures to store the locations and magnitudes of the Fourier coefficients
- Eliminates loop-carried dependence and non-thread-safety data structures and functions
- Applies loop blocking techniques to improve data locality

Parallel sFFT for Multicore CPUs and GPUs

Parallel computer architectures are *ubiquitous*:

(a) Tianhe-2: Top supercomputer in the world. Number of 3,120,000 cores

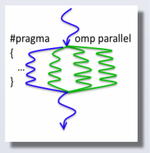
(b) Intel Xeon Processor. From 4 to 16 cores

(c) Texas Instruments C66x multicore DSPs. From 4 to 8 cores.

(d) Nvidia Tesla K20x GPU. Number of 2688 cores

Parallelizing the sFFT multicore CPUs using OpenMP:

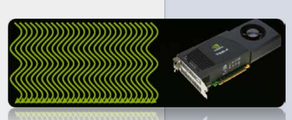
- Index coalescing
- Data affiliated loops
- Data structure optimizations
- Locality Optimizations, blocking



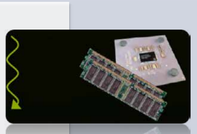
Parallelizing the sFFT for GPUs using CUDA:

- Index mapping
- Fine-grained loop partitioning
- Linear-time quick selection algorithm on GPUs
- Asynchronous data layout transformation

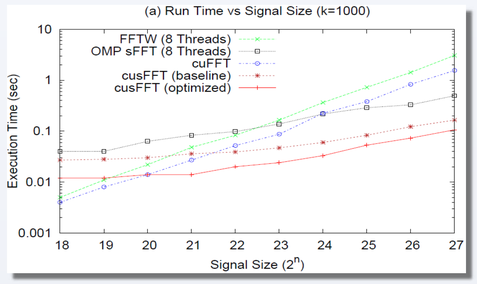

```
__global__ void PermFilterKernel(cuDoubleComplex* d_origx, cuDoubleComplex* d_filter,
cuDoubleComplex* d_x_sampt, int B, int n, int loops, int round, int ai) {
  int i = threadIdx.x + blockIdx.x * blockDim.x;
  int i_mod_B = i & (B-1);
  int ai = d_permute[i/B];
  cuDoubleComplex tmp_value1, tmp_value2;
  for(int j=0; j<round; j++){
    int off = i_mod_B + B*j;
    int index = off*ai &(n-1);
    tmp_value1 = cuCmul(d_origx[index],d_filter[off]);
    tmp_value2 = cuCadd(tmp_value1, tmp_value2);
  }
  d_x_sampt[i] = tmp_value2;
}
```

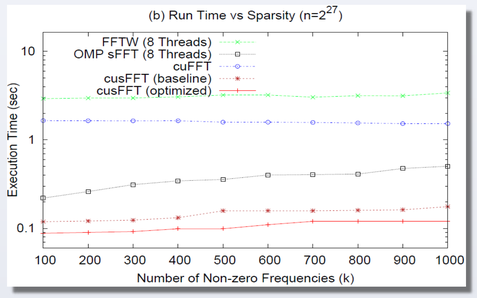


```
// Set CUDA grid and block size
dim3 dimBlock(CUDA_BLOCK);
dim3 dimGrid((B+dimBlock.x-1)/dimBlock.x);
// Allocate space and copy data from host to device
cudaMalloc((void**)&d_origx, n*sizeof(complex_t));
cudaMemcpy(d_origx, x, n*sizeof(complex_t), cudaMemcpyHostToDevice);
// Lunch CUDA kernel
PermFilterKernel<<<dimGrid, dimBlock>>>((cuDoubleComplex*)d_origx,
(cuDoubleComplex*)d_filter, (cuDoubleComplex*)d_x_sampt, B, n,loops,round, ai);
//Copy results back to host
cudaMemcpy(x, d_origx, n*sizeof(complex_t), cudaMemcpyDeviceToHost);
```



Evaluation





- Evaluates the parallel sFFT on both Intel Intel(R) Xeon(R) CPU E5-2640 (8 cores) and Nvidia K20x GPUs.
- OpenMP Parallel sFFT (*Ps*FFT) is over 5x faster than FFTW, one of the most widely used dense FFT libraries for multicore CPUs
- CUDA Parallel sFFT (*cus*FFT) is 14x faster than cuFFT, which is the fastest dense FFT implementation for Nvidia GPUs.

Conclusion and Future Work

- Proposed high-performance parallel sparse FFT algorithms for multicore CPUs and GPUs
- Much faster than state-of-the-art dense FFT implementations
- Explores the parallelism of the sFFT for other massively parallel architectures, including Intel Xeon Phi and Texas Instruments DSP architectures.

Acknowledgements

The authors would like to thanks Piotr Indyk from MIT for helpful discussions. This research has been supported by grants from Shell International E&P Inc. We very much thank Detlef Hohl for his support and insights on this work.

Reference

[1] C.Wang, M. Araya-Polo, S. Chandrasekaran, A. St-Cyr, B. Chapman, and D. Hohl. "Parallel Sparse FFT". *In Proceedings of the 3rd Workshop on Irregular Applications: Architectures and Algorithms, IA3 '13*, pages 10:1–10:8, New York, NY, USA, 2013.

[2] C.Wang, S. Chandrasekaran and B. Chapman, cusFFT: A High-Performance Sparse Fast Fourier Transform Algorithm on GPUs. In *Proceedings of 30th IEEE International Parallel & Distributed Processing Symposium (IPDPS). [To Appear]*

Contact Information

Cheng Wang: cwang35@uh.edu, http://www2.cs.uh.edu/~cwang35
Sunita Chandrasekaran: schandra@udel.edu, https://www.eecis.udel.edu/~schandra/