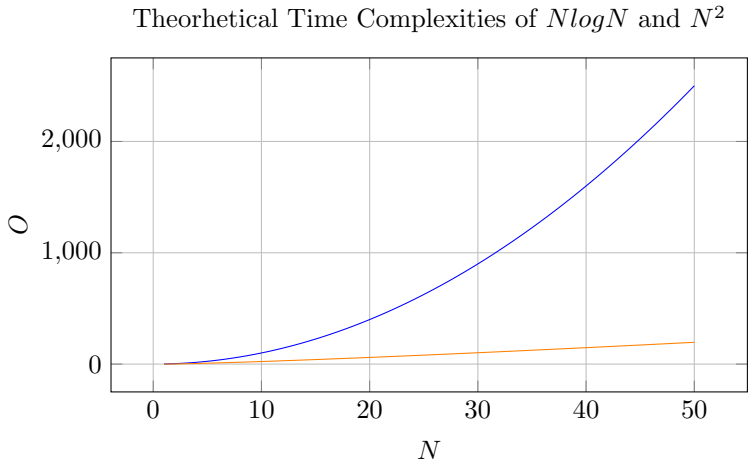
1 Parallelization

In the grand scheme of calculations, there comes a point when there is a limiting factor in which the amount of operations per second becomes an inhibitor. As previously mentioned, the Discrete Fourier Transform has time complexity of $O(N^2)$, where N is the vector size (for an NxN matrix this expands to: $O(N^2N^2) = O(N^4)$). In contrast, the Fast Fourier Transform has time complexity of $O(Nlog(N))$ ($O(N^2log(N))$ when expanded to NxN).



In computers there are two major processing units: Graphics processing unit and Central processing Unit. The CPU is made with a handful of cores while a modern GPU has thousands. Different processes can be offloaded to each core. This is where the time complexity of $O(nlogn)$ makes this so good for modern computers. We can offload the 'split' vectors/matrices to different GPU cores and compute each of these piecewise. Figure 2 below shows a rudimentary theorhetical graph of what this would look like.

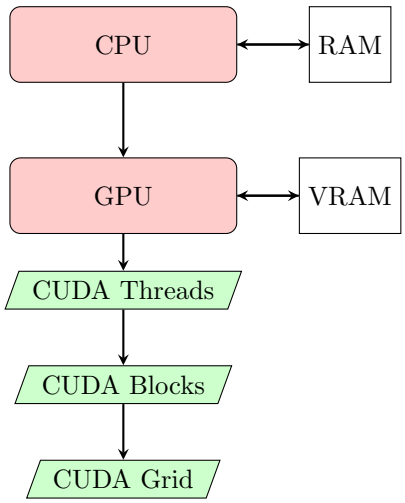


Figure 1: CUDA GPU Parallelization for FFT

To demysitfy this graph, one must first understand that it is not possible to know *exactly* what is happening on any set core. CUDA grids are collections of CUDA Blocks and CUDA Blocks are collections of CUDA Threads. CUDA Threads are the pieces of information that work on some part of the FFT. This can be doing the actual calculations, the combinations, or putting it into a new vector. If the tasks are similar

enough they will get diverted to a CUDA Block. A CUDA Block is a collection of these threads. So this is using the VRAM over threads to share and seperately compute information. The Block works on something called a streaming multiprocessor. This is a collection of CUDA cores on a shared piece of L1,L2 Cache (on processor memory) or VRAM. If the process stream is big enough, multiple Blocks can be executed. This is called a CUDA Grid. A single streaming multiprocessor is only capable of working on Blocks. A CUDA Grid will take multiple streaming multiprocessors are use them to work on the same area of calculations, but divert tasks to Blocks. [1]

The new question is now how can we look at the FFT and Matlab's utilization of CUDA parallel processing to get a better understanding of why it is more effcient.

2 Basic Results

In this section, We will show the results of using the fft command in Matlab and the speed of dynamic arrays.

```
Gsize = 0:15000000:1000000000;
timeVal = zeros(size(Gsize));

% Pre-allocate a large enough GPU array to hold the largest
    matrix
maxSize = max(Gsize);
A_gpu = gpuArray.zeros(maxSize, 1, 'single');
% Pre-allocate GPU memory for column vector

% Loop over different matrix sizes
for index = 1:length(Gsize)
    N = Gsize(index);

    % Create a random matrix of size N on the CPU
    A = single(rand(N, 1));  % Create a column vector of size
        N

    % Transfer the data to the GPU (only the relevant part)
    A_gpu(1:N) = gpuArray(A);
    % Assign to the first N elements of the column vector
    disp(['Processing GPU: Matrix size = ', num2str(N)]);
    % Perform FFT on the GPU (now it's just a vector
        computation)
    tic;
    fft(A_gpu(1:N));
    % Perform FFT on the relevant portion of the GPU array (N
        x 1 vector)
    timeVal(index) = toc;  % Record the time
end

totalGPUtime = sum(timeVal)
```

This is the pseudo-code sort of thing...

References

[1] Pradeep Gupta. "CUDA Refresher: The CUDA Programming Model". In: (2020).