Chan, Ethan Lester
Dolon, John Michael
Lu, Andre Giancarlo
Teng, Adriel Shanlley

Speeding Up DFT and IDFT using CUDA
Integrating Project Milestone 2: Project Update Document

During the Independent Learning Week of DLSU, the group started to form the logic for DFT and IDFT in the C programming language, and has finished doing so. The CUDA integration, however, has not been implemented so far. In this document, code snippets of the logic for DFT and IDFT will be shown along with short explanations of the codes. Note that these algorithms, for now, are implemented separately. Note that the implementation of these codes can change anytime during the development process.

A. Discrete Fourier Transform (DFT) C Code

```c
int main() {
        int N = 10;

        float* x = (float*)malloc(N * sizeof(float));

        for(int i = 0; i < N; i++) {
                x[i] = (float)i;
        }

        dft(x, N);
        free(x);
        return 0;
}
```

Code Snippet 1: Initialization and Calling of DFT Function

To start the DFT code, first, we first initialize the array x with $N = 10$ elements, just for initial testing. The floating-point values inside the array will be set using the current value of index i in the initialization loop in the array. After the initialization of values, we call the DFT function.

```
void dft(float* x, int N) {
        float* xr = (float*)malloc(N * sizeof(float));
        float* xi = (float*)malloc(N * sizeof(float));
        int k, n;
        float theta;

        for (k = 0; k < N; k++) {
                xr[k] = 0;
                xi[k] = 0;
                for (n = 0; n < N; n++) {
                        theta = (2 * PI * k * n) / N;
                        xr[k] = xr[k] + x[n] * cos(theta);
                        xi[k] = xi[k] - x[n] * sin(theta);
                }
                printf("%f + j(%f)\n", xr[k], xi[k]);
        }

        free(xr);
        free(xi);
}
```

Code Snippet 2: DFT Function (3rd implementation from Project Clarifications)

The values passed to the function are the initialized array *x* and the element count *N*. Then, to start the function, we initialized 2 more arrays: *xr*, which will store the real component of the DFT result, and *xi*, which will store the imaginary component of the DFT result. Additionally, for preparation, we initialized the variable *theta*, which will store the result of (2 * PI * *k* * *n*) / *N*.

To start the algorithm, first, we initialize the outer loop that iterates from $k = 0$ to $N - 1$. Then, we set the current index of *xr*[*k*] and *xi*[*k*] to 0 for value initialization. Then, the nested loop executes, iterating from $n = 0$ to $N - 1$. The value of theta is computed next. Then, we solve for the value of *xr*[*k*] and *xi*[*k*], with *xr*[*k*] = *xr*[*k*] + *x*[*n*] * cos(*theta*), and *xi*[*k*] = *xi*[*k*] - *x*[*n*] * sin(*theta*). Then, the function prints the result, and finally, the memory allocated for *xr* and *xi* is freed up. Once the DFT function is completed, the memory allocated for *x* is freed up, and the program finishes execution.

B. Inverse Discrete Fourier Transform (IDFT) C Code

```
int main() {
        int N = 10;
        float* xr = (float*)malloc(N * sizeof(float));
        float* xi = (float*)malloc(N * sizeof(float));

        for (int i = 0; i < N; i++){
                xr[i] = (float)i;
                xi[i] = (float)(i*2);
        }

        idft(xr, xi, N);

        free(xr);
        free(xi);
        return 0;
}
```

Code Snippet 3: Initialization and Calling of IDFT Function

For IDFT, on the other hand, we first initialize the arrays *xr* and *xi* with $N = 10$ elements, just for initial testing. The floating-point values inside the array *xr* will be set using the current value of index *i* in the initialization loop in the array, and the floating-point values inside the array *xi* will be set using the current value of index *i* * 2 in the initialization loop in the array. After the initialization of values, we call the IDFT function.

```
void idft(float* xr, float* xi, int N) {
        float* y = (float*)malloc(N * sizeof(float));
        float theta;

        printf("IDFT:\n");
        for (int n = 0; n < N; n++){
                for (int k = 0; k < N; k++){
                        theta = (2 * PI * k * n)/N;
                        y[n] = y[n] + xr[k] * cos(theta) - xi[k] * sin(theta);
                }
                y[n] = y[n]/N;
                printf("y[%d] = %.2f\n", n, y[n]);
        }

        free(y);
}
```

Code Snippet 4: IDFT Function (3rd implementation from Project Clarifications)

The values passed into the IDFT function are the initialized arrays *xr* and *xi* followed by the element count *N*. In the start of the function, we allocate space for the *y* array, which will contain the results of the IDFT. Then we initialize variable *theta* as a float.

The algorithm for IDFT starts with a for loop that iterates from $n = 0$ until $N - 1$. Inside this loop, there is a nested for loop that iterates from $k = 0$ to $N - 1$, and it first computes for theta by performing (2 * PI * *k* * *n*) / *N*. With theta, we can compute for the value of $y[n]$ by doing $y[n] + xr[k] * \cos(theta) + (- xi[k] * \sin(theta))$. Outside the nested for loop, we divide $y[n]$ with *N* to get the final value of $y[n]$, which is then printed out. Lastly, the memory allocated for *y*, *xr*, and *xi* is freed up and the program completes its execution.