De La Salle University, Manila
Computer Technology Department

Term 2, A.Y. 2024-2025

CEPARCO - Integrating Project

Accelerating Fast Walsh-Hadamard Transform and Inverse Fast Walsh-Hadamard Transform using SIMT

GROUP 4

Arca, Althea Denisse G.

Co Chiong, Sean F.

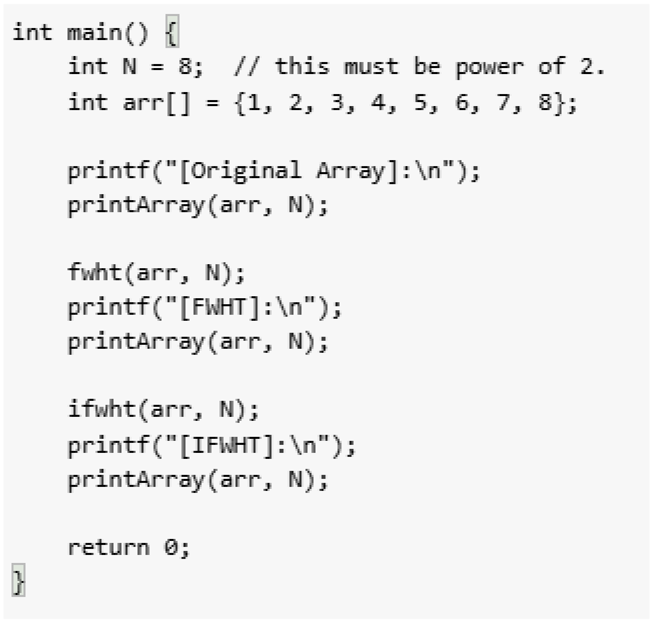
Uy, Wesley King C.

Mar 17, 2025

Accelerating Fast Walsh-Hadamard Transform and Inverse Fast Walsh-Hadamard Transform using SIMT

This document presents the progress of our Integrating Project for the course. As of this milestone, our group has successfully implemented the core logic of the Fast Walsh-Hadamard Transform (FWHT) and Inverse Fast Walsh-Hadamard Transform (IFWHT) in C. However, the CUDA integration remains incomplete and requires further revisions to achieve full functionality. It is important to note that our implementation may change throughout the duration of the project as we continue brainstorming ways to enhance its uniqueness. Future iterations will focus on optimizing the CUDA integration to fully leverage SIMT (Single Instruction, Multiple Threads) for improved computational efficiency.

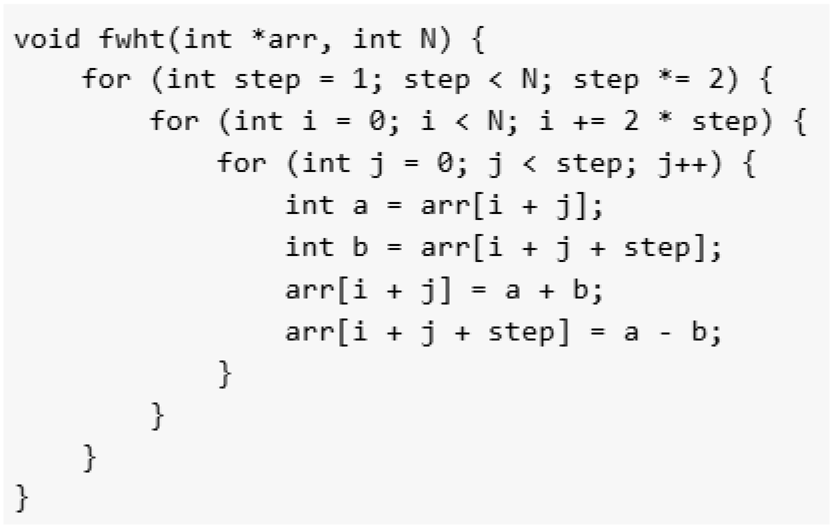
A. Fast Walsh-Hadamard Transform (FWHT) C Code



```
int main() {
    int N = 8;  // this must be power of 2.
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};

    printf("[Original Array]:\n");
    printArray(arr, N);

    fwht(arr, N);
    printf("[FWHT]:\n");
    printArray(arr, N);

    ifwht(arr, N);
    printf("[IFWHT]:\n");
    printArray(arr, N);

    return 0;
}
```

Code Snippet 1: Initialization and Calling of FWHT AND IFWHT Function

To start the FWHT code, we first initialize the array ***arr*** with ***N = 8*** elements for initial testing. The integer values inside the array are set sequentially from 1 to 8. After initializing the values, we call the FWHT function.



```c
void fwht(int *arr, int N) {
    for (int step = 1; step < N; step *= 2) {
        for (int i = 0; i < N; i += 2 * step) {
            for (int j = 0; j < step; j++) {
                int a = arr[i + j];
                int b = arr[i + j + step];
                arr[i + j] = a + b;
                arr[i + j + step] = a - b;
            }
        }
    }
}
```

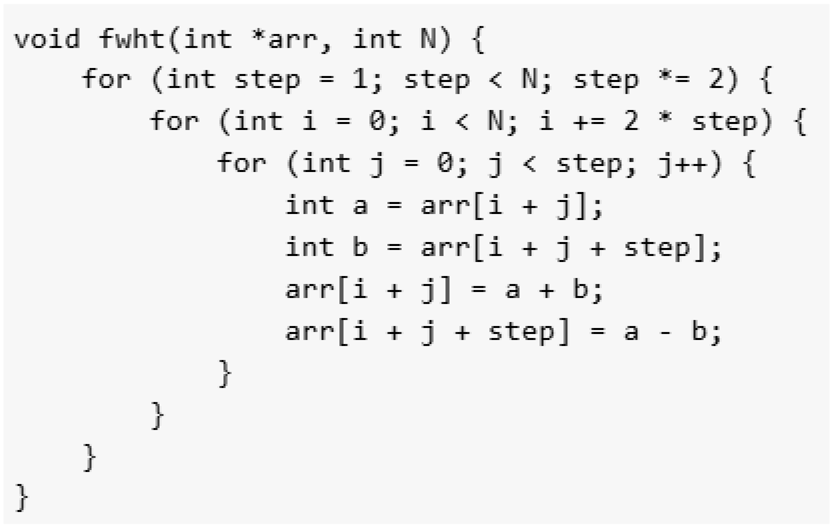
Code Snippet 2: FWHT Function

The FWHT algorithm follows a recursive approach, iterating through the data in log-scale steps. The transformation is performed using a series of **butterfly operations**, which manipulate pairs of elements at different strides. The outermost loop iterates through different step sizes, which double at each iteration (starting from 1 and increasing to N/2N/2N/2). Inside this loop, another nested loop iterates through the elements of the array in increments of twice the current step size. The innermost loop processes adjacent pairs of elements using the Walsh-Hadamard transform formula:

$$A=X+Y$$

$$B=X-Y$$

where ***X*** and ***Y*** are two adjacent values being processed, and ***A***, ***B*** are the transformed outputs. These operations effectively mix and redistribute data across the array, following a pattern similar to the Fast Fourier Transform (FFT), but without complex numbers. Once all iterations are completed, the transformed data replaces the original values in the array.



```c
void fwht(int *arr, int N) {
    for (int step = 1; step < N; step *= 2) {
        for (int i = 0; i < N; i += 2 * step) {
            for (int j = 0; j < step; j++) {
                int a = arr[i + j];
                int b = arr[i + j + step];
                arr[i + j] = a + b;
                arr[i + j + step] = a - b;
            }
        }
    }
}
```

Code Snippet 3: IFWHT Function

The inverse transformation follows the same computational steps as FWHT, as the Walsh-Hadamard transform is self-inverse. This means that applying the FWHT twice results in the original input sequence (up to a normalization factor). To obtain the correct inverse, we first apply FWHT to the transformed array and then divide each element by ***N*** to restore the original values. The function maintains the same structure as the forward transformation, ensuring that all values are correctly inverted back to their original form.

B. Fast Walsh-Hadamard Transform (FWHT) C Code with CUDA Integration

Following the completion of the C implementation, the CUDA integration is currently in progress. However, the group has encountered several errors that require changes and revision. Additionally, we are still evaluating potential topics for the uniqueness of our implementation from existing approaches. These enhancements have not yet been incorporated into the code below:

```c
#include "cuda_runtime.h"

#include "device_launch_parameters.h"



#include <stdio.h>

#include <stdlib.h>
```

```
#include <windows.h>
#include <math.h>


__global__ void fwht_kernel(double* data, size_t N) {
    int index = blockIdx.x * blockDim.x + threadIdx.x;
    int stride = blockDim.x * gridDim.x;

    for (int len = 1; len < N; len *= 2) {
        for (int i = index; i < N; i += stride) {
            int pairIndex = i ^ len;
            if (pairIndex > i) {
                double temp = data[i];
                data[i] = temp + data[pairIndex];
                data[pairIndex] = temp - data[pairIndex];
            }
        }
        __syncthreads();
    }
}

__global__ void ifwht_kernel(double* data, size_t N) {
    fwht_kernel<<<gridDim.x, blockDim.x>>>(data, N);
    __syncthreads();
    int index = blockIdx.x * blockDim.x + threadIdx.x;
    if (index < N) {

        data[index] /= (double)N;

    }

}


int main() {

    LARGE_INTEGER StartingTime, EndingTime, ElapsedMicroseconds;

    LARGE_INTEGER Frequency;

    QueryPerformanceFrequency(&Frequency);

    double total_time, ave_time;

    const size_t ARRAY_SIZE = 1 << 17;

    const size_t ARRAY_BYTES = ARRAY_SIZE * sizeof(double);

    const size_t loope = 5;


    int device = -1;

    cudaGetDevice(&device);

    double* x;

    cudaMallocManaged(&x, ARRAY_BYTES);


    for (int i = 0; i < ARRAY_SIZE; i++) {

        x[i] = (double)i;

    }


    size_t numThreads = 1024;

    size_t numBlocks = (ARRAY_SIZE + numThreads - 1) / numThreads;
    printf("*** FWHT ***\n");

    printf("numElements = %lu\n", ARRAY_SIZE);

    printf("numBlocks = %lu, numThreads = %lu \n", numBlocks, numThreads);


    total_time = 0.0;

    for (size_t i = 0; i < loope; i++) {

        QueryPerformanceCounter(&StartingTime);

        fwht_kernel<<<numBlocks, numThreads>>>(x, ARRAY_SIZE);

        cudaDeviceSynchronize();

        QueryPerformanceCounter(&EndingTime);

                       total_time   +=   ((double)((EndingTime.QuadPart   -
StartingTime.QuadPart) * 1000000 / Frequency.QuadPart)) / 1000;

    }


    ave_time = total_time / loope;

    printf("Time taken for FWHT: %f ms\n\n", ave_time);


    total_time = 0.0;

    for (size_t i = 0; i < loope; i++) {

        QueryPerformanceCounter(&StartingTime);

        ifwht_kernel<<<numBlocks, numThreads>>>(x, ARRAY_SIZE);

        cudaDeviceSynchronize();

        QueryPerformanceCounter(&EndingTime);

                       total_time   +=   ((double)((EndingTime.QuadPart   -
StartingTime.QuadPart) * 1000000 / Frequency.QuadPart)) / 1000;

    }
    ave_time = total_time / loope;

    printf("Time taken for IFWHT: %f ms\n\n", ave_time);



    cudaFree(x);

    return 0;

}
```

C. Future goals and Implementation

1. Uniqueness

   The primary goal for the upcoming week is to innovate and expand the capabilities of the Fast Walsh Hadamard transformation Exploring more tasks that can be done instead of generic arrays transformation.

2. Benchmarking

   Our benchmarking will be using CUDA profiler as a gauge to check any potential bottlenecks within the program and comparing different ways to optimize the program and studying the optimization.

3. Documentation and Final Report

   This final report will be a comprehensive summary of the project’s findings. The documentation will include the challenges the team faced during development, along with the solution. The report will also include the benchmark performance results of the program. The will provide additional knowledge and potential application of the FWHT in future projects.