```
__global__ void find_roots_kernel(float *A, float *B, float *C, float *X0,
                                  float *X1, float *FX0, float *FX1, int n) {

    int row = (blockDim.y)*(blockIdx.y) + threadIdx.y;
    int col = (blockDim.x)*(blockIdx.x) + threadIdx.x;
    int id = (gridDim.x)*(blockDim.x)*row + col;
    float d;

    d = sqrt(B[id]*B[id] - 4*A[id]*C[id]); // √(b²-4ac) 를 의미.
    X0[id] = (-B[id]-d)/(2*A[id]); // 항상 서로 다른 두 실근을 가진다는 전제가 있으므로 d>0 이다.
    X1[id] = (-B[id]+d)/(2*A[id]); // (-b-√(b²-4ac))/2a < (-b+√(b²-4ac))/2a 이므로 '-'로 연결된 근이 더작다.
                                   //                                          (X0 < X1)

    FX0[id] = (A[id]*X0[id] + B[id])*X0[id] + C[id]; // ax²+bx+c=0 의 식에서 x에 위에서 구한
    FX1[id] = (A[id]*X1[id] + B[id])*X1[id] + C[id]; // 근을 대입한 함숫값이다. 이 나와야 한다.
}

void find_roots_GPU(float *A, float *B, float *C, float *X0, float *X1, float *FX0, float *FX1)
{   float *d_A, *d_B, *d_C, *d_X0, *d_X1, *d_FX0, *d_FX1;
    size_t size;

    size = n*sizeof(float); // device Memory의 크기를 지정
    CUDA_CALL(cudaMalloc(&d_A, size)); // device Memory 할당
    CUDA_CALL(cudaMemcpy(d_A, A, size, cudaMemcpyHostToDevice)); // host Memory 버퍼의 값을
                                                                 // device Memory로 복사.
    CUDA_CALL(cudaMalloc(&d_B, size)); // device Memory 할당
    CUDA_CALL(cudaMemcpy(d_B, B, size, cudaMemcpyHostToDevice)); // host Memory 버퍼의 값을
                                                                 // device Memory로 복사.
    CUDA_CALL(cudaMalloc(&d_C, size)); // device Memory 할당
    CUDA_CALL(cudaMemcpy(d_C, C, size, cudaMemcpyHostToDevice)); // host Memory 버퍼의 값을
                                                                 // device Memory로 복사.

    CUDA_CALL(cudaMalloc(&d_X0, size)); ////////////
    CUDA_CALL(cudaMalloc(&d_X1, size)); //  device Memory  //
    CUDA_CALL(cudaMalloc(&d_FX0, size)); //      할당      //
    CUDA_CALL(cudaMalloc(&d_FX1, size)); ////////////

    int n1=BLOCK_SIZE; int n2 = n / BLOCK_SIZE;

    dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);
    dim3 dimGrid(n1/dimBlock.x, n2/dimBlock.y);

    CHECK_TIME_INIT_GPU()  CHECK_TIME_START_GPU() // GPU 시간측정
    find_roots_kernel <<< dimGrid, dimBlock >>> (d_A, d_B, d_C, d_X0, d_X1, d_FX0, d_FX1); // 커널 수행
    CHECK_TIME_END_GPU(device_time)  CHECK_TIME_DEST_GPU() // GPU 시간측정

    CUDA_CALL(cudaMemcpy(X0, d_X0, size, cudaMemcpyDeviceToHost)); ////////////////////
    CUDA_CALL(cudaMemcpy(X1, d_X1, size, cudaMemcpyDeviceToHost)); //  결과값을 Host  //
    CUDA_CALL(cudaMemcpy(FX0, d_FX0, size, cudaMemcpyDeviceToHost)); // Memory로 복사함. //
    CUDA_CALL(cudaMemcpy(FX1, d_FX1, size, cudaMemcpyDeviceToHost)); ////////////////////
}
```

```
__global__ void Fibonacci_kernel(int *x, int *y){
    int row=(blockDim.y)*(blockIdx.y)+threadIdx.y;
    int col=(blockDim.x)*(blockIdx.x)+threadIdx.x;
    int id = (gridDim.x)*(blockDim.x)*row + col;

    float sqrt_5 = sqrt(5.0); //√5
    float tmp0 = 1.0, tmp1 = 1.0; float x_0, x_1;
    x_0 = (1+sqrt_5)/2.0; x_1 = (1-sqrt_5)/2.0;

    for(int i=0; i < x[id]; i++){
        tmp0 *= x_0; tmp1 *= x_1; // tmp0 = ((1+√5)/2)^x[id], tmp1 = ((1-√5)/2)^x[id]
    }
    y[id] = (int)((tmp0-tmp1)/sqrt_5+0.5); // 피보나치 수열의 n번째 항
}

cudaError_t Fibonacci_GPU(int *x, int*y){
    cudaError_t cudaStatus = cudaSetDevice(0);
    int *d_x, *d_y;
    size_t size = n*sizeof(int); // device Memory의 크기를 지정.
    int n1 = BLOCK_SIZE, n2 = n/BLOCK_SIZE;

    CUDA_CALL(cudaMalloc(&d_x, size)); //device Memory 할당
    CUDA_CALL(cudaMemcpy(d_x, x, size, cudaMemcpyHostToDevice)); // host Memory의 값을 device Memory로 복사.
    CUDA_CALL(cudaMalloc(&d_y, size)); // device Memory 할당.

    dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);
    dim3 dimGrid(n1/dimBlock.x, n2/dimBlock.y);

    CHECK_TIME_INIT_GPU(); // GPU 시간측정
    CHECK_TIME_START_GPU(); // GPU 시간측정
    Fibonacci_kernel <<< dimGrid, dimBlock >>>(d_x,d_y); // 커널 수행
    CHECK_TIME_END_GPU(device_time); // GPU 시간측정
    CHECK_TIME_DEST_GPU(); // GPU 시간측정

    CUDA_CALL(cudaDeviceSynchronize()); // 커널이 끝날 때까지 기다림. 동기화를 위한 것.
    CUDA_CALL(cudaMemcpy(y,d_y,size,cudaMemcpyDeviceToHost)); // 계산한 피보나치 수열을 Host Memory로 복사함.

    return cudaStatus;
}
```

```
__global__ void sum_n_elements_kernel(int *d_Array, int *d_sum, int N, int Nf){
    int row = (blockDim.y)*(blockIdx.y) + threadIdx.y;
    int col = (blockDim.x)*(blockIdx.x) + threadIdx.x;
    int id = (gridDim.x)*(blockDim.x)*row + col;

    for(int i=0; i<Nf; i++){                    // id번째 원소를 기준으로 왼쪽 원소 Nf개를 더한다.
        if((id-i)>=0) d_sum[id] += d_Array[id-i]; // 유효한 메모리 주소인지 확인
    }
    for(int i=1; i<Nf; i++){                    // id번째 원소를 기준으로 오른쪽원소 (Nf-1)개를 더한다.
        if(id+i<N) d_sum[id] += d_Array[id+i]; // 유효한 메모리 주소인지 확인
    }
}
    // 최종적으로 d_sum[id] = Σ_{k=i-Nf}^{i+Nf} d_Array[k] 의 값을 갖는다.

cudaError_t sum_n_elements_GPU(int *P_Array, int *P_sum, int Nf){
    cudaError_t cudaStatus = cudaSetDevice(0);
    int *d_Array, *d_sum;
    size_t size = N * sizeof(int); // device Memory의 크기 결정
    int n1 = BLOCK_SIZE, n2 = N/BLOCK_SIZE;

    CUDA_CALL(cudaMalloc(&d_Array, size)); // device Memory 할당
    CUDA_CALL(cudaMemcpy(d_Array, P_Array, size, cudaMemcpyHostToDevice)); // host Memory의 값을
                                                                           device Memory로 복사
    CUDA_CALL(cudaMalloc(&d_sum, size)); // device Memory 할당

    dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);
    dim3 dimGrid(n1/dimBlock.x, n2/dimBlock.y);

    CHECK_TIME_INIT_GPU(); // /// /// /// ///
    CHECK_TIME_START_GPU(); // GPU 시간측정 //
                            // /// /// ///
    sum_n_elements_kernel <<< dimGrid, dimBlock >>> (d_Array, d_sum, N, Nf); // 커널 수행
    CHECK_TIME_END_GPU(device_time); // GPU 시간측정
    CHECK_TIME_DEST_GPU(); // GPU 시간측정

    CUDA_CALL(cudaDeviceSynchronize()); // 커널이 끝날때까지 기다림
    CUDA_CALL(cudaMemcpy(P_sum, d_sum, size, cudaMemcpyDeviceToHost)); // 계산한 부분적인 수열의 합을
                                                                       Host Memory로 복사함.
    return cudaStatus;
}
```