CSC453
Parallel Processing
Project

Ammar Alamri Std#: 438104833

Bitonic Parallel Sort Implementation:

Note: I tried my best to replace module (%), division and pow() operations, with shift and bitwise operations, to increase performance and provide a more unique solution, since this might affect readability I also added comments showing how the operations would be done using traditional methods.

```
// BitonicSorter works only on arrays of size 2^n.
__global__ void BitonicSorter(int* in, int N, int max_iterations) {
	int index = (threadIdx.x);
	int seq_length = 0, shift = 0;

	for (int i = 1; i < max_iterations + 1; i++) { // for steps, for stages.
		for (int j = 1; j < i + 1; j++) {

			// 1 << i-j+1 = pow(2, i-j+1).
			seq_length = 1 << i - j + 1;

			// seq_length / 2.
			shift = seq_length >> 1;

			// index % seq_length < shift.
			if ((index & (seq_length - 1)) < shift) {

				//  if (index / pow(2,i)) is even.
				if ((index >> i & 1) == 0) {
					if (in[index] > in[index + shift]) {
						int temp = in[index];
						in[index] = in[index + shift];
						in[index + shift] = temp;
					}
				}
				else if (in[index] < in[index + shift]) {
					int temp = in[index];
					in[index] = in[index + shift];
					in[index + shift] = temp;
				}
			}
			__syncthreads();
		}
	}
}
```

```
int main() {
	/* =main function= */
	int N = 50;
	int fillers = 0, max_iterations = 0, altSize = 0;

	/* allocation and array generation */
	int* a, * b, * d_a;
	int size = sizeof(int) * N;

	a = (int*)malloc(size);
	b = (int*)malloc(size);

	cudaMalloc(&d_a, size);

	random_ints(a, N);

	// Checking if size is 2^n.
	for (int i = 0; i < 20; i++) {
		// altSize = pow(2, i)
		altSize = 1 << i;

		if (N <= altSize) {
			fillers = altSize - N;
			max_iterations = i;
			break;
		}
	}
	// if original size not 2^n, we add fillers to the end of the array.
	if (fillers > 0) {
		size = sizeof(int) * altSize;
		realloc(a, size);

		for (int i = 0; i < fillers; i++)
			a[N + i] = INT_MAX;
	}

	cudaMalloc(&d_a, size);
	cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);

	BitonicSorter << <1, altSize >> > (d_a, altSize, max_iterations);
	cudaDeviceSynchronize();

	cudaMemcpy(a, d_a, size, cudaMemcpyDeviceToHost);

	// Trim output to original size N.
	memcpy(b, a, sizeof(int) * N);
	printf("After sorting, array is: "); printArray(b, N);
	return 0;
}
/* Helper functions */
void random_ints(int*& array, int size) {
	for (int i = 0; i < size; i++)
		array[i] = rand() % size;
}

void printArray(int* array, int size) {
	for (int i = 0; i < size - 1; i++)
		printf("%d, ", array[i]);
	printf("%d.\n", array[size - 1]);
}
```