CUDA Homework Assignment — Group 6

ACA2025

Homework Title: Bitonic Sort on GPU

1 Objective

Implement the Bitonic Sort algorithm using CUDA to sort large arrays efficiently. This parallel sorting network is ideal for GPUs because of its structured, predictable access patterns.

2 CUDA Implementation Requirements

- Implement Bitonic Sort using CUDA kernels.
- Work on array sizes that are powers of two (e.g., 1024, 2048).
- Use shared memory to load array segments per thread block.
- Synchronize threads using `__syncthreads()` as needed.
- Structure your code to clearly separate memory allocation, kernel execution, and cleanup.

3 Python Integration

- Connect your CUDA kernel to Python using one of the following:
  - `ctypes` or `cffi`
  - `pybind11`
  - `cupy` (if you follow NumPy-like syntax)
- Provide a Python script that:
  - Generates random input arrays using NumPy.
  - Calls your CUDA sort function.
  - Validates results by comparing against `numpy.sort()`.
  - Benchmarks and compares performance.

4 Evaluation Criteria

- **Correctness**: Sorted array matches NumPy output.
- **Performance**: Demonstrate GPU speedup for large inputs.
- **Code Quality**: Clean, modular code with comments.
- **Integration**: Fully working Python-CUDA pipeline.

5 Important Note

Your code will be tested exactly once using your instructions. If your code fails to compile or run, you will lose your grade for this assignment. Ensure your README contains all required build and run commands.

6 Deliverables

- `bitonic_sort.cu` — CUDA kernel code.
- `main.py` — Python integration script.
- `main.py` must run both CPU and GPU versions for multiple input sizes and generate a performance comparison graph (e.g., saved as `comparison_plot.png`).
- `README.md` — Must include the exact command to run your code:
  - Example: `python3 main.py`
- Benchmark results or timing plots.

7 Resources

- Bitonic Sort Background
- CUDA C++ Programming Guide (General Reference)