Prepare a design spec which contains

1. The specific equations you intend to parallelize (LaTeX or MathType formatted)
2. Sequential reference code implementation of algorithm(s)
   1. initial performance timing baseline for sequential version (to be used for verification against parallel code)
3. Analysis of algorithm complexity and estimated speedup
   1. Use techniques such as arithmetic intensity, algorithmic complexity, speedup estimate, Amdahl's law, roofline model analysis of problem
4. Pseudo-code for parallel OpenCL kernels
5. List of primary references/sources for your algorithms, equations, and any models

Design Spec

* k Nearest Neighbor (KNN) algorithm
  + Euclidean
  + Sorting (Batcher\_odd–even\_mergesort, size , and depth

For data in 2D, sequential.cpp has the distance and the odd-even merge sort algorithm written in cpp. It can be used as a verification against parallel code.

Complexity and est. speedup

AI = computational work / communication = W(work) / Q(memory traffic)

|  |  |  |
| --- | --- | --- |
|  | Complexity | AI |
| Distance matrix calculation | O(n) | , d=2  (for each test point) |
| Sorting | , non-parallel  (Best, Worst, Average) Performance in parallel = | (for each test point) |

Speed up = Sp = , where T1 latency of program with one worker, and Tp is the latency of program with P workers

Amdahl’s Law = Sp , Wpar in this case is parallel portion of the W, which is all the comparisons.

Pseudo-code for parallel OpenCL kernels

Note:

Choose different k values. K < sqrt(n) theoretical properties

Weighted Euclidean

Reference

“An Efficient FPGA Implementation for odd-even sort based KNN algorithm using OpenCL”, Hai Peng, Letian Huang

“OpenCL Parallel Programming Development Cookbook” by Raymond Tay (2013)

<http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/networks/oemen.htm>

<https://en.wikipedia.org/wiki/Batcher_odd%E2%80%93even_mergesort>