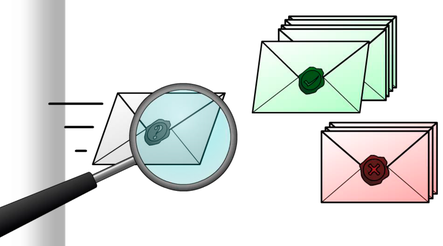


Massively Parallel kNN using CUDA on Spam-Classification

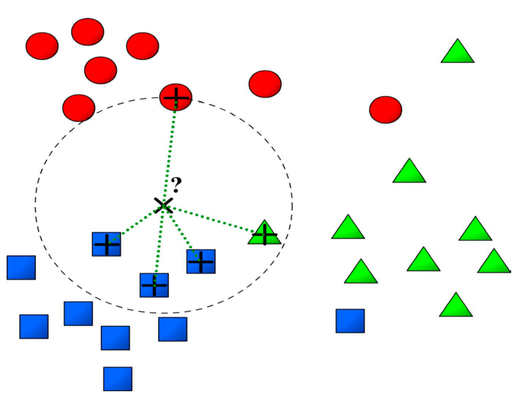
Joshua M. Smithrud[1], Patrick McElroy[1], Razvan Andonie[2]

[1] Undergraduate Researcher, [2] Faculty Advisor

Department of Computer Science, Central Washington University

k-Nearest Neighbors Algorithm

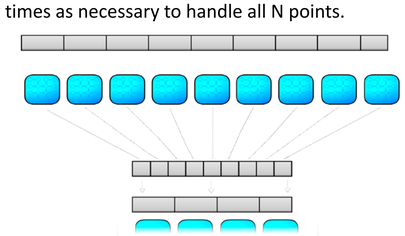
- Classification algorithm
- Generally regarded as one of the most important machine-learning algorithms.
- How it works
  - Given a dataset of points (training set) with known attributes and known classification, a point with known attributes but unknown classification is classified based on a weighted average of its most similar points among the training set. In terms of application and complexity, the algorithm can be broken down as follows:
    - Distance Calculation Phase
      - Measure the (Euclidian) distance from the query point to all N points in the training set.
      - O (N)
      - Embarrassingly parallelizable
    - Sorting Phase
      - Sort all N elements of the Training Set by their distance from the query point.
      - Using Introsort: O (N * log (N)) - average
      - Complexity-driver
    - Classification Phase
      - Determine query point's classification based on a weighted average of the k-first elements of the sorted set.
      - O (k)
      - k is very small (we used 10), and is therefore negligible in terms of complexity



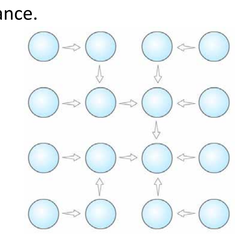
- Overall complexity: O (N * log(N)) – average
- Our application focus: **Spam-Classification**

Our Solution

- Massively Parallel Reduction using CUDA
  - Given the following:
    - A training set with N points
    - A GPU with a maximum of T concurrently running threads, t possible threads-per-block, where the number of blocks used is $b = \lceil T / t \rceil$
  - We divide the N training set points into a number of blocks (b) such that each block is utilizing its maximum number of threads (t).
    - If N > T, then, we loop as many times as necessary to handle all N points.



  - Each block reduces its <= t points by comparing adjacent pairs, until the minimum k points within the block are found. Each block returns its k points with minimum distance.



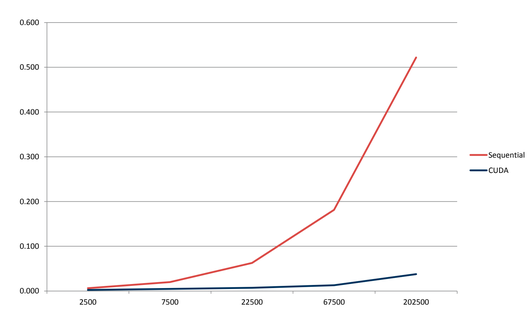
  - This set of k * b points is then passed back into this same process, which is repeated until the absolute k-closest points are found, at which point the object in question is classified based on a weighted average of the remaining k.
- This Parallel Reduction method yields a complexity of approximately $\lceil N / T \rceil * k * \log_2(t) * \log_{t/k}(T)$ + Overhead.

Theoretical Results

- Our GPU allowed for:
  - T = 30 multiprocessors * 32 concurrent warps * 32 threads/warp = 30720
  - t = 512
- We used k = 10
- Ignoring the overhead yielded from copying data from main memory to GPU memory, the resulting theoretical complexity obtained is $\lceil N / 30720 \rceil * 10 * \log_2(512) * \log_{51.2}(30720)$
  - Approximately 236.281 * $\lceil N / 30720 \rceil$

Experimental Results

- In the real world, taking overhead into account, our results are still very promising.
- The following graph shows the experimental difference between a sequential implementation using Introsort, and our Parallel Reduction in CUDA.
  - # points in the Training Set VS time (seconds).



More About Us

- Joshua M. Smithrud
  - Has a B.A. in Mathematics from the University of Washington
  - Graduating in March with a B.S. in Computer Science
- Patrick McElroy
  - Graduating in March with a B.S. in Computer Science
- Razvan Andonie
  - Professor and Director of the Computational Science Master's Program at Central Washington University

Our Research Done Using…

- OS: Windows 7 Ultimate
- CPU: Intel i7 950
- Memory: 12GB DDR3 1600
- GPU: GeForce GTX 275
  - GPU Engine Specs
    - CUDA Cores: 240
    - Graphics Clock (MHz): 633MHz
    - Processor Clock (MHz): 1404MHz
    - Texture Fill Rate (billion/sec): 50.6
  - Memory Specs
    - Memory Clock: 1134MHz
    - Standard Memory Config: 896MB
    - Memory Interface: GDDR3
    - Memory Interface Width: 448-bit
    - Memory Bandwidth (GB/sec): 127.0