

# CIS 700/010: Assignment 2

Suresh Venkatasubramanian

## 1 Problem 1: Matrix Operations (40)

1. Implement a GPU algorithm to add two  $n \times n$  matrices. Use the simple approach that runs in  $n/b$  passes.
2. Implement a GPU program for multiplying two dense  $n \times n$  matrices. Use the scheme that employs  $n^2$  memory, runs in  $n/b$  passes,

In both cases,  $b$  represents the number of arithmetic operations you perform per pass. Find the value of  $b$  that maximises the efficiency of the resulting program and report it for both operations.

## 2 Problem 2: Reverse Nearest Neighbours (60)

When dealing with *mobile servers* that serve clients in a domain, a common problem is determining the load on a server. One approach to assigning clients to servers is to assign a client to the server that is currently nearest to it.

Think of clients ( $C$ ) and servers ( $S$ ) being sets of moving points in two dimensions. At any time, we would like to determine how loaded each server is. The formula for calculating the load on a server is as follows: for each server  $s \in S$ , find the set of clients  $C(s)$  consisting of all  $c \in C$  that would consider  $s$  their nearest neighbour among all servers. Formally,

$$C(s) = \{c \in C \mid d(c, s) = \min_{s' \in S} d(c, s')\}$$

The set  $C(s)$  is called the *reverse nearest neighbour* set of  $s$ , because instead of it containing the points closest to  $s$ , it contains the points for whom  $s$  is closest to them.

**Problem:** Design a GPU algorithm that takes as input a set of client and server points, and renders on a 2D window the set of all servers, with each server represented by a colored circle whose center is the server location and radius equals  $|C(s)|$ . The colors should be unique; one per center.

The algorithm should be efficient: As clients and servers move, the computation should be updated appropriately. Assume a model where each client and server starts at a random location with a random velocity, and then bounces off the walls of the window.

### 2.1 Extra Credit (10)

The fastest three implementations (measured in terms of the frame rate for processing 100 points and 15 servers), will get 10 bonus points. Speed will be measured on dink. We will provide a frame rate measurement tool.

## 3 Extra credit: Addition in $\log n$ passes (20)

Implement a scheme for adding all the numbers in a 2D  $n \times n$  texture in  $O(\log n)$  passes.