CSE590: Special Topics Course (Supercomputing), Spring 2012 **Date:** April 1

Homework #3

(Due: April 14)

Task 1. [25 Points] Nonrecursive Prefix Sums. Given an array $A[1:n]$ of n elements with a binary associative operation $\oplus$, the *Prefix Sums* problem asks you to compute another array $S[1:n]$, where $S[i] = A[1] \oplus A[2] \oplus \ldots \oplus A[i]$ for $i \in [1, n]$.

(a) [**25 Points**] Implement the following nonrecursive Prefix Sums algorithm (Par-Prefix-Sums) in CUDA for $\oplus = min$, and optimize as much as possible. Assume for simplicity that $n = 2^k$ for some integer $k \geq 0$. For integer array A (initialized with random values), find the value of k larger than 10 for which your CUDA implementation gives the best speedup w.r.t. the serial CPU implementation of Par-Prefix-Sums. Only consider the values of k for which A does not overflow the RAM. Find the best value of k when the time for transferring data between host and device memories is included in the running time of your CUDA implementation, and also when that time is not included. Explain your findings.

```
Par-Prefix-Sums( $A[1:n]$, $S[1:n]$, $\oplus$ )        {assume $n = 2^k$ for some integer $k \geq 0$}
    array $B[1:2n-1]$, $C[1:2n-1]$
    parallel for $i \leftarrow 1$ to $n$ do $B[i] \leftarrow A[i]$
    $m \leftarrow n$, $t \leftarrow 0$
    for $h \leftarrow 1$ to $k$ do
        $s \leftarrow t$, $t \leftarrow t + m$, $m \leftarrow \frac{m}{2}$
        parallel for $i \leftarrow 1$ to $m$ do
            $B[t+i] \leftarrow B[s+2i-1] \oplus B[s+2i]$
    for $h \leftarrow k$ downto $0$ do
        parallel for $i \leftarrow 1$ to $m$ do
            if $i = 1$ then $C[t+i] \leftarrow B[t+i]$
            else if $i = even$ then $C[t+i] \leftarrow C[s+\frac{i}{2}]$
                else $C[t+i] \leftarrow C[s+\frac{i-1}{2}] \oplus B[t+i]$
        $m \leftarrow 2m$, $s \leftarrow t$, $t \leftarrow t - m$
    parallel for $i \leftarrow 1$ to $n$ do $S[i] \leftarrow C[i]$
```

Task 2. [75 Points] Pairwise Sequence Alignment. Sequence alignment plays a central role in biological sequence comparison, and can reveal important relationships among organisms. Given two strings $X = x_1x_2\ldots x_m$ and $Y = y_1y_2\ldots y_n$ over a finite alphabet Σ, an *alignment* of X and Y is a matching M of sets $\{1, 2, \ldots, m\}$ and $\{1, 2, \ldots, n\}$ such that if $(i, j), (i', j') \in M$ and $i < i'$ hold then $j < j'$ must also hold. The i-th letter of X or Y is said to be in a *gap* if it does not appear in any pair in M. Given a *gap penalty* g and a mismatch cost $s(a,b)$ for each pair $a, b \in \Sigma$, the *basic (global) pairwise sequence alignment problem* asks for a matching M_{opt} for which $(m + n - |M_{opt}|) \times g + \sum_{(a,b) \in M_{opt}} s(a,b)$ is minimized.

The formulation of the basic sequence alignment problem favors a large number of small gaps while real biological processes favor the opposite. The alignment can be made more realistic by using an *affine gap penalty* which has two parameters: a *gap introduction cost* g_i and a *gap extension cost* g_e. A run of t gaps incurs a total cost of $g_i + g_e \times t$. Such an alignment with minimum cost can be found by solving the following dynamic programming recurrences.

$$D(i,j) = \begin{cases} G(0,j) & \text{if } i = 0 \wedge j > 0, \\ \min \left\{ \begin{matrix} D(i-1,j), \\ G(i-1,j) + g_i \end{matrix} \right\} + g_e & \text{if } i > 0 \wedge j > 0. \end{cases} \qquad I(i,j) = \begin{cases} G(i,0) & \text{if } i > 0 \wedge j = 0, \\ \min \left\{ \begin{matrix} I(i,j-1), \\ G(i,j-1) + g_i \end{matrix} \right\} + g_e & \text{if } i > 0 \wedge j > 0. \end{cases}$$

$$G(i,j) = \begin{cases} 0 & \text{if } i = 0 \wedge j = 0, \\ g_i + g_e \times j & \text{if } i = 0 \wedge j > 0, \\ g_i + g_e \times i & \text{if } i > 0 \wedge j = 0, \\ \min \left\{ \begin{matrix} D(i,j), \\ I(i,j), \\ G(i-1,j-1) + s(x_i, y_j) \end{matrix} \right\} & \text{if } i > 0 \wedge j > 0. \end{cases}$$

The optimal alignment cost is given by $\min\{G(m,n), D(m,n), I(m,n)\}$.

In the rest of this task we will assume for simplicity that $m = n = 2^k$ for some integer $k \geq 0$. We will also assume that $\Sigma = \{A, C, G, T\}$, $g_i = 2$, $g_e = 1$, and $s(x_i, y_j) = 1$ if $x_i \neq y_j$ and 0 otherwise.

(*a*) [**25 Points**] The entries of D, I and G in any given row (say, row i) depend only on entries in the previous row (i.e., row $i-1$). So one can easily compute all entries of those three 2D arrays row by row using only two arrays of length n each. However, since each entry of I and G also depends on entries in the previous column (that is, column j depends on column $j-1$), a parallel implementation of this row-by-row approach is not very straightforward. But as shown below it is not that difficult either.

Suppose we have already computed all D, I, G entries of row $i-1$. Since D values in row i depend only on D and G values in row $i-1$, we can compute all $D(i,j)$ entries in parallel. However, I values in row i depend on I and G values in the same row. But we can compute these $I(i,j)$ values efficiently in parallel (described below). Once we have all I and D values in row i, we can compute all $G(i,j)$ values in parallel.

In order to compute the $I(i,j)$ values we assume $g_i \geq 0$, and define the following variables for each cell $j \in [1,n]$ of row i.

$$u(j) \quad = \quad \min\{D(i,j), G(i-1,j-1) + s(x_i, y_j)\},$$

$$\text{and } v(j) \quad = \quad u(j) + g_i - j \times g_e.$$

$$\text{Then } G(i,j) \quad = \quad \min\{u(j), I(i,j)\}.$$

$$\begin{aligned}
\text{Now let } s(j) &= I(i,j+1) - (j+1)\times g_e \\
&= \min\{I(i,j)+g_e, G(i,j)+g_i+g_e\} - (j+1)\times g_e \\
&= \min\{I(i,j), G(i,j)+g_i\} - j\times g_e \\
&= \min\{I(i,j), u(j)+g_i, I(i,j)+g_i\} - j\times g_e
\end{aligned}$$

$$\begin{aligned}
\text{Assuming } g_i \geq 0,\ s(j) &= \min\{I(i,j), u(j)+g_i\} - j\times g_e \\
&= \min\{I(i,j) - j\times g_e, u(j)+g_i - j\times g_e\} \\
&= \min\{s(j-1), v(j)\} \\
&= \min_{1\leq k\leq j}\{v(k)\}
\end{aligned}$$

Thus the s array stores the prefix mins of the values in the v array.

The $I(i,j)$ values can now be derived from the $s(j)$ values as follows.

$$I(i,j+1) = s(j) + (j+1)\times g_e, \text{ when } 1 \leq j < n,$$

$$\text{and } I(i,1) = \min\left(I(i,0), G(i,0)+g_i\right) + g_e.$$

Implement this parallel row-by-row algorithm in CUDA using only $\Theta(n)$ space. You only need to compute the optimal alignment cost (computing the aligned sequences is not required).

(b) [**25 Points**] The entries of D, I and G can also be computed diagonal by diagonal, that is, in step $k \in [0, 2n]$, we compute all $D(i,j)$, $I(i,j)$ and $G(i,j)$ entries with $i+j=k$. It turns out that all entries in the same diagonal can be computed in parallel. Implement this parallel diagonal-by-diagonal algorithm in CUDA using only $\Theta(n)$ space. Again you only need to compute the optimal alignment cost, and not the aligned sequences.

(c) [**25 Points**] For each test input (see Appendix 1) report the optimal alignment cost, CPU running time of the serial row-by-row implementation, GPU running time of the parallel row-by-row implementation, and the GPU running time of the parallel diagonal-by-diagonal implementation. The running time must include the time needed for transferring data between host and device memories.

Task 3. [Optional, No Points] Floyd-Warshall's All-Pairs Shortest Paths.

(a) [**No Points**] Implement the parallel Floyd-Warshall's APSP algorithm given in the following Figure (PAR-FW-APSP) in CUDA. Report speedup w.r.t. the serial CPU implementation for reasonably large input sizes (e.g., $n \approx 2000$).

```
PAR-FW-APSP( $c$, 1, $n$ )
(The input $c[1 \ldots n, 1 \ldots n]$ is an $n \times n$ matrix with all diagonal entries set to zero and the remaining entries set to positive values.)
for $k \leftarrow 1$ to $n$ do
    parallel for $i \leftarrow 1$ to $n$ do
        parallel for $j \leftarrow 1$ to $n$ do
            $c[i,j] \leftarrow \min\{c[i,j], c[i,k] + c[k,j]\}$
```

APPENDIX 1: Input/Output Format for Task 2

Your code must read from standard input and write to standard output.

- **Input Format:** The first line of the input will contain a single integer giving the value of n (guaranteed to be a power of 2). Each of the next two lines will contain a sequence (strings of A, C, G, T) of length at least n. You must read only the first n symbols of each sequence.
- **Output Format:** The output will contain an integer giving the optimal alignment cost.
- **Test Input/Output:** Please check the folder `/work/01905/rezaul/CSE590/HW3/tests` on Lonestar.

APPENDIX 2: What to Turn in

Please email one compressed archive file (e.g., zip, tar.gz) containing the following items to `cse590hw@cs.stonybrook.edu`.

- Source code, makefiles and any other scripts required for running your code.
- A PDF document containing all answers.

APPENDIX 3: Things to Remember

- You can use Amazon Web Services (AWS) for your homework. You can access your account from the following page: `https://cse590.signin.aws.amazon.com/console`.
- The following is a tutorial prepared by our grader (Gaurav Menghani) on how to use GPU's on AWS: `http://www.gaurav.im/files/aws-cuda/aws-cuda.html`.
- Please remember to terminate any instances on AWS that you no longer need. It turns out that the AWS credits burn pretty fast! So we need to be very careful with our usage. Perhaps we should first debug our code on our local machine or on XSEDE (e.g., you can run GPU jobs on Lonestar), and then run experiments on AWS.
- Please start working on the homework as early as possible so that you can identify any difficulties in using AWS and CUDA as soon as possible, and solve them in time.