ME759
High Performance Computing for Engineering Applications
Assignment 9
Parallel Prefix Scan
Date Assigned: November 4, 2013
Date Due: November 11, 2013 – 11:59 PM

The goals of this assignment are as follows:

- Understand the advantages of using, when available, libraries such as **`thrust`**
- Revisit your old code to gauge the performance improvement that you gain when using **`thrust`**
- Understand additional support available in **`thrust`** that we didn't get a chance to cover in class
- Understand the benefit of fusing operations in **`thrust`** (Bonus problem)

Problem 1. For this problem you'll have to revisit your old code for parallel reduction. Perform a scaling analysis of both your code and a solution provided by **`thrust`** (note that in some cases there are several ways for accomplishing the same outcome in **`thrust`**, simply pick one that you think scales well). To this end, for a collection of N entries in an array (vector), sum up these random numbers and report in a plot the amount of time required to carry out this operation. If you prefer not to use your code, the TA will provide a "default" solution for the parallel reduce operation. That being said, I strongly encourage you to use your implementation, it doesn't matter how fast/slow it is.
For this problem, start with $N = 2$ and go up at least to $N = 50,000,000$. In addition to a brief report,

- Post on the forum the largest value of N with which you could run your code
- Post the **`png`** plot that illustrates the scaling behavior of the two solutions

Format your forum post like this:

```
Problem 1:
Max N Reduction: blah
Time required to reduce Max N: blahblah
```

Problem 2. Just like **Problem 1**, except that this time around you will have to consider the prefix scan implementation you produced. In case you prefer not to work with your implementation, a default implementation will be provided by the TA. That being said, I strongly encourage you to use your implementation, it doesn't matter how fast/slow it is.

Problem 3. Consider the data provided in class on the "Example: Processing Rainfall Data" slide. Below it has been augmented with a couple of extra columns. Write a short piece of code that computes

a) The number of days for which the rainfall exceeded 5. Hint: You might use `count_if` and a placeholder.
b) Total rainfall at each site. Hint: You might use `sort_by_key` and `reduce_by_key`.

```
day         [0  0  1  2  5  5  6  6  7  8  9  9  9 10 11]
site        [2  3  0  1  1  2  0  1  2  1  3  4  0  1  2]
measurement [9  5  6  3  3  8  2  6  5 10  9 11  8  4  1]
```

When running the executable called **`problem3.exe`**, the TA should get the two answers or relevance.

Problem 4 [BONUS PORBLEM]. Consider this **thrust** forum posting of last year: http://groups.google.com/group/thrust-users/browse_thread/thread/8eef4bf8b1606f9c. Your job is to fix Shankar's original problem. In his code, there is a matrix vector multiplication that we'll avoid since it is not relevant and would bring into the picture another CUDA library called **cusp** (useful for sparse linear algebra computation). To this end, assume that his matrix $\mathbf{A}$ is a diagonal matrix with a constant diagonal value α; i.e., $\mathbf{A}=\alpha\mathbf{I}_{n\times n}$ and as such, the operation $\mathbf{y}=\mathbf{A}\cdot\mathbf{x}$ becomes $\mathbf{y}=\alpha\mathbf{x}$. Specifically, pretend that you have to implement the following sequence of operations: $\mathbf{y}=\alpha\mathbf{x}$, $\mathbf{x}=f(\mathbf{y},\mathbf{p})$, and $a=\text{reduce}(\mathbf{x})$. As indicated in the forum posting, this set of operations is carried out a number of **iterCount** times, most likely as part of an iterative algorithm. Based on limited information, let's assume that the algorithm that Shankar wants to implement is like this:

1 for k=0...interCount

2 $\quad\mathbf{y}^{(k+1)}=\dfrac{n}{a^{(k)}}\mathbf{x}^{(k)}$

3 $\quad i=1...n:\quad x^{(k+1)}[i]=\sqrt{y[i]^{(k+1)}\cdot y[i]^{(k+1)}+p[i]\cdot p[i]}$ //defines function $f(\mathbf{y},\mathbf{p})$

4 $\quad a^{(k+1)}=\text{reduce}(\mathbf{x}^{(k+1)})$

5 endfor

In our case, $n=20,000,000$, $a^{(0)}=n$, **iterCount**=20, and both $\mathbf{x}^{(0)}$ and $\mathbf{p}$ are arrays of random numbers between 0 and 1. For this problem, your code should handle the math in **double precision**.

a) Report on the forum the value of $a^{(21)}$ i.e., the value of a at the end of the algorithm
b) Determine, using the right CUDA timing functions, the amount of time it takes to run the approach originally implemented by Shankar using **thrust**; i.e., no operation fusion
c) Implement the solution suggested by Teemu and determine the amount of time required by this implementation
d) Do you get any benefit from implementing the solution suggested by Mark Harris?
e) Can you think of any way to fuse the operations required by the algorithm above beyond the suggestion made by Teemu? If so, describe it and provide the time it takes to implement the algorithm above.

When reporting your findings on the forum, please format your posting like this:

```
Assignment 9 - Problem 4 [YourFirstLastNameHere]:
a_21 = blah (provide at least 8 significant digits)
t_Shankar = blahblahblahblah
t_Teemu = blahblahblah
t_Mark = blahblah [provide if you've got results]
t_YourFirstNameHere = blah [provide if you've got results]
```

NOTE: If you produce a solution that is faster than what is currently listed on the "Assignment Champion[s]" discussion thread please edit the thread and post your results. Your claim to fame will be subsequently validated by the TA :-). This problem, when solved in MATLAB 7.10.0.499 (R2010a) on my Windows 7 desktop, Intel Xeon X5650@2.67GHz w/ 12 GB RAM, took about 8.1 seconds to finish. The MATLAB code is provided on the next page.

```
n = 20000000
a = n;
x = rand(1,n,'double')' ;
p = rand(1,n,'double')' ;

tstart = tic;
for k=0:20
     y = n/a*x;
     x = sqrt(y.*y + p.*p);
     a = norm(x,1);
end
telapsed = toc(tstart)
```