CS 385 Lab 6 – Wed., Mar. 11, 2009

Summary

In this lab, you will instrument your parallel for-loop to figure out how TBB partitions tasks. Then you will get some practice with concurrent containers.

Deliverables:
yourlastnameL6a.cpp
yourlastnameL6b.cpp

You should submit *yourlastname*L6a.cpp and *yourlastname*L6b.cpp to ~srivoire/cs385/submit. If you are successful, you will see your file listed at http://rivoire.cs.sonoma.edu/cs385/lab6sub.txt

Preparation

If TBB isn't working for you, you may need to type the following command:

```
source ~/.bash_profile
```

Remember to compile your programs using the –ltbb flag (lowercase L):

```
g++ -ltbb [source] -o [output]
```

Parallel For

Copy your parallel-for code from Lab 5a into a new file.

Add code to the () operator to print out the beginning and end indices of each range on its own line, like this:

[0, 12)

That's a left-hand square bracket and a right-hand angle bracket, since this particular range starts at 0 but stops before 12.

Set N to a large number and use the auto-partitioner (see Lab 5) instead of manually adjusting GRAINSIZE.

In an initial comment, answer the following:

- How many chunks did the initial array get divided into?
- What are the first 5 lines and the last 5 lines of the chunk printouts?

Concurrent Containers

In this part of the lab, you will use the concurrent_vector container and the parallel_sort function to accomplish a familiar task:

Your job is to parallelize a program that scans, character by character, through an array of input text and does the following:

- If the ith character is 'q' or 'Q', it sets the next 16 characters to 0 and continues the loop with the (i + 16)th character.
- Otherwise, it copies the character into the output buffer, unmodified.

The steps you should follow are:

- Generate a random array of text (of size N + 15, in order to keep us from running off the end of the array)
- Using a parallel_for loop, add the locations of all the 'q's to a concurrent_vector
- Use the parallel_sort function on the concurrent vector
- Sequentially, determine which q's are the start of 0 blocks and which ones will be canceled out. Add the q's that will begin 0 blocks to a new concurrent vector.
- Using a parallel_for loop, loop over this vector and zero out the 'q' at each location and the next 15 characters.

Documentation on TBB concurrent containers can be found online at:
http://www.threadingbuildingblocks.org/wiki/index.php?title=Concurrent_Vector

See this article for the usage of parallel sort:
http://www.devx.com/SpecialReports/Article/40879/1763/page/4

Use the TBB timing functions to get the running time of the sequential code and of your parallelized code:

```
...
#include "tbb/tick_count.h"
using namespace tbb;
...

tick_count t0, t1;

t0 = tick_count::now();
// Code to be timed goes here
t1 = tick_count::now();

printf("Took %3.5f seconds.\n", (t1-t0).seconds());
…
```

[Thanks to Joe Muller for this code!]

Take 3 measurements for both the sequential and the parallel code, and report each of these measurements and their average in your initial comment.