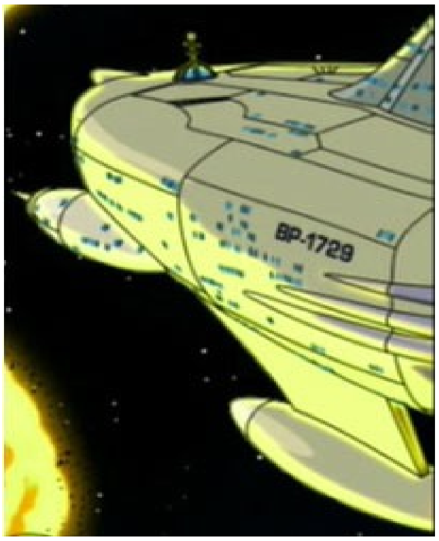
CS 302 – Assignment #6

Purpose: Learn concepts regarding threading and data races
Due: Monday (10/15) → Must be submitted on-line before class.
Points: Part A → 50 pts, Part B → 25 pts

Assignment:

Part A:



In recreational number theory, a Harshad number (or Niven number)[1] is an integer that is evenly divisible by the sum of its digits. For example, given the integer 1729[2], the sum of its digits is 19 (1+7+2+9) and 1729/19 = 91.

Write a threaded C++11 program to find the count and provide list of Harshad numbers between 1 and a user provided limit (inclusive). Since the program will be threaded, the Harshad numbers must be stored and printed at the end. For example, between 1 and 100 there are exactly 33 Harshad numbers[3].

```
Harshad Numbers
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 18, 20, 21, 24, 27, 30, 36, 40, 42,
45, 48, 50, 54, 60, 63, 70, 72, 80, 81, 84, 90, 100
```

In order to improve performance, the program should use threads to perform computations in parallel. The program should read the thread count and limit from the command line in the following format:

```
    ./harshadNums -t <threadCount> -l <limitValue>
```

The command line argument processing must be in a dedicated function. The thread count should be between MIN_THREAD_COUNT and MAX_THREAD_COUNT (inclusive) and the limit value should be between MIN_LIMIT and MAX_LIMIT (inclusive). As such, you will need to use **`unsigned long long`** as the data type. You will need to create your own main source file and makefile. Refer to the example execution for output formatting. Global variables will be required for the counts array and limit value.

Additionally, the program should use the C++11 high resolution clock to provide an execution time in milliseconds. Refer to the following pages for a detailed explanation of how to use the C++ high resolution clock.

1 For more information, refer to: https://en.wikipedia.org/wiki/Harshad_number
2 For more information, refer to: https://en.wikipedia.org/wiki/1729_(number)
3 For more information, refer to: https://oeis.org/A005349

Each thread should check HARSHAD_STEP number of values for Harshad numbers. Each thread should place its count into a harshadCounts[] array based on its thread number which is set and passed from the main.

```
static      const unsigned long long      MIN_THREAD_COUNT = 1;
static      const unsigned long long      MAX_THREAD_COUNT = 64;
static      const unsigned long long      MIN_LIMIT = 10;
static      const unsigned long long      MAX_LIMIT = 10000000000;
static      const unsigned long long      HARSHAD_STEP = 10000;
```

Part B:

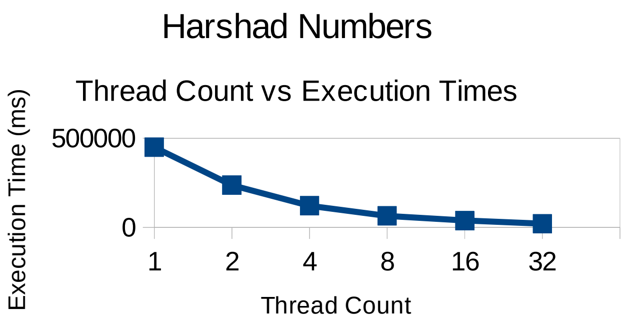
When completed, use the provided timing script, **`ast6Exec`**, to execute the program various times with single and multiple threads (>30 minutes). The script writes the results to a file (a6times.txt). To ensure consistent results, use the **`cssmp.cs.unlv.edu`** which has 32 cores (uses CS login). Enter the thread counts and times into a spreadsheet and create a line chart plot of the execution times versus the thread counts. Refer to the example below for how the plot should look.

Note, the default g++ compiler version on CSSMP does not support the C++11 standards. In order to use the current C++11 standard, you will need to type the following (once per session):

```
scl enable devtoolset-2 bash
```

Create and submit a brief write-up (PDF format), not to exceed ~500 words, including the following:

- Name, Assignment, Section.
- Include a copy of the timing script output
- Include a copy of the chart (see simplified example to right).
- Summarize the results from the timing script.
- Determine the percentage speed-up[4] using the below formula (for each thread count > 2):



$$speedUp = \frac{Time_{single}}{Time_{new}}$$

- Remove the mutex locks and execute the program. Report the results. Include a description of what happened and an explanation of why.

Submission:

When complete, submit:

- Part A → A copy of the **source files** via the class web page (assignment submission link) by class time on the due date. The source files, with an appropriate *makefile*, should be placed in a ZIP folder. The grader will download, uncompress, and type **make** (so you must have a valid, working *makefile*).
- Part B → A copy of the write-up including the chart (see example). Must use PDF format.

4 For more information, refer to: https://en.wikipedia.org/wiki/Speedup

Assignments received after the due date/time will not be accepted. Make sure your program includes the appropriate documentation. See Program Evaluation Criteria for CS 302. ***Note, points will be deducted for especially poor style or inefficient coding.***

Threads

C++11 introduced a new thread library. This library includes utilities for starting and managing threads. It also contains utilities for synchronization like mutexes and other locks, atomic variables and other concurrency related functions.

When you create an instance of a **std::thread**, it will automatically be started. When you create a thread you have to give it the code by passing the function as a pointer. For example, the following is a very common (but not useful) threaded Hello World program:

```
#include <iostream>
#include <thread>

using namespace std;

void helloThdFunc(int a1)
{
    cout << "Hello from thread: " << a1 << endl;
}

int main(){
    thread thd1(helloThdFunc, arg1);      // start thread, pass arg
    thd1.join();                          // wait for thread to finish

    return 0;
}
```

As shown, the ***join()*** is used to wait for the thread to complete. You must wait for all threads to complete before terminating the program. You can dynamically create an array of threads in order to effectively handle a varying number of threads.

In addition, mutex's must be used to avoid race data races. To use a mutex, first declare a mutex variable (global in this example) and use the mutex variable as shown in order to avoid race conditions.

```
mutex   myMutex;

void someThreadFunc(){
    lock_guard<mutex> guard(myMutex);
    // perform applicable stuff here...
}
```

The `lock_guard<mutex> guard(mutexVariable)` operation will ensure that the subsequent statements are locked and thus only executed by one thread at a time. Specifically, if the mutex is free, it will be locked and the the subsequent code will executed. If the mutex is already locked, all other threads will be blocked from executing the code until one obtains the lock. The lock will be released when it goes out of scope (in this example, at the end of the function).

To compile the, include the `-std=c++11` option to get the C++11 support activated. Additionally include the `-pthread` option for the threading libraries. You will need to `#include <thread>` and `#include <mutex>` as part of the program includes.

C++11 High Resolution Timer

The functions for the C++11 high resolution timer are located in the chrono library.

```
// get start timer...
auto t1 = chrono::high_resolution_clock::now();

// do stuff...

// get end time...
auto t2 = chrono::high_resolution_clock::now();

// show results with times
cout << "Program took: " <<
      << std::chrono::duration_cast<std::chrono::milliseconds>
            (t2 – t1).count() << " milliseconds" << endl;
```

You will need to **`#include <chrono>`** as part of the program includes.

Hardware Core Count

The hardware count can be obtained as follows:

```
unsigned long hwthd = thread::hardware_concurrency();
```

Example Execution

Below are some example executions for reference.

```
ed-vm%
-----------------------------------------------------------------
CS 302 - Assignment #7

Hardware Cores: 8
Thread Count: 1
Numbers Limit: 100

Results:
  Count of Harshad numbers between 1 and 100 is 33

Threads took: 0 milliseconds
ed-vm%
ed-vm%
ed-vm% ./harshad
Usage: ./harshad -t <thereadCount> -l <limit>
ed-vm%
ed-vm% ./harshad -t 90 -l 100
Error, invalid thread count.
ed-vm%
ed-vm% ./harshad -t 1 -l 7
Error, invalid number limit (7).
ed-vm%
ed-vm%
```

```
ed-vm%
ed-vm% ./harshad -t 4 -l 1000
-----------------------------------------------------------------
CS 302 - Assignment #7

Hardware Cores: 8
Thread Count: 4
Numbers Limit: 1000

Results:
  Count of Harshad numbers between 1 and 1000 is 213

Threads took: 0 milliseconds
ed-vm%
ed-vm%
ed-vm% ./harshad -t 4 -l 1000000
-----------------------------------------------------------------
CS 302 - Assignment #7

Hardware Cores: 8
Thread Count: 4
Numbers Limit: 1000000

Results:
  Count of Harshad numbers between 1 and 1000000 is 95428

Threads took: 13 milliseconds
ed-vm%
ed-vm%
ed-vm% ./harshad -t 4 -l 1000000000
-----------------------------------------------------------------
CS 302 - Assignment #7

Hardware Cores: 8
Thread Count: 4
Numbers Limit: 1000000000

Results:
  Count of Harshad numbers between 1 and 1000000000 is 61574510

Threads took: 8244 milliseconds
ed-vm%
ed-vm%
ed-vm% ./harshad -t 4 -l 5000000000
-----------------------------------------------------------------
CS 302 - Assignment #7

Hardware Cores: 8
Thread Count: 4
Numbers Limit: 5000000000

Results:
  Count of Harshad numbers between 1 and 5000000000 is 292460332

Threads took: 44392 milliseconds
ed-vm%
ed-vm%
```