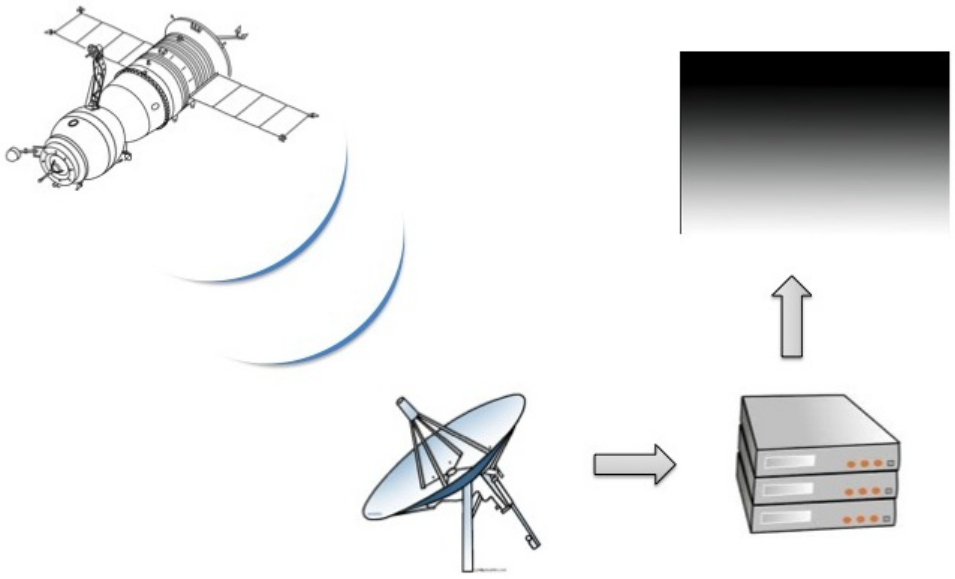
Project #5: Hubble Simulator

CMSC 341 - Spring 2014

Due date: May 13^{th}



1 Introduction

Satellite telescopes gather a huge amount of data. Often satellites store their data in solid state memory and transfer it to Earth in batches. To prevent missing information, while satellites transfer data, the telescope continues to collect information. Once the information is received, the data is process, sorted, and analyzed.

In this project students will be implementing a Hubble simulator that will (1) collect data, (2) transfer the data in batches, (3) sort and process the data stored in shared buffers, and (4) display the information for further analysis and verification.

2 Objectives

The purpose of this project is to introduce the basic concepts of multi-threads, shared memory, synchronized methods, thread-safe operations, Java Fork/Join, and visual illustrations of data.

3 Primary Requirements

Four concurrent threads will be in charge of *collecting, storing, receiving*, and *processing* the data.

1. **Data collection:** the satellite thread will be generating and adding elements into a shared, thread-safe buffer B_1. In this particular project, the satellite thread will produce random integers between $0 - 4096$. If there's no space on the shared buffer B_1, the satellite thread should wait.

2. **Shared Buffer:** the buffer thread creates and manages a thread-safe array B_1 of size $N^2 * 2$, where N is a variable provided to each class. For this particular project $N = 2^i$ for $8 <= i <= 11$, thus there will be only four possible values for $N \in \{2^8 = 256, 2^9 = 512, 2^{10} = 1024, 2^{11} = 2048\}$.

3. **Receiver:** the receiver thread will try to obtain data from the satellite through the shared buffer B_1. However, the thread will have to wait until there are at least N^2 elements in the shared buffer. The buffer B_1 will notify the receiver when enough data becomes available and the receiver thread will then move the data into a different shared buffer B_2 of size N^2. Once the data is transferred, the information should be removed from the satellite's buffer B_1.

4. **Processing:** the processing thread will (a) sort the elements, (b) normalize the data, and (c) save the information into an image.

   (a) The processing thread will first sort the array using a *Fork/Join* implementation of *Mergesort*. The Mergesort algorithm will receive a parameter T that defines the threshold for sorting the data. If the number of elements in a particular process are less than T, *insertion sort* should be used to sort the elements, otherwise the data will be split and a *ForkingAction* or *RecursiveAction* will be called. The Mergesort implementation doesn't have to be a generic class, instead it can be a similar implementation to the one discussed in class (see slides on Blackboard, slide #39, file called *Parallel Programing - 2 - Caban*). The time t_{sec} that it takes to sort the elements should be computed. For this particular project, $T = 10^j$ where $j \in \{1, 2, 3, 4, 5\}$.

   (b) Once the elements in buffer B_2 have been sorted, the elements should be normalized between -128 and 127 and transferred into a byte array. A very useful tip to

know in programming languages is how to rescale an array with values between $[min, mix]$ to another with values between $[a, b]$. There are multiple ways to do that, one of the most popular techniques is to used the following formula:

$$f(x) = \frac{(b-a)(x-min)}{max-min} + a \tag{1}$$

(c) Once the byte array has been created, the data should be saved as a grayscale image. See documentation for the following classes:

```
BufferedImage
ImageIO
ByteArrayInputStream
```

You don't have to use any of these classes. There are many ways to do this, just pick one that works for you.

(d) Note: B_1 should only be created once for every possible N. You should only remove N^2 elements from B_1, process the data, and save the resulting image. Then remove another N^2 elements from B_1, process the data with a larger T and save the image. Repeat this for any possible T.

4 Other requirements and hints

1. **Producer / Consumer:** For the *satellite, buffer*, and *receiver* threads, students should thinking about the producer / consumer examples discussed in class (see "Threads - Coded Examples" on Blackboard). Specifically study the *3.Polling, 4.Wait*, and *5.WaitAgain* programs that are under the same zip file in blackboard or here (http://userpages.umbc.edu/ slupoli/notes/DataStructures/code/threads/).

2. **Number of runs:** the program will be executed 20 times. That is, four different N values ($8 <= i <= 11$) times five different thresholds T ($1 <= j <= 5$). The driver (i.e. main function) should have a loop to execute the program 20 times. Since we are using threads, make sure that all threads are completed before executing the next iteration of the loop.

3. The ant project should create an *images* folder where the images will be stored.

4. You must use Java 7. The Fork-Join framework is new to Java 7.

5. For additional information about *Fork-Join* and pseudocode for *Merge Sort*, please see slides $34 - 39$ in Blackboard. (Click on file called *Parallel Programing - 2 - Caban*). The easiest way to implement this is following the pseudocode shown in slide 39.

6. **Number of cores:** To get the number of processors (cores) and amount of free memory students should read more about *Runtime.getRuntime().availableProcessors()* and *Runtime.getRuntime().freeMemory()*

7. **Output:**

```
Available processors (cores): 8
Available memory (bytes): 123081744

Run #1: i=8,j=1,N=256,B1=131072,B2=65536,T=10
Time mergesort: 108ms
Saving image: images/output_N256_T10.jpg

Run #2: i=8,j=2,N=256,B1=131072,B2=65536,T=100
Time mergesort: 54ms
Saving image: images/output_N256_T100.jpg

Run #3: i=8,j=3,N=256,B1=131072,B2=65536,T=1000
Time mergesort: 61ms
Saving image: images/output_N256_T1000.jpg

Run #4: i=8,j=4,N=256,B1=131072,B2=65536,T=10000
Time mergesort: 41ms
Saving image: images/output_N256_T10000.jpg

Run #5: i=8,j=5,N=256,B1=131072,B2=65536,T=100000
Time mergesort: 45ms
Saving image: images/output_N256_T100000.jpg

Run #6: i=9,j=1,N=512,B1=524288,B2=262144,T=10
Time mergesort: 98ms
Saving image: images/output_N512_T10.jpg

Run #7: i=9,j=2,N=512,B1=524288,B2=262144,T=100
Time mergesort: 73ms
Saving image: images/output_N512_T100.jpg

Run #8: i=9,j=3,N=512,B1=524288,B2=262144,T=1000
Time mergesort: 84ms
Saving image: images/output_N512_T1000.jpg

Run #9: i=9,j=4,N=512,B1=524288,B2=262144,T=10000
Time mergesort: 113ms
Saving image: images/output_N512_T10000.jpg

Run #10: i=9,j=5,N=512,B1=524288,B2=262144,T=100000
Time mergesort: 68ms
```

```
Saving image: images/output_N512_T100000.jpg

Run #11: i=10,j=1,N=1024,B1=2097152,B2=1048576,T=10
Time mergesort: 421ms
Saving image: images/output_N1024_T10.jpg

Run #12: i=10,j=2,N=1024,B1=2097152,B2=1048576,T=100
Time mergesort: 1217ms
Saving image: images/output_N1024_T100.jpg

Run #13: i=10,j=3,N=1024,B1=2097152,B2=1048576,T=1000
Time mergesort: 386ms
Saving image: images/output_N1024_T1000.jpg

Run #14: i=10,j=4,N=1024,B1=2097152,B2=1048576,T=10000
Time mergesort: 364ms
Saving image: images/output_N1024_T10000.jpg

Run #15: i=10,j=5,N=1024,B1=2097152,B2=1048576,T=100000
Time mergesort: 403ms
Saving image: images/output_N1024_T100000.jpg

Run #16: i=11,j=1,N=2048,B1=8388608,B2=4194304,T=10
Time mergesort: 5201ms
Saving image: images/output_N2048_T10.jpg

Run #17: i=11,j=2,N=2048,B1=8388608,B2=4194304,T=100
Time mergesort: 1739ms
Saving image: images/output_N2048_T100.jpg

Run #18: i=11,j=3,N=2048,B1=8388608,B2=4194304,T=1000
Time mergesort: 1701ms
Saving image: images/output_N2048_T1000.jpg

Run #19: i=11,j=4,N=2048,B1=8388608,B2=4194304,T=10000
Time mergesort: 1549ms
Saving image: images/output_N2048_T10000.jpg

Run #20: i=11,j=5,N=2048,B1=8388608,B2=4194304,T=100000
Time mergesort: 1595ms
Saving image: images/output_N2048_T100000.jpg
```

8. The implementation for *satellite, buffer, receiver*, and *mergesort* should be stored in individual files. See producer / consumer examples discussed in class.

9. Please take a look at the code posted on Blackboard. Specifically study the *3.Polling, 4.Wait*, and *5.WaitAgain* programs that are under the same zip file in blackboard or here (http://userpages.umbc.edu/ slupoli/notes/DataStructures/code/threads/). Your *main* function should follow this general concept. You can send any parameters that you would like to any of the classes or methods.

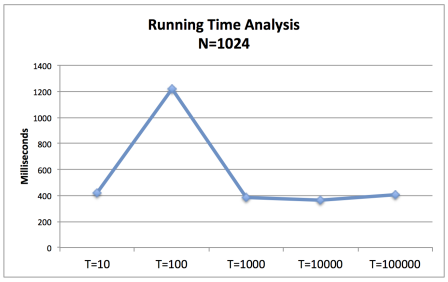
```
for N in {256,512,1024,2048}:
   B1 = Buffer(N); //The actual size of buffer will be 2*N^2
   P = Producer(B1, N); //Thread that will add data to B1

   for T in {10,100,1000,10000,100000}:
       //Thread that will consume data from B1 (only N^2 elements)
       B2 = Consume(B1,N);

       //Sort using Merge Sort
       t1 = GetTime();
       Merge(B2, T);
       t2 = GetTime();

       //Process data
       BArray = Normalize(B2);
       Save_Byte_to_Image(BArray);
       Print_info(t2-t1);
```

5 What to submit

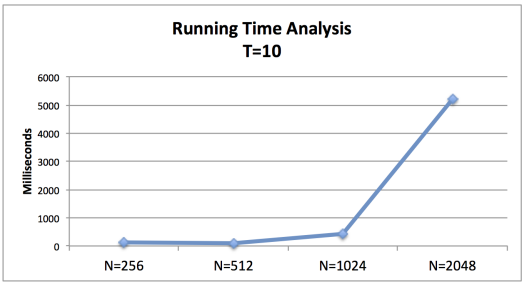
1. Source code in a package named *project5*.

2. 9 line plots illustrating the running time for mergesort. (1) Note, you don't have to do any implementation for this step. Instead run your program, copy/paste the results into your favorite plotting software application, and generate the plots. You can use MS Excel, MatplotLib, or any other preferred software. (2) The images should be included in a directory outside the package *project5* called *results*. (3) The images should either be .PNG, .JPG, or .TIF formats.

   (a) For a each $N \in \{2^8 = 256, 2^9 = 512, 2^{10} = 1024, 2^{11} = 2048\}$, create a line plot illustrating the time taken to sort the elements using the five different thresholds T. This will result in four plots. Please make sure to have a title and labels in your plots.



(a)



(b)

Figure 1: (a) Line plot illustrating the time taken to sort the elements using the five different thresholds T with 1024^2 elements. (b) Line plot illustrating the time taken to sort the elements using a treshold $T = 10$ for four different N values.

```
running_time_analysis_N256.jpg
running_time_analysis_N512.jpg
running_time_analysis_N1024.jpg
running_time_analysis_N2048.jpg
```

(b) For each $T \in \{10^1, \ldots, 10^5\}$, create a line plot illustrating the time taken to sort the elements using different N. This will result in five plots. Please make sure to include a title and labels in your plots.

```
running_time_analysis_T10.jpg
running_time_analysis_T100.jpg
running_time_analysis_T1000.jpg
running_time_analysis_T10000.jpg
running_time_analysis_T100000.jpg
```

3. For your submission, the *images* folder should not have any images. That directory will be created by *ant* when we compile your code and the images (e.g. images/output_N256_T10.jpg) will be generated when we run your program. The only images that you will submit are those inside the *results* folder showing the running time analysis of your application.