A Comparison of Job Completion Times Using Three Different Batch Processing Implementations

Lee R. Clendenning
Berry College
lclendenning@berry.edu

Abstract

This study is an exploration and comparison of the gross completion times for three different schemes to analyze a batch of bitmapped image files. The project originated from a separate research effort to visually evaluate the accuracy of student's freehand drawings. Using linear algebraic functions and standard computer image analysis techniques, it was possible to evaluate the "neatness" or "messiness" of such drawings on a ratio-level scale using the variation of sampled pixel location errors from ideal representations. Desired output was a simple database table on disk with records of task performance scores for each subject. The three schemes explored were (1) traditional object-oriented linear programming; (2) concurrent thread processing using a master Java class, and creating and activating a separate thread to analyze all tasks on each subject's file; and (3) parallel processing using a master computer, distributing the image file analysis tasks to slaves in a multiple computer grid.

During the developmental stages of each scheme's implementation, considerable debugging through screen reporting confused the efficiency issues. Such communication between the master and the slaves in the grid scheme actually increased overall batch times as more slaves were added. Under those constraints, the threaded scheme showed only a small improvement in batch processing times over the single linear program. However, removing all unnecessary screen reporting resulted in drastically improved performance of the grid as more slaves were added. Also under that condition, the concurrent threaded scheme was slightly faster than the traditional linear programming solution.

This study will be used as an example of processing choices in future undergraduate operating system classes. The procedures and comparison results should be of interest to other operating system instructors or individuals faced with image analysis problems.

Introduction

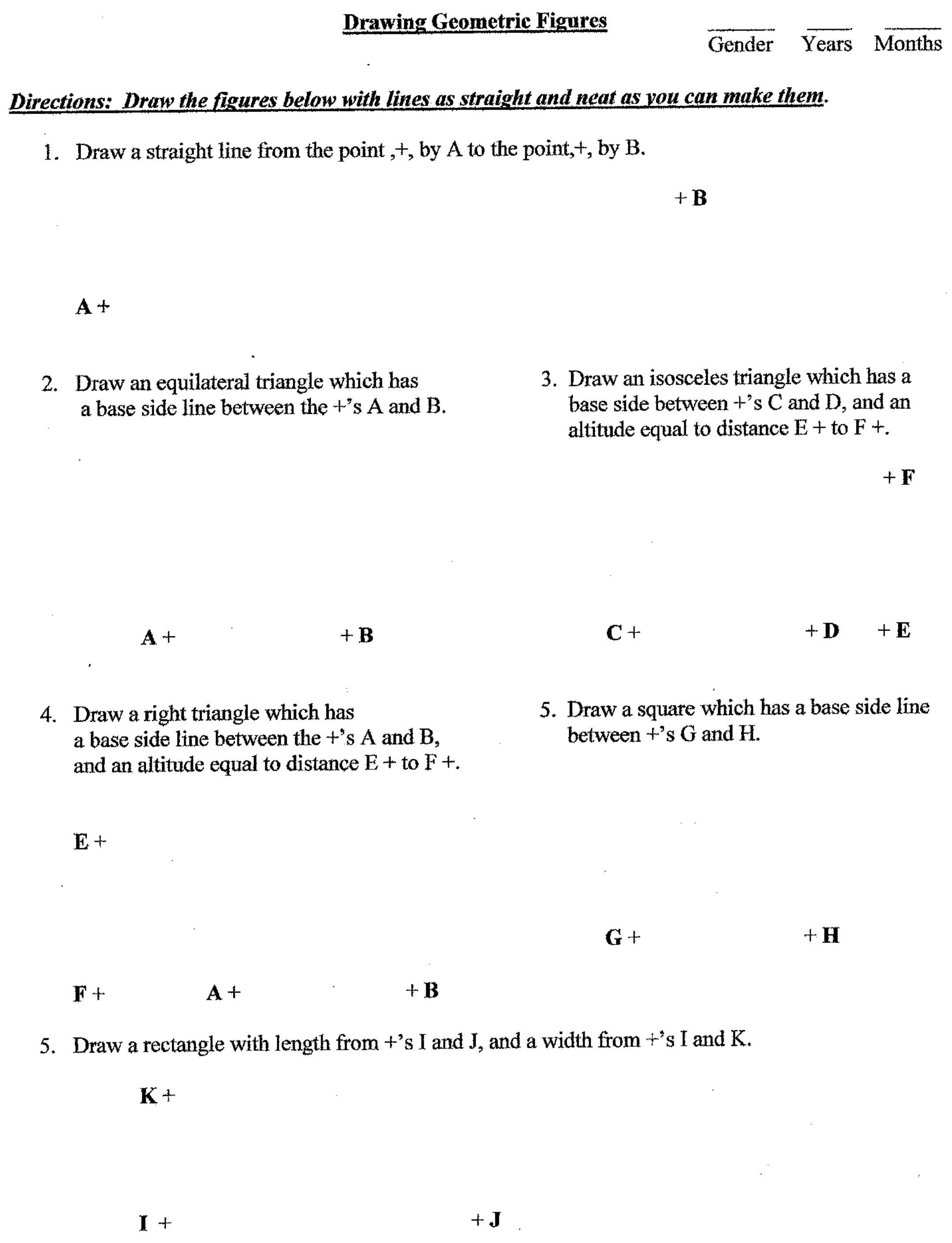
When teaching operating systems in undergraduate programs, it is sometimes difficult to identify realistic problems to illustrate and compare conventional linear programming with solutions using concurrent threads or systems of multiple processors. Often, students only feel competent in one particular language or with one operating system. Recently, the author was confronted with the real-world problem of analyzing a batch of bitmapped images. This problem was a prime opportunity for exploration, demonstrations, and comparisons of different solution schemes. Since students were comfortable with Java, this language was chosen for three modes of attack. The study evaluates total batch completion times for the three different schemes: (1) using one large linear Java class/batch program, calling on six processing members to read in the files and process them in sequence; (2) using a single master class with a main method creating and activating a synchronized Java thread for processing each input file; or (3) using a master/slave multiprocessor grid with the same program on each slave to process the files assigned to the slave from the batch. Desired output from each of the schemes was a simple database table on disk with a record analysis summary for each input file. Since the gross batch of jobs required a mixture of I/O and calculation bursts, choice of the best scheme was not obvious. The purpose of this study was to implement the three different schemes and compare the batch completion times.

Psychomotor issues of differences in performance between identified gender and age groups and other inter-relationships are not extensively reported as part of this study because they are not of paramount concern to computer scientists. The output database is in comma separated form easily accessible to those interested in further explorations.

Input Files and Required Output

The original study requiring the image analysis involved evaluating the psychomotor behavior of middle school students. Twenty six (26) male and 26 female students were asked to complete a series of six tasks drawing basic linear geometric forms: (1) a straight line between two points, (2) an equilateral triangle with given side, (3) an isosceles triangle with given base and altitude, (4) a right triangle with given base and altitude, (5) a square with given side, and (6) a rectangle with given length and width. These tasks were all presented in different areas of a single response sheet (see Figure 1). Each sheet was scanned into a separate Microsoft formatted, bitmapped file. The collection of response files became the batch to be processed. The input files were assigned names, such as “SheetDD,” where the D’s represented an unbroken sequence of identification numbers.

Using a combination of linear algorithms and image analysis techniques, it is possible to evaluate the “messiness” (or “negative neatness”) of the drawing tasks on a ratio scale. The operational definition of messiness is the sums of squared deviations of errors in position of sampled image pixels from ideal locations described by mathematical models [1]. All six tasks required drawing one or more straight lines in a specific location. Therefore, the tasks could be subdivided into analysis of the 18 required lines. The desired output was a database table with records for each subject, showing sample sizes and sums of squared errors for each task subpart. Since sums of squared errors can be meaningfully added from task to task, a total sum of squares for the subject’s entire sheet was also reported.



Drawing Geometric Figures

Gender	Years	Months

Directions: Draw the figures below with lines as straight and neat as you can make them.

1. Draw a straight line from the point ,+, by A to the point,+, by B.

+ **B**

A +

2. Draw an equilateral triangle which has a base side line between the +'s A and B.

A + +**B**

3. Draw an isosceles triangle which has a base side between +'s C and D, and an altitude equal to distance E + to F +.

+ **F**

C + +**D** +**E**

4. Draw a right triangle which has a base side line between the +'s A and B, and an altitude equal to distance E + to F +.

E +

F + **A** + +**B**

5. Draw a square which has a base side line between +'s G and H.

G + +**H**

5. Draw a rectangle with length from +'s I and J, and a width from +'s I and K.

K +

I + +**J**

Figure 1: Student Drawing Task Sheet (photo reduced)

The Traditional Linear Programming Scheme

A single Java class called Analysis1 was designed to read in the image files and process them one at a time. An output record for each processed input was sent to the disk database file. Although written in Java, this scheme used the ancient COBOL concept that whatever you do with one subject, you repeat the processing for the rest of the subjects. The main method is essentially a big for loop with counter, i, incrementing from 1 to the number of files in the batch (in this case, 56 files).

Reading Input Files

All input files were scanned as black and white packed pixel bitmaps, 736 pixels in width and 1,000 pixels in length. Forcing the width in pixels to be divisible by 32 (4 bytes) eliminated the problem of standard Microsoft software packing the end of pixel rows with “dummy” placeholders. This makes every bit in the file (beyond the header) a binary indicator of a black or white pixel (black = 0). Each packed pixel byte describes 8 pixels with the least significant bit addressing the right-most pixel in the group. Pixel bytes are stored from the far left bottom column, across the bottom row, and jumping to the left-most pixel of the row above. The left-most bottom pixel has coordinates of column X = 0, row Y =0 [2]. (Confusing the understanding of the file structure is the fact that Microsoft Paint reports pixel coordinates of the mouse cursor with X=0, Y=0, starting at the top left corner of the image.)

A Java character array of size 92,100 was large enough to accommodate the bytes of each entire image file and provided easy access to pixel data through the array index. A while loop read the input file one byte at a time, cast the byte as a char, and inserted it into the array. Each input file was opened only once and closed immediately after the entire char array was populated.

Processing Input Data

Eighteen separate member methods of the Analysis1 class were called in sequence to evaluate the assigned sub-lines. These functions use two nested for loops to scan the local pixel data across the assigned rows and columns which should contain the pixels of the line. The ideal locations of the sampled pixels were identified by a linear regression formula specific to that line. The sums of squared errors in locations for sampled pixels was calculated and saved in appropriately named member variables. When all 18 methods have populated the reporting variables, then a single disk out command wrote (appended) the report record to the output disk file.

The disk output file was opened and closed before and after each record was appended. This detracts from the efficiency of this scheme, but traditionally, leaving a disk file open just long enough to use it has been considered the safest procedure in terms of data integrity. This safety procedure was used because the college has a history of commercial power spikes and failures.

The Concurrent Thread Scheme

The concurrent thread scheme was implemented with three classes following published Java thread programming concepts [3, 4]. This scheme used a controlling/monitoring ScoreMaster class and two working thread classes. The ScoreMaster class' main method creates and starts one new object of the ScoringThread class (extends Thread) for every input sheet file to be processed. Each object of the ScoringThread class reads in its assigned image file, completes the image analysis using processing members essentially copied from the traditional linear programming scheme, and forms the corresponding reporting record. As they finish processing, they place each record in a static member string array, rather than repeatedly opening and closing a disk file. The ScoreMaster main thread also creates one new object of the RecordOutput class. This class' object prints to disk the processed records from the static string array of the ScoringRecord object. Therefore, this thread must be synchronized so that it waits for the processing thread to put valid information into the array. So, the ScoringThread objects were the producers; the array of string records was the buffer, while the sole consumer was the object of the RecordOutput class. The ScoreMaster class also timed the overall processing of the whole batch and reported that time to the screen.

The Multiprocessor Grid Scheme

The multiprocessor grid scheme used a master computer networked to additional slaves. Both the master computer and the slaves have the capability of booting in Linux or Windows XP. For this project, they were booted in Linux. The Java programs ran on the slaves using the 1.6 version of the Java virtual machine. A Python script was used to control the master, sending the same processing program to each slave in the test run, reading the source files, assigning an input file to a slave, and accepting the scored output from each slave. Using this arrangement, it was easy to vary the number of slaves used in any test run of the batch. The master system also timed the overall processing.

Observations

When detailed monitoring and debugging screen reports were involved, the I/O bursts and overhead communication swamped out the actual processing bursts and were the determining factors in time needed to process the batch. Table 1 shows the batch times for the three schemes. The difference between the linear programming scheme and the concurrent scheme was not significant. The linear programming scheme was probably slowed somewhat by the safety involved in opening and closing the output file as records were appended. The multiple processor grid schemes with screen monitoring reports actually took more time as more slaves were assigned because of the I/O communications overhead.

When all unnecessary screen monitoring was eliminated, the CPU bursts controlled the overall batch processing times. Table 2 shows the average batch processing times under these conditions. Concurrent threads are slightly faster than simple linear programming when run on a single system. Again, opening and closing the disk file may account for this difference. It is not surprising that running only one slave does not differ significantly from either single system times. However, batch processing times are greatly improved as more systems are added to the grid.

Table 1: Average Batch Processing Times with Screen Monitoring

Processing Scheme	Batch Processing Time (Sec)
Linear Programming	9.7
Concurrent Threads	9.2
Master – 1 Slave	9.2
Master – 2 Slaves	10.1
Master – 9 Slaves	13.9

Table 2: Average Processing Times without Screen Monitoring

Processing Scheme	Batch Processing Time (Sec)
Linear Programming	6.1
Concurrent Threads	5.2
Master – 1 Slave	5.7
Master – 2 Slaves	3.2
Master – 9 Slaves	0.9

Comparison of Performance by Gender

As a side issue for those interested, no significant difference in overall “neatness” or “messiness” was detected between male and female middle school students. There were great variations in performance among both genders.

Conclusions

The following three different schemes for processing a batch of bitmapped files were implemented and compared: (1) traditional object-oriented linear programming, (2) concurrent threaded programming, and (3) parallel programming using a master/slave grid. The simple measure of efficiency was overall job completion time for the entire batch. The desired output was a simple comma separated disk file with a record of analysis results for each input file. With all three schemes, use of extensive screen report monitoring introduced enough overhead to effectively be the controlling factor in job completion times. In fact, such communication overhead on the networked master/slave grid resulted in an increase in job times as more slaves were added to the grid.

When all unnecessary screen monitoring was eliminated, the concurrent threaded scheme showed a slight gain in speed over traditional linear programming, and job times were greatly reduced as more slaves were added to the grid. In situations where a large number of input files are to be processed, it would be clearly advantageous to carefully debug a master/slave system, eliminate all unnecessary overhead monitoring communication, and use that system for the final processing.

Acknowledgements

The author is indebted to his colleague, Dr. Steven Benzel, and his student assistant, Elizabeth Crompton, for administration of the master/slave grid and writing the Python scripts to implement running the Java analysis programs using the parallel schemes [5]. He is also indebted to Miss Cynthia Terrill, a middle school teacher for Barrow County Schools in Winder, GA, for administering the drawing tasks to her students.

References

[1] Clendenning, Lee R., "Measurement of Variations in Free-Hand Renderings Using Image Analysis of Geometric Tasks," *The Journal of Computing Sciences in Colleges*, Vol. 21, Number 2, December, 2005, page 349.

[2] Murray, James D., and vanRiper, William, *Encyclopedia of Graphic File Formats*. O'Reilly & Associates, Inc; Sebastopol, CA, 1994, pp. 434–443.

[3] Goetz, Brian, with Tim Peierls, et al., *Java Concurrency in Practice*, Addison Wesley Publishers, 2006.

[4] Lewis, Bil, and Berg, Daniel, *Multithreaded Programming with Java Technology*, Sun Microsystems Press, 2000.

[5] Benzel, Steven, and Crompton, Elizabeth, Unpublished Python Scripts for Implementing a Master/Slave Parallel Computing Grid Using UNIX Operating Systems, 2008.

Biography

Lee R. Clendenning is currently Professor and Chair of the Department of Mathematics and Computer Science at Berry College in Mount Berry, GA. He is a Certified Senior Industrial Technologist through the National Association of Industrial Technology. He holds a Ph.D. in Adult and Technical Education from the University of Illinois. He has more than 45 years of teaching and administrative experience in industrial education, industrial and engineering technology, and computer science programs. He has served NAIT as an accreditation team member, visiting team chair, and reviewer/referee for the *Journal of Industrial Technology*.