Homework 3 Documentation

Author:

William Montgomery

University of Illinois at Chicago

CS 385 - Operating Systems

Fall 2013

Information

Order Finder is a toy program based upon the idea SETIi@home used to demonstrate the use of threads.

The SETIi@home program is a massively parallel, distributed program used to analyze electromagnetic waves for the existence of extra-terrestrial life. Our task was to build a program that simulates this on a much smaller scale.

Overview

It was decided early on that Order Searcher would be designed to be fairly flexible. As such, it was decided that the program would treat each byte as an unsigned integer between 0 and 255 inclusive. To determine the effects of virtual memory on performance it was decided that the program be able to process chunks of data of variable sizes. Additionally, the program should be able to use different criteria to evaluate whether a block exhibits some sort of order.

Criteria

Various criteria were examined for suitability, but were rejected. These include:

Range - A horizontal line with no variance would exhibit the lowest range, while many more "interesting" functions, such as a diagonal line, would exhibit a much greater range.

Maximum absolute value of the change - If we are thinking that these are possible radio waves, a large spike or drop in signal might just by an anomoly in the receiving antennae instead of a real signal.

Standard deviation - This criteria would tend to favor data points that lie in a horizontal manner, instead of data that might exhibit interesting behaviors.

Standard deviation of the change in consecutive elements. This criteria was initially targetted as the best initial criteria. Essentially, this is a meaure of continuity of the data. This was implemented in code and worked reasonably well, however it was abandoned in favor of more sophisticated techniques.

Best fit to a Nth order polynomial - Eventually, this was the criteria that was decided upon. The algorithm is not too terribly difficult, but the implementation involves lots of matrix multiplication. It was decided that the user should be able to choose the order of polynomial to fit from the command line. The techniques used in this implementation could certainly be used to develop any sort of least squares fit to a function, but I believe this is beyond the scope of the project.

Implementation

The solution is implemented with 4 C++ classes. These include the Runner (using the Singleton design patter), Chunker, Chunk, and Semaphore.

[Runner](#) - controls the program, including the launching of threads
[Chunker](#) - divides the input file into chunks of a size determined by command line options
[Chunk](#) - processes the chunk and computes the error
[Semaphore](#) - wrapper for semaphore related system calls

Initially, the program was implemented by reading in chunks of data into memory allocated on the heap as the threads requested it. This was replaced with simply mapping the memory and providing a pointer to the data as well as a the size. This implementation proved to be faster and easier to implement.

Best Fits

The best fit that was implemented was a best fit to an arbitrary order polynomial. Assume that the stream of bytes are readings taken at time t = 0, 1, 2, 3, ...n. The byte at time is taken to be the reading, with a value of between 0 and 255. If we try to fit it to a line of the form C1t + C2 = y, we have ths system of equations:

```
C1 * t0 + C2 = y0
C1 * t1 + C2 = y1
...
C1 * tn + C2 = yn
```

We can represent these as matrices of coefficients. Since we know t and y at each data point, we have a system in the form of A c = b, where A is a matrix of the coefficients of C1 and C2. If we multiply both sides by A' (A-transpose), we get A'*A*c = A'*c, which we can solve using Gaussian elimination and back substitution.

We can then compare how well a polynomial estimates the data by computing the root mean squared (RMS) error. The sum of the square of the difference between the actual reading and the estimated reading is divided by the number of elements. The square root is taken of this value, which is simply a floating point number. The lower the error, the better the approximation of the polynomial.

Unfortunately, this method is not conducive to implementing branch and bound. The error term is the last thing to compute, and it's computation is trivial compared to the solving of the linear equations. The best fit is compared to the current best fit, and the best is replaced by the current chunk if it is better.

Project Directories

There are two directories included. The docs directory has the ggenerated documentation and the orderSearcher directory has the source, makefile, and test script.

Makefile

A makefile is included. The programs can be built by entering make at the command line.

Sample Commands

To run, the program takes 4 command line arguments.

inputDataFile - The name of the file to process
chunkSize - The size of the chunk
nThreads - The number of processor threads to launch
polynomialOrder - The order of the polynomial to best fit

For example, the command orderSearcher diver.raw 4096 5 2 will process the file "diver.raw" in chunks of size 4096, with 5 processing threads (plus one thread to print the results) and will try to fit a quadratic to the data.

Output

The program outputs in two phases. In the first phase, it outputs statistics for the best fit and the error. Results are displayed only if the error is better (less than) the current best.

At the end of the processing, the best fit is displayed. The entire chunk is printed out in a vertical graph (so chunksizes of arbitrary sizes can be displayed). The first column is the value of t within the chunk, the second is the value at that time point, and the third represents the value in graphical form. The asterisks represent the value of the data points, while the O's represent the function evaluated at that time. For example:

```
################################################################################
Best polynomial:
y = 0 * x^5 + 0 * x^4 + 0 * x^3 + 0 * x^2 + 0 * x^1 + 223

Error: 0
################################################################################
       0 223 *******************************************************O
       1 223 *******************************************************O
       2 223 *******************************************************O
       3 223 *******************************************************O
       4 223 *******************************************************O
       5 223 *******************************************************O
       6 223 *******************************************************O
       7 223 *******************************************************O
```

Testing

A test bash script is provided in the source directory for various combinations of the commands.

Results

These are the results running locally.

As expected, multiple threads speed up execution time. The results below are from the test script. The real time is probably the most important statistic for the end user, as it determines the time it takes for the final result to be displayed. With one processing thread and one printing thread, it takes over 17 1/2 seconds to finish the result on the test machine (which has 4 cores + hyperthreading for a total of 8 virtual processors). The time decreases until there are 8 threads, at which point

the time starts to rise again. This is as expected because more threads increase the overhead.

```
Running timing on different numbers of threads
NumberThreads   Real    User    Sys
            1   0:17.42 19.80   15.01
            2   0:08.87 18.73   7.87
            3   0:06.05 18.49   5.67
            4   0:05.88 23.82   5.52
            5   0:05.66 28.44   5.47
            6   0:04.91 29.24   5.04
            7   0:04.89 33.91   4.68
            8   0:04.82 33.71   4.37
            9   0:04.84 34.04   3.91
           10   0:04.82 34.29   3.55
           11   0:05.01 35.61   3.78
           12   0:05.10 36.44   3.56
           13   0:05.14 35.89   3.81
           14   0:05.07 36.22   3.37
           15   0:05.10 36.23   3.41
           16   0:05.13 36.06   3.48
```

Below is an image with the processor usage during the first part of this test. When the script starts, there are only two threads (one for processing and one for printing). This increases until all 8 virtual processors are at 100%

At this point in the test script, we look at the effect of different chunk sizes. My assumption that the best size for the chunks would equal the pagesize. This did not seem to hold. As can be seen in the statistics below, the chunksizes of powers of two were roughly the same over the entire test. There was some deviation, but not enough to be conclusive.

```
Running timing on different chunk sizes
Chunksizes are powers of 2
ChunkSize   Real    User    Sys
        8   0:07.01 47.78   6.83
       16   0:07.25 50.02   7.13
       32   0:07.07 48.57   7.12
       64   0:07.08 48.71   7.09
      128   0:07.33 50.50   7.22
      256   0:07.01 48.10   7.13
      512   0:06.83 46.78   6.92
     1024   0:07.01 48.38   6.90
     2048   0:07.08 48.78   7.02
     4096   0:07.33 50.43   6.92
     8192   0:07.84 54.20   7.40
    16384   0:08.13 56.29   7.95
    32768   0:07.16 49.27   7.17
    65536   0:07.04 48.45   7.11
```

When I looked at chunk size that were not a power of 2, I saw a slowdown, although not huge. I was expecting more of a difference. I suspect that the test machine had copious amounts of RAM (16 GB) and the fact that I process the file sequentially are the reasons I did not exxperience a signficant slowdown. I beieve that a non-sequential processing of the file on a machine with limited RAM would benefit more from choosing chunk sizes of powers of 2 that match the machine's page size.

Chunksizes are NOT powers of 2

ChunkSize	Real	User	Sys
7	0:07.25	50.06	7.07
9	0:07.30	50.43	7.13
15	0:07.48	51.40	7.43
17	0:07.43	50.59	7.28
31	0:07.99	55.07	7.79
33	0:07.77	53.54	7.69
63	0:07.40	49.70	7.29
65	0:07.10	48.22	6.78
127	0:07.51	51.03	7.20
129	0:07.12	48.97	7.05
255	0:07.80	54.08	7.44
257	0:07.47	51.53	7.14
511	0:07.84	53.87	7.57
513	0:07.97	55.50	7.77
1023	0:08.31	56.17	7.96
1025	0:07.73	46.37	6.67
2047	0:07.87	52.08	7.08
2049	0:06.84	46.72	6.92
4095	0:06.82	47.04	6.67
4097	0:06.87	47.06	6.92
8191	0:06.86	47.64	6.80
8193	0:06.74	46.74	6.82
16383	0:06.76	46.84	6.79
16385	0:06.68	46.17	6.80
32767	0:06.69	46.20	6.75
32769	0:06.72	46.55	6.70
65535	0:06.99	48.42	6.96
65537	0:07.23	50.39	6.99

Best Fits

To test the algorithm, thre program was run against various file types with a set chunk size of 32 and with a polynomial of order 20. The results are below. It should be noted that increasing the chunk size creates more interesting results, but the results are longer. In the graphs below, the asterisks represent the values of the byte and the 'O's represent the best fit solution.

The results for the RAW file were not that interesting, in that there was a chunk with all values at 255. This resulted in an equation that was essentially y = 255.

```
################################################################################
        0 255 ***************************************************************O
        1 255 ***************************************************************O
        2 255 ***************************************************************O
        3 255 ***************************************************************O
        4 255 ***************************************************************O
        5 255 ***************************************************************O
        6 255 ***************************************************************O
        7 255 ***************************************************************O
        8 255 ***************************************************************O
        9 255 ***************************************************************O
       10 255 ***************************************************************O
       11 255 ***************************************************************O
```

```
        12 255 ***************************************************************O
        13 255 ***************************************************************O
        14 255 ***************************************************************O
        15 255 ***************************************************************O
        16 255 ***************************************************************O
        17 255 ***************************************************************O
        18 255 ***************************************************************O
        19 255 ***************************************************************O
        20 255 ***************************************************************O
        21 255 ***************************************************************O
        22 255 ***************************************************************O
        23 255 ***************************************************************O
        24 255 ***************************************************************O
        25 255 ***************************************************************O
        26 255 ***************************************************************O
        27 255 ***************************************************************O
        28 255 ***************************************************************O
        29 255 ***************************************************************O
        30 255 ***************************************************************O
        31 255 ***************************************************************O
################################################################################
Best polynomial:
y = 5.72849e-47 * x^20 + 0 * x^19 + 4.90777e-44 * x^18 + -1.4311e-42 * x^17 + 0 * x^16
+ 0 * x^15 + 3.48015e-38 * x^14 + -1.0001e-36 * x^13 + 0 * x^12 + 0 * x^11 + 0 * x^10
+ 0 * x^9 + 0 * x^8 + 0 * x^7 + 0 * x^6 + 0 * x^5 + 0 * x^4 + 0 * x^3 + 0 * x^2 + 0 *
x^1 + 255

Error: 1.68188e-17
################################################################################
################################################################################
Processing completed for diver.raw
User time: 59.304342
System time: 10.304584
Processing used 5 processing threads and 1 printing thread.
26 candidates were found.
The best candidate is plotted above.
################################################################################
```

The analysis of the JPG file resulted in a best fit curve that was simply a straight line with the equation y = 0. Surely this shows order, but perhaps is the least interesting.

```
  1   0 O
  2   0 O
  3   0 O
  4   0 O
  5   0 O
  6   0 O
  7   0 O
  8   0 O
  9   0 O
 10   0 O
 11   0 O
 12   0 O
 13   0 O
 14   0 O
 15   0 O
 16   0 O
 17   0 O
 18   0 O
 19   0 O
 20   0 O
 21   0 O
 22   0 O
 23   0 O
 24   0 O
 25   0 O
 26   0 O
 27   0 O
 28   0 O
```

```
       29   0 O
       30   0 O
       31   0 O
################################################################################
Best polynomial:
y = 0 * x^20 + 0 * x^19 + 0 * x^18 + 0 * x^17 + 0 * x^16 + 0 * x^15 + 0 * x^14 + 0 *
x^13 + 0 * x^12 + 0 * x^11 + 0 * x^10 + 0 * x^9 + 0 * x^8 + 0 * x^7 + 0 * x^6 + 0 *
x^5 + 0 * x^4 + 0 * x^3 + 0 * x^2 + 0 * x^1 + 0

Error: 0
################################################################################
################################################################################
Processing completed for diver.jpg
User time: 5.202088
System time: 0.854372
Processing used 5 processing threads and 1 printing thread.
5 candidates were found.
The best candidate is plotted above.
################################################################################
```

The analysis of the BMP file was similar to most of the other files. The best fit turned out to be a straight line through y = 255.

```
################################################################################
        0 255 ****************************************************************O
        1 255 ****************************************************************O
        2 255 ****************************************************************O
        3 255 ****************************************************************O
        4 255 ****************************************************************O
        5 255 ****************************************************************O
        6 255 ****************************************************************O
        7 255 ****************************************************************O
        8 255 ****************************************************************O
        9 255 ****************************************************************O
       10 255 ****************************************************************O
       11 255 ****************************************************************O
       12 255 ****************************************************************O
       13 255 ****************************************************************O
       14 255 ****************************************************************O
       15 255 ****************************************************************O
       16 255 ****************************************************************O
       17 255 ****************************************************************O
       18 255 ****************************************************************O
       19 255 ****************************************************************O
       20 255 ****************************************************************O
       21 255 ****************************************************************O
       22 255 ****************************************************************O
       23 255 ****************************************************************O
       24 255 ****************************************************************O
       25 255 ****************************************************************O
       26 255 ****************************************************************O
       27 255 ****************************************************************O
       28 255 ****************************************************************O
       29 255 ****************************************************************O
       30 255 ****************************************************************O
       31 255 ****************************************************************O
################################################################################
Best polynomial:
y = 5.72849e-47 * x^20 + 0 * x^19 + 4.90777e-44 * x^18 + -1.4311e-42 * x^17 + 0 * x^16
+ 0 * x^15 + 3.48015e-38 * x^14 + -1.0001e-36 * x^13 + 0 * x^12 + 0 * x^11 + 0 * x^10
+ 0 * x^9 + 0 * x^8 + 0 * x^7 + 0 * x^6 + 0 * x^5 + 0 * x^4 + 0 * x^3 + 0 * x^2 + 0 *
x^1 + 255

Error: 1.68188e-17
################################################################################
################################################################################
Processing completed for diver.bmp
User time: 56.471479
System time: 9.679683
```

```
Processing used 5 processing threads and 1 printing thread.
27 candidates were found.
The best candidate is plotted above.
################################################################################
```

For the GIF analysis, the results were more interesting than most filetypes. It an clearly be seen that the best fit tries to mimic the curve of the data.

```
################################################################################
        0 136 **********************************        O
        1 120 ******************************             O
        2 136 **********************************         O
        3 152 **************************************     O
        4 136 **********************************         O
        5 152 **************************************     O
        6 168 ****************************************** O
        7 152 **************************************     O
        8 168 ****************************************** O
        9 184 *******************************************O**
       10 168 ****************************************** O
       11 184 *******************************************O**
       12 200 *******************************************O******
       13 184 *******************************************O**
       14 200 *******************************************O******
       15 216 *******************************************O**********
       16 200 *******************************************O******
       17 216 *******************************************O**********
       18 232 *******************************************O**************
       19 107 **************************                 O
       20 155 **************************************     O
       21 199 *******************************************O*****
       22 127 *******************************            O
       23 183 *******************************************O*
       24 233 *******************************************O**************
       25 120 ******************************             O
       26 172 *******************************************O
       27 214 ******************************************O**********
       28 142 ***********************************       O
       29 181 *****************************************O***
       30 215 ****************************************O************
       31 136 **********************************     O
################################################################################
Best polynomial:
y = -1.03197e-30 * x^20 + -3.20879e-29 * x^19 + -9.94446e-28 * x^18 + -3.06981e-26 *
x^17 + -9.43158e-25 * x^16 + -2.8812e-23 * x^15 + -8.74135e-22 * x^14 + -2.63083e-20 *
x^13 + -7.8486e-19 * x^12 + -2.32282e-17 * x^11 + -6.85326e-16 * x^10 + -2.04487e-14 *
x^9 + -6.36941e-13 * x^8 + -2.18076e-11 * x^7 + -8.60011e-10 * x^6 + -3.87974e-08 *
x^5 + -1.85855e-06 * x^4 + -8.27721e-05 * x^3 + -0.00220804 * x^2 + 0.268049 * x^1 +
171.5

Error: 34.0906
################################################################################
################################################################################
Processing completed for diver.gif
User time: 9.587693
System time: 1.649882
Processing used 5 processing threads and 1 printing thread.
4 candidates were found.
The best candidate is plotted above.
################################################################################
```

For the diver.nz file, the best fit was essentially a straight line.

```
################################################################################
        0 255 ***************************************************************O
        1 255 ***************************************************************O
        2 255 ***************************************************************O
        3 255 ***************************************************************O
```

```
 4 255 ***********************************************************************O
 5 255 ***********************************************************************O
 6 255 ***********************************************************************O
 7 255 ***********************************************************************O
 8 255 ***********************************************************************O
 9 255 ***********************************************************************O
10 255 ***********************************************************************O
11 255 ***********************************************************************O
12 255 ***********************************************************************O
13 255 ***********************************************************************O
14 255 ***********************************************************************O
15 255 ***********************************************************************O
16 255 ***********************************************************************O
17 255 ***********************************************************************O
18 255 ***********************************************************************O
19 255 ***********************************************************************O
20 255 ***********************************************************************O
21 255 ***********************************************************************O
22 255 ***********************************************************************O
23 255 ***********************************************************************O
24 255 ***********************************************************************O
25 255 ***********************************************************************O
26 255 ***********************************************************************O
27 255 ***********************************************************************O
28 255 ***********************************************************************O
29 255 ***********************************************************************O
30 255 ***********************************************************************O
31 255 ***********************************************************************O
################################################################################
Best polynomial:
y = 5.72849e-47 * x^20 + 0 * x^19 + 4.90777e-44 * x^18 + -1.4311e-42 * x^17 + 0 * x^16
+ 0 * x^15 + 3.48015e-38 * x^14 + -1.0001e-36 * x^13 + 0 * x^12 + 0 * x^11 + 0 * x^10
+ 0 * x^9 + 0 * x^8 + 0 * x^7 + 0 * x^6 + 0 * x^5 + 0 * x^4 + 0 * x^3 + 0 * x^2 + 0 *
x^1 + 255

Error: 1.68188e-17
################################################################################
################################################################################
Processing completed for diver.nz
User time: 61.419652
System time: 10.421154
Processing used 5 processing threads and 1 printing thread.
25 candidates were found.
The best candidate is plotted above.
################################################################################
```

Conclusion

This was an interesting project, and I believe that I created a program that works as intended. I was able to successfully find order in files of different types. I believe that with a stream of readings from a receiver, I could use some of these techniques to design a program to search out the order.

Todo List

Class Chunk

In an ideal world, I would extract all calculations to another class, so that I could offer different calculations.

Member Runner::processArgs ()

Need to get the order for the polynomial

Member Runner::run ()

When the program finishes it should report final statistics, such as the number of threads used, the number of candidates found, the time it took, and also report the best candidates found, with their criteria and plots.

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

File Index

File List

Here is a list of all files with brief descriptions:

Class Documentation

Chunk Class Reference

The Chunk class Encapsulates a chunk of data.

`#include <chunk.h>`

Public Member Functions

Chunk (int)

Chunk::Chunk Constructor.

~Chunk ()

Chunk::~Chunk Destructor.

Chunk * process ()

Chunk::process Process this chunk.

Chunk * setData (char *)

Chunk::setData Setter for data.

Chunk * setSize (int)

Chunk::setSize Setter for size.

Chunk * setPosition (int)

Chunk::setPosition Setter for position.

long double getError ()

Chunk::getError Getter for error.

Chunk * printBlock ()

Chunk::printBlock Prints out the block.

Chunk * printData ()

Chunk::printData Prints out the data.

Private Member Functions

Chunk * fillArray ()

Chunk::fillArray Convert the bytes to an array of long doubles.

Chunk * processBlock ()

Chunk::processBlock Process the block.

Chunk * computeSum ()

Chunk::computeSum Computes the sum of an array.

Chunk * computeAvgDelta ()

Chunk::computeAvgDelta Computes the average change between consecutive elements.

Chunk * computeStdDevDelta ()

Chunk::computeStdDevDelta Computes the standard deviation of the change between elements.

Chunk * computeBestFit ()

Chunk::computeBestFit Tries to fit a polynomial of arbitrary order to the data. We set x equal to the index of the array, y to the value, and compute the best A and B.

Chunk * computeError ()

Chunk::computeError Compute the difference between the best fit polynomial and the actual values.

long double ** createMatrix (int, int)

Chunk::createMatrix Creates a 2-dimensional array.

long double evaluatePolynomial (long double)
Chunk * freeMatrix (long double **, int)

Chunk::freeMatrix Frees a matrix.

Chunk * backSubstitution ()

Chunk::backSubstitution Perform the back substitution to solve a set of linear equations.

Chunk * gaussianElimination ()

Chunk::gaussianElimination Perform the Gaussian elimination step of solving the linear equations. This method introduces rounding error that might lead to a loss of significance, but is easiest to implement.

Chunk * computeATransposeA ()

*Chunk::computeATransposeA Computes the left side of the equation A(T) * A.*

Chunk * computeATransposeB ()

*Chunk::computeATransposeB Computes the right side of the equation A(T) * b.*

Private Attributes

char * data
int size
int numCoeffs
long double * array
long double sum
long double avgDelta
long double stdDevDelta
long double error
long double ** leastSquareCooefs
long double * coeffs
long double ** ATransposeA
long double * ATransposeB
int position

Detailed Description

The Chunk class Encapsulates a chunk of data.

Todo:

In an ideal world, I would extract all calculations to another class, so that I could offer different calculations.

Definition at line 22 of file chunk.h.

Constructor & Destructor Documentation

Chunk::Chunk (int *polynomialOrder*)

Chunk::Chunk Constructor.

Definition at line 27 of file chunk.cpp.

```
28 {
29     this->numCoeffs = polynomialOrder + 1;
30     this->size = 0;
31     this->position = 0;
32     this->sum = 0.0;
33     this->avgDelta = 0.0;
34     this->coeffs = (long double*) malloc (this->numCoeffs * sizeof(long double));
35     for (int i = 0; i < this->numCoeffs; i++){
36         this->coeffs[i] = 0.0;
37     }
38 }
```

Chunk::~Chunk ()

Chunk::~Chunk Destructor.

Definition at line 43 of file chunk.cpp.

```
43                {
44     free (coeffs);
45     free (array);
46 }
```

Member Function Documentation

Chunk * Chunk::backSubstitution () [private]

Chunk::backSubstitution Perform the back substitution to solve a set of linear equations.

Returns:

this

Definition at line 126 of file chunk.cpp.

```
127 {
128     for (int i = this->numCoeffs - 1; i >= 0; i--){
129         for (int j = i + 1; j < this->numCoeffs; j++){
130             ATransposeB[i] = ATransposeB[i] - ATransposeA[i][j] * this->coeffs[j];
131         }
132         this->coeffs[i] = ATransposeB[i] / ATransposeA[i][i];
133     }
134     return this;
135 }
```

Chunk * Chunk::computeATransposeA () [private]

Chunk::computeATransposeA Computes the left side of the equation A(T) * A.

Returns:

this

Definition at line 165 of file chunk.cpp.

```
166 {
167     ATransposeA = createMatrix(this->numCoeffs, this->numCoeffs);
168
169     long double sum = 0;
170     for (int i = 0; i < this->numCoeffs; ++i) {
171         for (int j = 0 ; j < this->numCoeffs; j++) {
172             for(int k = 0; k < this->size; k++){
173                 sum = sum + this->leastSquareCooefs[k]
[i]*this->leastSquareCooefs[k][j];
```

```
174             }
175             ATransposeA[i][j] = sum;
176             sum = 0;
177         }
178     }
179
180     return this;
181 }
```

Chunk * Chunk::computeATransposeB () [private]

Chunk::computeATransposeB Computes the right side of the equation A(T) * b.

Returns:

this

Definition at line 187 of file chunk.cpp.

```
188 {
189     ATransposeB = (long double*)malloc(this->numCoeffs * sizeof(long double));
190
191     for (int i = 0; i < this->numCoeffs; ++i) {
192         for(int k = 0; k < this->size; k++){
193             sum = sum + this->leastSquareCooefs[k][i]*this->array[k];
194         }
195        ATransposeB[i] = sum;
196         sum = 0;
197     }
198     return this;
199 }
```

Chunk * Chunk::computeAvgDelta () [private]

Chunk::computeAvgDelta Computes the average change between consecutive elements.

Returns:

this

Definition at line 298 of file chunk.cpp.

```
298                            {
299     avgDelta = 0;
300     double diffSum = 0;
301
302     for (int i = 0; i < size - 1; i++){
303         diffSum += array[i + 1] - array[i];
304     }
305
306     avgDelta = diffSum / double((size - 1));
307
308     return this;
309 }
```

Chunk * Chunk::computeBestFit () [private]

Chunk::computeBestFit Tries to fit a polynomial of arbitrary order to the data. We set x equal to the index of the array, y to the value, and compute the best A and B.

Returns:

this

Definition at line 215 of file chunk.cpp.

```
                         {
    // To compute we use a matrix for the coefficients
    this->leastSquareCooefs = createMatrix(this->size, this->numCoeffs);

    // Make the coefficient matrix A
    for (int i = 0; i < size; i++){
        for (int j = 0; j < this->numCoeffs; j++){
            this->leastSquareCooefs[i][j] = 1.0;
            for (int k = j; k < this->numCoeffs - 1; k++){
                this->leastSquareCooefs[i][j] *= double(i);
            }
        }
    }

    // Compute A' * A
    computeATransposeA();

    // Compute A' * b
    computeATransposeB();

    // Gaussian - Elimination
    gaussianElimination();

    // Back - Substitution
    backSubstitution();

    // Clean up
    freeMatrix(ATransposeA, this->numCoeffs);
    free(ATransposeB);
    freeMatrix(leastSquareCooefs, this->size);

    return this;
}
```

Chunk * Chunk::computeError () [private]

Chunk::computeError Compute the difference between the best fit polynomial and the actual values.

Returns:

this

Definition at line 109 of file chunk.cpp.

```
109                        {
110     long double sum = 0;
111     long double value = 0;
112     for (int i = 0; i < this->size; i++){
113         value = evaluatePolynomial(double(i));
114         sum += (value - this->array[i]) * (value - this->array[i]);
115     }
116 
117     this->error = sqrt(sum / this->size);
118     return this;
119 }
```

Chunk * Chunk::computeStdDevDelta () [private]

Chunk::computeStdDevDelta Computes the standard deviation of the change between elements.

Returns:

this

Definition at line 316 of file chunk.cpp.

```
316                           {
317     double sum = 0;
318     double temp;
319
320     for (int i = 0; i < size - 1; i++){
321         temp = (array[i + 1] - array[i]) - avgDelta;
322         sum += temp * temp;
323     }
324
325     stdDevDelta = sqrt(sum / (double) (size - 1)); // There are n - 1 deltas
326
327     return this;
328 }
```

Chunk * Chunk::computeSum () `[private]`

Chunk::computeSum Computes the sum of an array.

Returns:

this

Definition at line 284 of file chunk.cpp.

```
284                       {
285     this->sum = 0;
286     for (int i = 0; i < size; i++){
287         this->sum += array[i];
288     }
289
290     return this;
291 }
```

long double ** Chunk::createMatrix (int *nRows*, int *nCols*) `[private]`

Chunk::createMatrix Creates a 2-dimensional array.

Parameters:

nRows	The number of rows
nCols	The number of columns

Returns:

Pointer to the start of the array

Definition at line 255 of file chunk.cpp.

```
255                                                     {
256     long double **matrix;
257
258     matrix = (long double **) malloc (sizeof(long double*) * nRows);
259     for (int i = 0; i < nRows; i++){
260         matrix[i] = (long double *) calloc(sizeof(long double), nCols);
261     }
262     return matrix;
263 }
```

long double Chunk::evaluatePolynomial (long double *time*) `[private]`

Definition at line 201 of file chunk.cpp.

```
201                                                      {
202     long double sum = this->coeffs[0];
203     for (int i = 1; i < this->numCoeffs; i++){
204         sum += this->coeffs[i] + time * sum;
205     }
```

```
206     return sum;
207 }
```

Chunk * Chunk::fillArray () [private]

Chunk::fillArray Convert the bytes to an array of long doubles.

Returns:

this

Definition at line 73 of file chunk.cpp.

```
73                      {
74      double number = 0;
75      for (int i = 0; i < size; i++){
76          number = double(*(data + i) & 0xff);
77          array[i] = number;
78      }
79
80      return this;
81  }
```

Chunk * Chunk::freeMatrix (long double ** *matrix*, int *nRows*) [private]

Chunk::freeMatrix Frees a matrix.

Parameters:

matrix	A pointer to the start of the matrix
nRows	The number of rows

Returns:

this

Definition at line 271 of file chunk.cpp.

```
271                                                             {
272     for (int i = 0; i < nRows; i++){
273         free (matrix[i]);
274     }
275     free (matrix);
276     matrix = NULL;
277     return this;
278 }
```

Chunk * Chunk::gaussianElimination () [private]

Chunk::gaussianElimination Perform the Gaussian elimination step of solving the linear equations. This method introduces rounding error that might lead to a loss of significance, but is easiest to implement.

Returns:

this

Definition at line 143 of file chunk.cpp.

```
144 {
145     long double mult = 0;
146     for (int j = 0; j < this->numCoeffs - 1; j++){
147         if (abs(ATransposeA[j][j]) < pow(2.0, -32.0)){  // Zero pivot encoutered
148             for (int i = j + 1; j < this->numCoeffs; j++){
149                 mult = ATransposeA[i][j] / ATransposeA[j][j];
```

```
150                 for (int k = j + 1; j < this->numCoeffs; k++){
151                     ATransposeA[i][k] = ATransposeA[i][k] - (mult * ATransposeA[j][k]);
152                 }
153                 ATransposeB[i] = ATransposeB[i] - (mult * ATransposeB[j]);
154             }
155         }
156     }
157 
158     return this;
159 }
```

long double Chunk::getError ()

Chunk::getError Getter for error.

Returns:

The RMS error in the polynomial

Definition at line 424 of file chunk.cpp.

Referenced by Runner::setBestChunk().

```
424                              {
425     return this->error;
426 }
```

Chunk * Chunk::printBlock ()

Chunk::printBlock Prints out the block.

Returns:

this

Definition at line 387 of file chunk.cpp.

```
387                          {
388     cout << "##################################################################################" << endl;
389 //    cout << "Sum was " << sum << endl;
390 //    cout << "Average delta was " << avgDelta << endl;
391 //    cout << "Standard deviation of delta is " << stdDevDelta << endl;
392     cout << "Best polynomial:" << endl;
393     cout << "y = ";
394     for (int i = 0; i < numCoeffs; i++){
395         if (i > 0) {
396             cout << "+ ";
397         }
398         cout << coeffs[i];
399         if (i < numCoeffs - 1) {
400             cout << " * " << "x^" << numCoeffs - (i + 1) << " ";
401         }
402     }
403     cout << endl << endl;
404     cout << "Error: " << error << endl;
405     cout << "##################################################################################" << endl;
406 
407     return this;
408 }
```

Chunk * Chunk::printData ()

Chunk::printData Prints out the data.

Returns:

this

Definition at line 357 of file chunk.cpp.

```
358 {
359     double scaleFactor = 4.0;
360     const int bufferSize = 68;
361     for(int i = 0; i < size; i++){
362         printf ("%9d %3d ", i, (int)array[i]);
363         char buffer[bufferSize] = "
";
364         int numStars = (int)(array[i] / scaleFactor);
365 
366         for (int j = 0; j < numStars; j++) {
367             buffer[j] = '*';
368         }
369 
370         // Evaluate the function and print it if within the range
371         int evaluation = (int) (evaluatePolynomial(double(i)) / scaleFactor);
372 
373         if (evaluation < bufferSize && evaluation >= 0){
374             buffer[evaluation] = 'O';
375         }
376 
377         cout << buffer << endl;
378     }
379 
380     return this;
381 }
```

Chunk * Chunk::process ()

Chunk::process Process this chunk.

Returns:

this

Definition at line 52 of file chunk.cpp.

References Runner::getBestChunk(), Runner::getRunner(), and Runner::setBestChunk().

Referenced by Runner::analyze().

```
52                    {
53     processBlock();
54 
55     // Compare against the best chunk so far
56     // If better set this one as the best
57     Chunk *best = Runner::getRunner()->getBestChunk();
58     if (best == NULL || this->error <= best->getError()) {
59         Runner::getRunner()->setBestChunk(this);
60     }
61     else {
62         delete this;
63         return NULL;
64     }
65 
66     return this;
67 }
```

Chunk * Chunk::processBlock () `[private]`

Chunk::processBlock Process the block.

Returns:

this

Definition at line 87 of file chunk.cpp.

```
87                          {
88 //    computeSum();
89 //    computeAvgDelta();
90 //    computeStdDevDelta();
91     if (size >= numCoeffs){
92         fillArray();
93         computeBestFit();
94         computeError();
95     }
96     else {
97         printf ("Block is less than number of coefficients!\n");
98         error = double(INT_MAX);
99     }
100
101     return this;
102 }
```

Chunk * Chunk::setData (char * *data*)

Chunk::setData Setter for data.

Parameters:

data	The data

Returns:

this

Definition at line 347 of file chunk.cpp.

Referenced by Chunker::getChunk().

```
347                                 {
348     this->data = data;
349
350     return this;
351 }
```

Chunk * Chunk::setPosition (int *position*)

Chunk::setPosition Setter for position.

Parameters:

position	The position

Returns:

this

Definition at line 415 of file chunk.cpp.

Referenced by Chunker::getChunk().

```
415                                     {
416     this->position = position;
417     return this;
418 }
```

Chunk * Chunk::setSize (int *size*)

Chunk::setSize Setter for size.

Parameters:

size	The size

Returns:

this

Definition at line 335 of file chunk.cpp.

Referenced by Chunker::getChunk().

```
335                          {
336     this->size = size;
337     array = (long double *) malloc (sizeof(long double) * size);
338
339     return this;
340 }
```

Member Data Documentation

long double* Chunk::array [private]

The array of bytes converted to long doubles

Definition at line 29 of file chunk.h.

long double** Chunk::ATransposeA [private]

The left side of the least square equation

Definition at line 37 of file chunk.h.

long double* Chunk::ATransposeB [private]

The right side of the least square equation

Definition at line 38 of file chunk.h.

long double Chunk::avgDelta [private]

The average change between consecutive elements

Definition at line 31 of file chunk.h.

long double* Chunk::coeffs [private]

The calculated coefficients

Definition at line 36 of file chunk.h.

char* Chunk::data [private]

The data to process

Definition at line 25 of file chunk.h.

long double Chunk::error [private]

The RMS error between the calculated polynomial and the data

Definition at line 33 of file chunk.h.

long double** Chunk::leastSquareCooefs [private]

Matrix of coefficients for least square calculation

Definition at line 35 of file chunk.h.

int Chunk::numCoeffs `[private]`

The number of coefficients (should be the order + 1)

Definition at line 27 of file chunk.h.

int Chunk::position `[private]`

The position of the chunk

Definition at line 39 of file chunk.h.

int Chunk::size `[private]`

The size of the data to process

Definition at line 26 of file chunk.h.

long double Chunk::stdDevDelta `[private]`

The standard deviation of the change between consecutive elements

Definition at line 32 of file chunk.h.

long double Chunk::sum `[private]`

The sum of the values

Definition at line 30 of file chunk.h.

The documentation for this class was generated from the following files:

orderSearcher/chunk.h
orderSearcher/chunk.cpp

Chunker Class Reference

The Chunker class Encapsulates the process of breaking down a file into chunks.

`#include <chunker.h>`

Public Member Functions

Chunker (char *, int, int)

Chunker::Chunker Constructor.

~Chunker ()

Chunker::~Chunker Destructor.

Chunker * map ()

Chunker::map Map the file to virtual memory.

Chunker * unmap ()

Chunker::unmap Unmaps the file from virtual memory.

Chunk * getChunk ()

Chunker::getChunk Gets a chunk.

Private Types

enum semaphores { READ }

Private Member Functions

Chunker * setFileSize ()

Chunker::setFileSize Gets the size of the file and sets fileSize.

Chunker * openFile ()

Chunker::openFile Opens a file for reading.

Chunker * closeFile ()

Chunker::closeFile Closes a file.

bool moreChunks ()

Chunker::moreChunks Boolean of whether there are more chunks to process.

Private Attributes

char * filename
int chunkSize
int polynomialOrder
long long fileSize
long long bytesRead
int fd
char * mappedFile
char * currentPosition
Semaphore * semaphore

Detailed Description

The Chunker class Encapsulates the process of breaking down a file into chunks.

Definition at line 25 of file chunker.h.

Member Enumeration Documentation

enum Chunker::semaphores [private]

Enumerator

READ

Definition at line 28 of file chunker.h.

```
28 {READ};
```

Constructor & Destructor Documentation

Chunker::Chunker (char * *filename*, int *chunkSize*, int *polynomialOrder*)

Chunker::Chunker Constructor.

Parameters:

filename	The filename to chunk
chunkSize	The size of the chunk
polynomialOrder	The order of the polynomial to fit

Definition at line 28 of file chunker.cpp.

References bytesRead, chunkSize, filename, Semaphore::get(), map(), polynomialOrder, semaphore, Semaphore::set(), and setFileSize().

```
29 {
30     this->filename = filename;
31     this->chunkSize = chunkSize;
32     this->polynomialOrder = polynomialOrder;
33
34     this->bytesRead = 0;
35     this->semaphore = new Semaphore(NUMSEMAPHORES);
36     this->semaphore->get()->set(1);
37     setFileSize();
38     map();
39 }
```

Chunker::~Chunker ()

Chunker::~Chunker Destructor.

Definition at line 44 of file chunker.cpp.

References semaphore, and unmap().

```
44                      {
45     unmap();
46     delete semaphore;
47 }
```

Member Function Documentation

Chunker * Chunker::closeFile () [private]

Chunker::closeFile Closes a file.

Returns:

this

Definition at line 93 of file chunker.cpp.

References fd.

Referenced by unmap().

```
93                            {
94       close(fd);
95       return this;
96 }
```

Chunk * Chunker::getChunk ()

Chunker::getChunk Gets a chunk.

Returns:

Pointer to new Chunk, or NULL if none left to process

Definition at line 127 of file chunker.cpp.

References bytesRead, chunkSize, currentPosition, fileSize, moreChunks(), polynomialOrder, READ, semaphore, Chunk::setData(), Chunk::setPosition(), Chunk::setSize(), Semaphore::signal(), and Semaphore::wait().

```
127                          {
128     semaphore->wait(READ);
129     Chunk *chunk = NULL;
130     if (moreChunks()) {
131         chunk = new Chunk(this->polynomialOrder);
132         chunk->setData(currentPosition);
133         chunk->setPosition(bytesRead);
134
135         // Determine the new position
136         currentPosition = currentPosition + chunkSize;
137         int size = chunkSize;
138
139         // Determine the size of the chunk
140         if (!moreChunks()){
141             size = fileSize - bytesRead;
142         }
143         bytesRead += size;
144         chunk->setSize(size);
145     }
146     semaphore->signal(READ);
147     return chunk;
148 }
```

Chunker * Chunker::map ()

Chunker::map Map the file to virtual memory.

Returns:

this

Definition at line 53 of file chunker.cpp.

References currentPosition, fd, fileSize, mappedFile, and openFile().

Referenced by Chunker().

```
53                  {
54      openFile();
55      mappedFile = (char*)mmap (0, fileSize, PROT_READ, MAP_SHARED, fd, 0);
56      if (mappedFile == (caddr_t) -1){
57          perror("Could not map file:");
58          exit(-1);
59      }
60      currentPosition = mappedFile;
61      return this;
62 }
```

bool Chunker::moreChunks () [private]

Chunker::moreChunks Boolean of whether there are more chunks to process.

Returns:

true if more to process, false otherwise

Definition at line 119 of file chunker.cpp.

References currentPosition, fileSize, and mappedFile.

Referenced by getChunk().

```
119                       {
120     return (currentPosition < (mappedFile + fileSize));
121 }
```

Chunker * Chunker::openFile () [private]

Chunker::openFile Opens a file for reading.

Returns:

this

Definition at line 80 of file chunker.cpp.

References fd, and filename.

Referenced by map().

```
80                        {
81      fd = open(filename,  O_RDONLY, (mode_t)0600);
82      if (fd <= 0) {
83          cout << "The file could not be opened." << endl;
84          exit(-1);
85      }
86      return this;
87 }
```

Chunker * Chunker::setFileSize () [private]

Chunker::setFileSize Gets the size of the file and sets fileSize.

Returns:

this

Definition at line 102 of file chunker.cpp.

References filename, and fileSize.

Referenced by Chunker().

```
                                {
    struct stat results;

    if (stat(filename, &results) == 0) {
        fileSize = results.st_size;
    }
    else {
        perror("Could not determine file size:");
        exit(-1);
    }
    return this;
}
```

Chunker * Chunker::unmap ()

Chunker::unmap Unmaps the file from virtual memory.

Returns:

this

Definition at line 68 of file chunker.cpp.

References closeFile(), fileSize, and mappedFile.

Referenced by ~Chunker().

```
68                          {
69     if (munmap(mappedFile, fileSize) < 0){
70         perror("Could not unmap file:");
71     }
72     closeFile();
73     return this;
74 }
```

Member Data Documentation

long long Chunker::bytesRead [private]

The number of bytes "read" so far.

Definition at line 34 of file chunker.h.

Referenced by Chunker(), and getChunk().

int Chunker::chunkSize [private]

The size of the chunks (should be the same for all except possibly the last chunk).

Definition at line 30 of file chunker.h.

Referenced by Chunker(), and getChunk().

char* Chunker::currentPosition [private]

The current position of the file.

Definition at line 38 of file chunker.h.

Referenced by getChunk(), map(), and moreChunks().

int Chunker::fd [private]

A file descriptor to the file.

Definition at line 36 of file chunker.h.

Referenced by closeFile(), map(), and openFile().

char* Chunker::filename [private]

The name of the file.

Definition at line 29 of file chunker.h.

Referenced by Chunker(), openFile(), and setFileSize().

long long Chunker::fileSize [private]

The size of the file.

Definition at line 33 of file chunker.h.

Referenced by getChunk(), map(), moreChunks(), setFileSize(), and unmap().

char* Chunker::mappedFile [private]

A pointer to the mapped file.

Definition at line 37 of file chunker.h.

Referenced by map(), moreChunks(), and unmap().

int Chunker::polynomialOrder [private]

The order of the polynomial.

Definition at line 31 of file chunker.h.

Referenced by Chunker(), and getChunk().

Semaphore* Chunker::semaphore [private]

A semaphore, so that the same chunk is not given twice

Definition at line 40 of file chunker.h.

Referenced by Chunker(), getChunk(), and ~Chunker().

The documentation for this class was generated from the following files:

orderSearcher/chunker.h
orderSearcher/chunker.cpp

Runner Class Reference

The Runner class Singleton class that controls the flow of the program.

`#include <runner.h>`

Public Types

enum semaphores { BESTCHUNK, PRINTED }

Public Member Functions

Runner * run ()

Runner::run Runs the runner.

Runner (int argc, char **argv)

Runner::Runner Constructor for Runner.

~Runner ()

Runner::~Runner Destructor.

Runner * setBestChunk (Chunk *chunk)

Runner::setBestChunk Tries to set a chunk as the best chunk.

Chunk * getBestChunk ()

Runner::getBestChunk Getter for bestChunk.

Static Public Member Functions

static Runner * getRunner (int, char **)

Runner::getRunner Gets the Singleton (or creates if not created)

static Runner * getRunner ()

Runner::getRunner Gets the Singleton.

Private Member Functions

Runner * processArgs ()

Runner::processArgs Processes the command line arguments.

Runner * writeAuthorInformation ()

Master::writeAuthorInformation Writes the author information to stdout.

Runner * launchThreads ()

Runner::launchThreads Launches the threads to process the input file. If n threads are called from the command line, the program launches n + 1 threads. n threads are used for processing, and one is used for displaying the results.

Static Private Member Functions

static void * analyze (void *)

Runner::analyze Static method used for the processing of chunks of data.

static void * displayResults (void *)

Runner::displayResults Static method used for the displaying of results.

Private Attributes

int argc

char ** argv
int nThreads
int chunkSize
int polynomialOrder
int candidates
Chunker * chunker
Semaphore * semaphore
Chunk * bestChunk
bool printed
bool threadsDone

Static Private Attributes

static bool isSet = false
static Runner * runner = NULL

Detailed Description

The Runner class Singleton class that controls the flow of the program.

Definition at line 25 of file runner.h.

Member Enumeration Documentation

enum Runner::semaphores

Enumerator

BESTCHUNK
PRINTED

Definition at line 51 of file runner.h.

```
51 {BESTCHUNK, PRINTED};
```

Constructor & Destructor Documentation

Runner::Runner (int *argc*, char ** *argv*)

Runner::Runner Constructor for Runner.

Parameters:

argc	The argument count
argv	The argument values

Definition at line 43 of file runner.cpp.

```
44 {
45     this->argc = argc;
46     this->argv = argv;
47     this->semaphore = new Semaphore(2);
48     this->semaphore->get()->set(1);
49     this->bestChunk = NULL;
50     this->printed = true;
51     this->threadsDone = false;
52     this->candidates = 0;
```

```
53 }
```

Runner::~Runner ()

Runner::~Runner Destructor.

Definition at line 58 of file runner.cpp.

```
58              {
59     delete chunker;
60 
61     delete bestChunk;
62     delete semaphore;
63     isSet = false;
64 }
```

Member Function Documentation

void * Runner::analyze (void * *input*) [static], [private]

Runner::analyze Static method used for the processing of chunks of data.

Parameters:

input	A pointer to a struct. Currently just stubbed out for future development.

Returns:

0 if success

Definition at line 202 of file runner.cpp.

References getRunner(), and Chunk::process().

Referenced by launchThreads().

```
202                              {
203     (void) input; /*< input is not used, because we can get the object with Runner::getRunner() */
204 
205     Runner *self = Runner::getRunner();
206     Chunk *chunk;
207 
208     // While there are more chunks, get one and process it
209     chunk = self->chunker->getChunk();
210     while (chunk != NULL){
211         chunk->process();
212         chunk = self->chunker->getChunk();
213     };
214     delete chunk; // Delete the last NULL chunk
215 
216     pthread_exit(0);
217 }
```

void * Runner::displayResults (void * *input*) [static], [private]

Runner::displayResults Static method used for the displaying of results.

Parameters:

input	A pointer to a struct. Currently just stubbed out for future development.

Returns:

0 if success

< input is not used, because we can get the object with Runner::getRunner()

Definition at line 178 of file runner.cpp.

References getRunner().

Referenced by launchThreads().

```
178                                   {
179     (void) input;
181     Runner *self = Runner::getRunner();
182
183     // While the processing threads are still working print the best chunk so far
184     while (!self->threadsDone) {
185         self->semaphore->wait(BESTCHUNK);
186         if (!self->printed) {
187             self->bestChunk->printData();
188             self->bestChunk->printBlock();
189             self->candidates += 1;
190         }
191         self->printed = true;
192         self->semaphore->signal(BESTCHUNK);
193     }
194     pthread_exit(0);
195 }
```

Chunk * Runner::getBestChunk ()

Runner::getBestChunk Getter for bestChunk.

Returns:

This

Definition at line 280 of file runner.cpp.

Referenced by Chunk::process().

```
280                        {
281     return bestChunk;
282 }
```

Runner * Runner::getRunner (int *argc*, char ** *argv*) [static]

Runner::getRunner Gets the Singleton (or creates if not created)

Parameters:

argc	The argument count
argv	The argument values

Returns:

The Singleton instance

Definition at line 72 of file runner.cpp.

```
73 {
74     if(!isSet){
75         runner = new Runner(argc, argv);
76         isSet = true;
77     }
78
79     return runner;
80 }
```

Runner * Runner::getRunner () [static]

Runner::getRunner Gets the Singleton.

Returns:

The Singleton instance

Definition at line 86 of file runner.cpp.

Referenced by analyze(), displayResults(), main(), and Chunk::process().

```
{
    if(!isSet){
        return NULL;
    }

    return runner;
}
```

Runner * Runner::launchThreads () `[private]`

Runner::launchThreads Launches the threads to process the input file. If n threads are called from the command line, the program launches n + 1 threads. n threads are used for processing, and one is used for displaying the results.

Returns:

this

Definition at line 132 of file runner.cpp.

References analyze(), displayResults(), and str_thdata::thread_no.

```
{
    pthread_t *threads;
    thdata *threadData;

    // Create arrays for threads
    int total = this->nThreads + 1;
    int last = this->nThreads;
    threads = (pthread_t*) calloc(sizeof(pthread_t),  sizeof(pthread_t) * total);
    threadData = (thdata*) calloc(sizeof(thdata), sizeof(thdata) * total);

    // Launch processing threads
    for (int i = 0; i < nThreads; i++){
        threadData[i].thread_no = i;
        if( pthread_create (&threads[i], NULL, &Runner::analyze, (void*) &threadData[i]) < 0){
            perror("Error creating threads:");
        }
    }

    // Launch printing thread
    threadData[last].thread_no = last;
    if( pthread_create (&threads[last], NULL, &Runner::displayResults, (void*) &threadData[last]) < 0){
        perror("Error creating threads:");
    }

    // Join the processing threads
    for (int i = 0; i < nThreads; i++){
        pthread_join(threads[i], NULL);
    }

    // Signal that the processing threads are done and join the printing thread
    this->threadsDone = true;
    pthread_join(threads[last], NULL);

    // Clean up
    free(threads);
```

```
168     free(threadData);
169
170     return this;
171 }
```

Runner * Runner::processArgs () [private]

Runner::processArgs Processes the command line arguments.

Returns:

this

Todo:

Need to get the order for the polynomial

Definition at line 224 of file runner.cpp.

```
224                         {
225     if (argc != 5){
226         cout << "Usage: " << argv[0] << " inputDataFile chunkSize nThreads polynomialOrder" << endl;
227         exit (-1);
228     }
229
230     // argv[0] is name of executable
231     // argv[1] is filename
232     // argv[2] is chunk size
233     // argv[3] is the number of processing threads
234     // argv[4] is the order of the polynomial
235     chunker = new Chunker(argv[1], atoi(argv[2]), atoi(argv[4]));
236     this->nThreads = atoi(argv[3]);
237
238     if (this->nThreads <= 0){
239         this->nThreads = 1;
240     }
241
242     return this;
243 }
```

Runner * Runner::run ()

Runner::run Runs the runner.

Returns:

this

Todo:

When the program finishes it should report final statistics, such as the number of threads used, the number of candidates found, the time it took, and also report the best candidates found, with their criteria and plots.

Definition at line 99 of file runner.cpp.

Referenced by main().

```
99                    {
100     struct rusage usage;
101     processArgs();
102     writeAuthorInformation();
103     launchThreads();
104
105     // Print out the results
106     this->bestChunk->printData();
107     this->bestChunk->printBlock();
108
```

```
109     getrusage(RUSAGE_SELF, &usage);
115     cout <<
"##################################################################################" <<
endl;
116     cout << "Processing completed for " << argv[1] << endl;
117     cout << "User time: " << usage.ru_utime.tv_sec << "." <<
usage.ru_utime.tv_usec << endl;
118     cout << "System time: " << usage.ru_stime.tv_sec << "." <<
usage.ru_stime.tv_usec << endl;
119     cout << "Processing used " << nThreads << " processing threads and 1 printing
thread." << endl;
120     cout << this->candidates << " candidates were found." << endl;
121     cout << "The best candidate is plotted above." << endl;
122     cout <<
"##################################################################################" <<
endl;
123     return this;
124 }
```

Runner * Runner::setBestChunk (Chunk * *chunk*)

Runner::setBestChunk Tries to set a chunk as the best chunk.

Parameters:

chunk	The chunk to be set

Returns:

this

Definition at line 250 of file runner.cpp.

References Chunk::getError().

Referenced by Chunk::process().

```
250                                           {
251 
252     // Wait for the current best chunk to be printed
253     semaphore->wait(PRINTED);
254     while (!printed) {
255         semaphore->signal(PRINTED);
256         usleep(1);
257         semaphore->wait(PRINTED);
258     }
259 
260     // If this chunk is still the best chunk, delete the last chunk and set this
on
261     semaphore->wait(BESTCHUNK);
262     if (this->bestChunk == NULL || this->bestChunk->getError() >
chunk->getError()){
263         delete this->bestChunk;
264         this->bestChunk = chunk;
265         this->printed = false;
266     }
267     // Otherwise, there is a new better, discard this one
268     else {
269         delete chunk;
270     }
271     semaphore->signal(BESTCHUNK);
272     semaphore->signal(PRINTED);
273     return this;
274 }
```

Runner * Runner::writeAuthorInformation () `[private]`

Master::writeAuthorInformation Writes the author information to stdout.

Definition at line 287 of file runner.cpp.

```
287                                    {
288     cout << argv[0] << " by William Montgomery (wmontg2)" << endl;
289     cout << endl;
290     return this;
291 }
```

Member Data Documentation

int Runner::argc [private]

The argument count

Definition at line 28 of file runner.h.

char** Runner::argv [private]

The argument value

Definition at line 29 of file runner.h.

Chunk* Runner::bestChunk [private]

Pointer to the best chunk found so far

Definition at line 37 of file runner.h.

int Runner::candidates [private]

The number of candidates

Definition at line 33 of file runner.h.

Chunker* Runner::chunker [private]

The Chunker

Definition at line 35 of file runner.h.

int Runner::chunkSize [private]

The size of the chunks

Definition at line 31 of file runner.h.

bool Runner::isSet = false [static], [private]

True if the runner object has been created.

Definition at line 40 of file runner.h.

int Runner::nThreads [private]

The number of processing threads

Definition at line 30 of file runner.h.

int Runner::polynomialOrder [private]

The order of the polynomial

Definition at line 32 of file runner.h.

bool Runner::printed [private]

Boolean of whether the best chunk has been printed

Definition at line 38 of file runner.h.

Runner * Runner::runner = NULL [static], [private]

The Singleton

Definition at line 41 of file runner.h.

Semaphore* Runner::semaphore [private]

The Semaphore

Definition at line 36 of file runner.h.

bool Runner::threadsDone [private]

Boolean of whether the processing threads have completed

Definition at line 39 of file runner.h.

The documentation for this class was generated from the following files:

orderSearcher/runner.h
orderSearcher/runner.cpp

Semaphore Class Reference

The Semaphore class Encapsulates the semaphore operations.

`#include <semaphore.h>`

Public Member Functions

Semaphore (int)

Semaphore::Semaphore Constructor.

~Semaphore ()

Semaphore * get ()

Semaphore::get Uses a syscall to get a semaphore.

Semaphore * set (int)

Semaphore::set Sets all semaphores to a value.

Semaphore * wait (int)

Semaphore::wait Waits for a semaphore.

Semaphore * signal (int)

Semaphore::signal Signal a semaphore.

int getID ()

Semaphore::getID Getter for id.

Semaphore * setID (int)

Semaphore::setID Setter for id.

Private Member Functions

Semaphore * remove ()

Semaphore::remove Removes a semaphore.

Private Attributes

int id

int numSemaphores

Detailed Description

The Semaphore class Encapsulates the semaphore operations.

Definition at line 19 of file semaphore.h.

Constructor & Destructor Documentation

Semaphore::Semaphore (int *numSemaphores*)

Semaphore::Semaphore Constructor.

Parameters:

numSemaphores	The number of semaphores to create in the semaphore group.

Definition at line 25 of file semaphore.cpp.

```
25                                        {
26      id = -1;
27      this->numSemaphores = numSemaphores;
28 }
```

Semaphore::~Semaphore ()

Definition at line 30 of file semaphore.cpp.

```
30                       {
31      remove();
32 }
```

Member Function Documentation

Semaphore * Semaphore::get ()

Semaphore::get Uses a syscall to get a semaphore.

Returns:

This

Definition at line 38 of file semaphore.cpp.

Referenced by Chunker::Chunker().

```
38                           {
39      id = semget(IPC_PRIVATE, numSemaphores, IPC_CREAT | 0600);
40      if (id < 0){
41          perror("semget");
42      }
43      return this;
44 }
```

int Semaphore::getID ()

Semaphore::getID Getter for id.

Returns:

id

Definition at line 119 of file semaphore.cpp.

```
119                        {
120      return id;
121 }
```

Semaphore * Semaphore::remove () [private]

Semaphore::remove Removes a semaphore.

Returns:

This

Definition at line 105 of file semaphore.cpp.

```
105                              {
106      int result = semctl(id, 0, IPC_RMID, 0);
```

```
107     if (result < 0){
108         perror("semctl (remove)");
109     }
110     id = -1;
111
112     return this;
113 }
```

Semaphore * Semaphore::set (int *value*)

Semaphore::set Sets all semaphores to a value.

Parameters:

value	The value to set

Returns:

This

Definition at line 51 of file semaphore.cpp.

Referenced by Chunker::Chunker().

```
51                                     {
52     int result;
53     for (int i = 0; i < numSemaphores; i++){
54         result = semctl(id, i, SETVAL, value);
55         if (result < 0) {
56             perror("semctl");
57         }
58     }
59
60     return this;
61 }
```

Semaphore * Semaphore::setID (int *id*)

Semaphore::setID Setter for id.

Parameters:

id	id

Returns:

This

Definition at line 128 of file semaphore.cpp.

```
128                                    {
129     this->id = id;
130
131     return this;
132 }
```

Semaphore * Semaphore::signal (int *number*)

Semaphore::signal Signal a semaphore.

Parameters:

number	The number of the semaphore

Returns:

This

Definition at line 87 of file semaphore.cpp.

Referenced by Chunker::getChunk().

```
87                                {
88      struct sembuf sem_op;
89      sem_op.sem_num = number;
90      sem_op.sem_op = 1;
91      sem_op.sem_flg = 0;
92
93      int result = semop(id, &sem_op, 1);
94      if (result < 0) {
95          perror("semop (signal)");
96      }
97
98      return this;
99 }
```

Semaphore * Semaphore::wait (int *number*)

Semaphore::wait Waits for a semaphore.

Parameters:

number	The number of the semaphore

Returns:

This

Definition at line 68 of file semaphore.cpp.

Referenced by Chunker::getChunk().

```
68                              {
69      struct sembuf sem_op;
70      sem_op.sem_num = number;
71      sem_op.sem_op = -1;
72      sem_op.sem_flg = 0;
73
74      int result = semop(id, &sem_op, 1);
75      if (result < 0){
76          perror("semop (wait)");
77      }
78
79      return this;
80 }
```

Member Data Documentation

int Semaphore::id `[private]`

The id of the semaphore set

Definition at line 21 of file semaphore.h.

int Semaphore::numSemaphores `[private]`

The number of semaphores in the set.

Definition at line 22 of file semaphore.h.

The documentation for this class was generated from the following files:

orderSearcher/semaphore.h
orderSearcher/semaphore.cpp

str_thdata Struct Reference

Struct used to pass information specific to a single thread. Currently is passed, but is ignrored.

Public Attributes

int thread_no

Detailed Description

Struct used to pass information specific to a single thread. Currently is passed, but is ignrored.
Definition at line 29 of file runner.cpp.

Member Data Documentation

int str_thdata::thread_no

Definition at line 31 of file runner.cpp.
Referenced by Runner::launchThreads().

The documentation for this struct was generated from the following file:

orderSearcher/runner.cpp

File Documentation

orderSearcher/chunk.cpp File Reference

Defines the Chunk class.

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <cmath>
#include <limits.h>
#include "chunk.h"
#include "runner.h"
```

Detailed Description

Defines the Chunk class.

Definition in file chunk.cpp.

orderSearcher/chunk.h File Reference

Defines the Chunk interface.

```
#include <limits.h>
```

Classes

class Chunk

The Chunk class Encapsulates a chunk of data.

Detailed Description

Defines the Chunk interface.

Definition in file chunk.h.

orderSearcher/chunker.cpp File Reference

Defines the implementation of the Chunker class.

```
#include <iostream>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
#include "chunker.h"
```

Detailed Description

Defines the implementation of the Chunker class.

Definition in file chunker.cpp.

orderSearcher/chunker.h File Reference

Defines the Chunker interface.

```
#include <fstream>
#include "semaphore.h"
#include "chunk.h"
```

Classes

class Chunker

The Chunker class Encapsulates the process of breaking down a file into chunks.

Detailed Description

Defines the Chunker interface.

Definition in file chunker.h.

orderSearcher/main.cpp File Reference

Main.

```
#include <iostream>
#include "runner.h"
```

Functions

int main (int argc, char **argv)

main

Detailed Description

Main.

Definition in file main.cpp.

Function Documentation

int main (int *argc*, char ** *argv*)

main

Parameters:

argc	The argument count
argv	The argument values

Returns:

0 if success

Definition at line 24 of file main.cpp.

References Runner::getRunner(), and Runner::run().

```
{
    Runner *runner = Runner::getRunner(argc, argv);
    runner->run();
    delete (runner);
    return 0;
}
```

orderSearcher/mainpage.dox File Reference

orderSearcher/runner.cpp File Reference

Defines the implementation of the Runner class.

```
#include <stdlib.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/resource.h>
#include "runner.h"
```

Classes

struct str_thdata

Struct used to pass information specific to a single thread. Currently is passed, but is ignrored. Typedefs

typedef struct str_thdata thdata

Struct used to pass information specific to a single thread. Currently is passed, but is ignrored.

Detailed Description

Defines the implementation of the Runner class.

Definition in file runner.cpp.

Typedef Documentation

typedef struct str_thdata thdata

Struct used to pass information specific to a single thread. Currently is passed, but is ignrored.

orderSearcher/runner.h File Reference

Defines the Runner interface.

```
#include <iostream>
#include <vector>
#include "chunker.h"
```

Classes

class Runner

The Runner class Singleton class that controls the flow of the program.

Detailed Description

Defines the Runner interface.

Definition in file runner.h.

orderSearcher/semaphore.cpp File Reference

Defines the Semaphore class.

```
#include <sys/sem.h>
#include <stdio.h>
#include <iostream>
#include "semaphore.h"
```

Detailed Description

Defines the Semaphore class.

Definition in file semaphore.cpp.

orderSearcher/semaphore.h File Reference

Defines the Semaphore interface.

Classes

class Semaphore

The Semaphore class Encapsulates the semaphore operations.

Detailed Description

Defines the Semaphore interface.

Definition in file semaphore.h.