**1 Introduction**

The application chosen to be parallelised was the Bioinformatics – Genome Similarity Using Frequency Vectors. This application’s purpose is to compare different types of bacteria to each other by calculating the correlation between two different bacteria.

**1.1 Software Architecture**

Figure 1 below shows the basic process of Bioinformatics – Genome Similarity Using Frequency Vectors’ architecture. The detailed software architecture can be found in full in section 7.3 of the Appendix of this report.

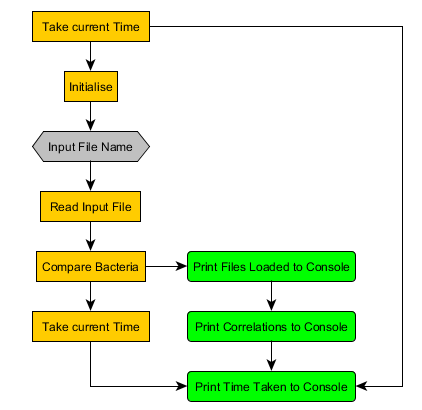


Figure 1: Basic View of Software Architecture

**1.1.1 Time**

The application takes the time before and after the application and the difference is calculated. This is printed to the console to notify the user how long the program took to complete the Bioinformatics – Genome Similarity Using Frequency Vectors application.

**1.1.2 Initialise**

This method initialises long variables M1 and M, based on the predefined value for AA\_NUMBER and LEN, for the class Bacteria.

**1.1.3 Input Filename and Read Input File**

The program takes a users’ input argument. This argument is the filename for a text file. This text file contains the number of bacteria and the names of the associated bacteria files. The first line contains the number of bacteria. From the second line down, these lines contain the name of each \*.faa file. The extension \*.faa is used by the NCBI for FASTA animo acids [1]. In this application, forty-one protein FASTA files are being used. Figure 2, on the next page, shows the detail view software architecture for this function.

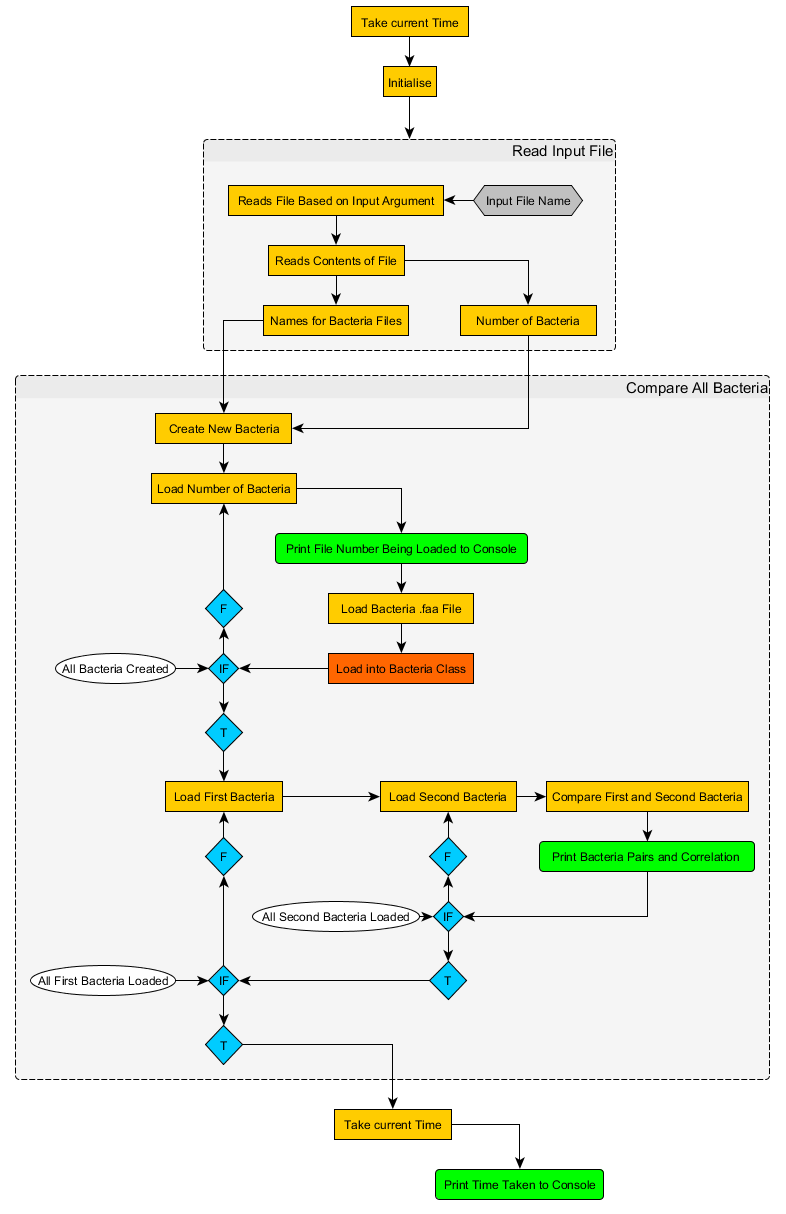


Figure 2: Read Input File Software Architecture

**1.1.4 Compare All Bacteria**

This component of the application is where most of the runtime and computational work in located. This function does three things. The first creates the number of classes of bacteria based on the number of bacteria to be read. The second component is made of two parts. The first part is printing to the console which file is being loaded. The second part is loading the \*.faa files and loading the data into their respective bacteria class. The third calls for the compare bacteria function which calculates the correlation between two pairs of different bacteria and then printing the result to the console. Figure 3 on the next page shows a detailed view of the CompareAllBacteria function.

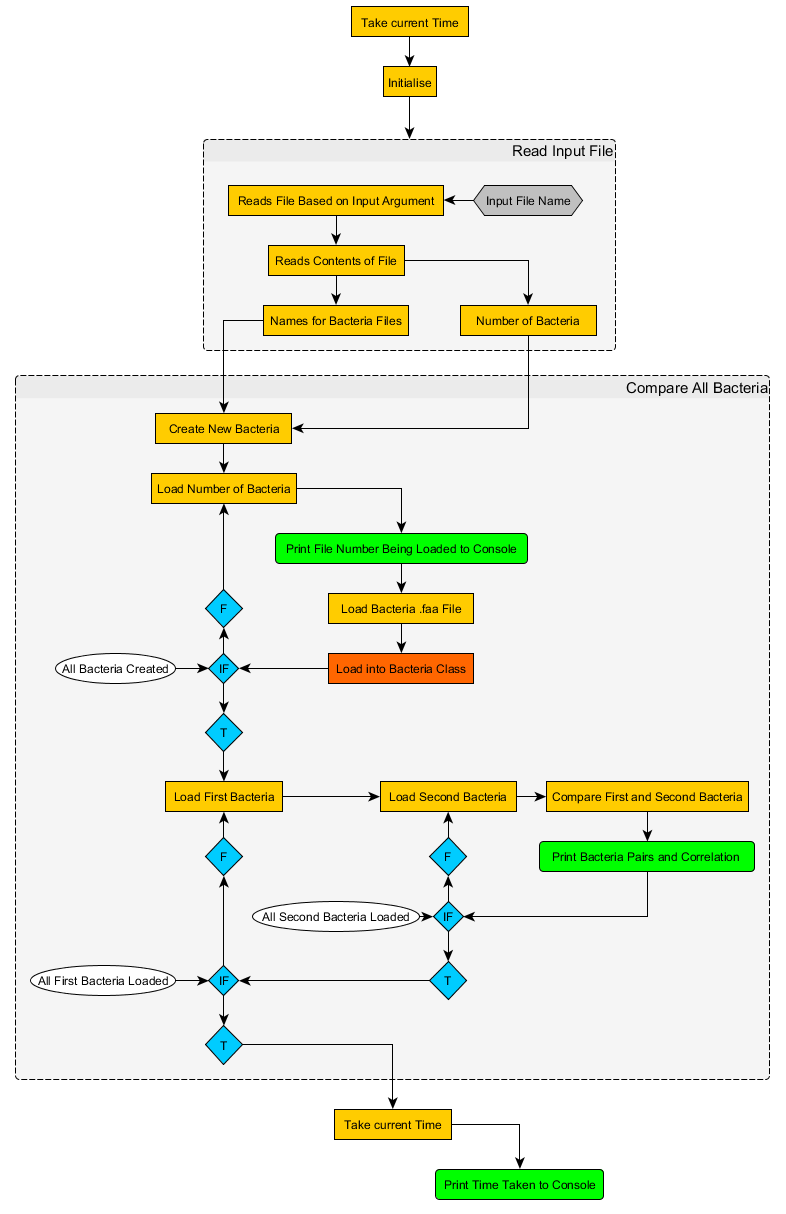


Figure 3: Compare All Bacteria Software Architecture

**1.1.5 Bacteria Class**

The bacteria files are opened and vectors are initialised. The bacteria class holds data based on the contents in the \*.faa files. The class obtains the necessary characters from the respective \*.faa file. Figure 4, shown below, shows some of the contents of file AcMNPV.faa.

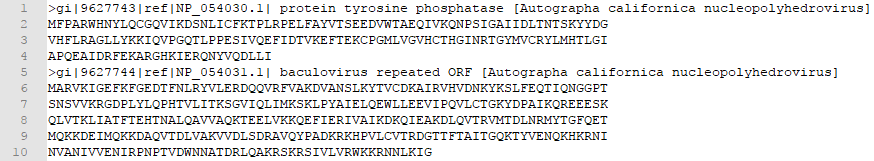


Figure 4: AcMNPV Animo Acid File Content Example

The first stage of the bacteria class is obtaining the data in the respective \*.faa file. In this process there are three parts. The first part is whether the end of the file has been met. If so, the search to get the characters for the bacteria class ends. If not, then the second part comes into effect. If the character identified is a ‘>’, then that line is skipped. This line is skipped because the line does not contain the data required, rather just the name and reference of the amino acid. This can be seen in lines one and five in Figure 4 above. When the line is skipped the characters are read initialise buffer is called. The last part is whether a ‘\n’, or space, character is identified. If not, the process continues to get a character. If a ‘\n’ is identified, then continue buffer is called. This section of the bacteria class can be seen below in Figure 5

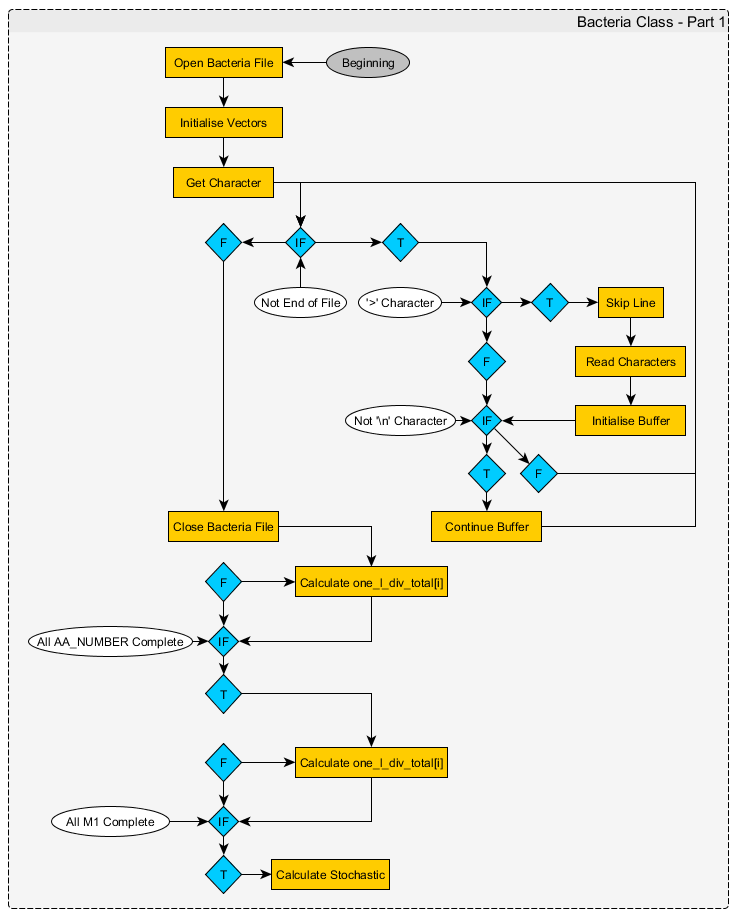


Figure 5: Get Data from \*.faa File

When the end of the file has been confirmed the respective file is closed. From here the calculations are performed. Figure 6, shown below, shows the software architecture for this stage up to the point where stochastic is calculated.

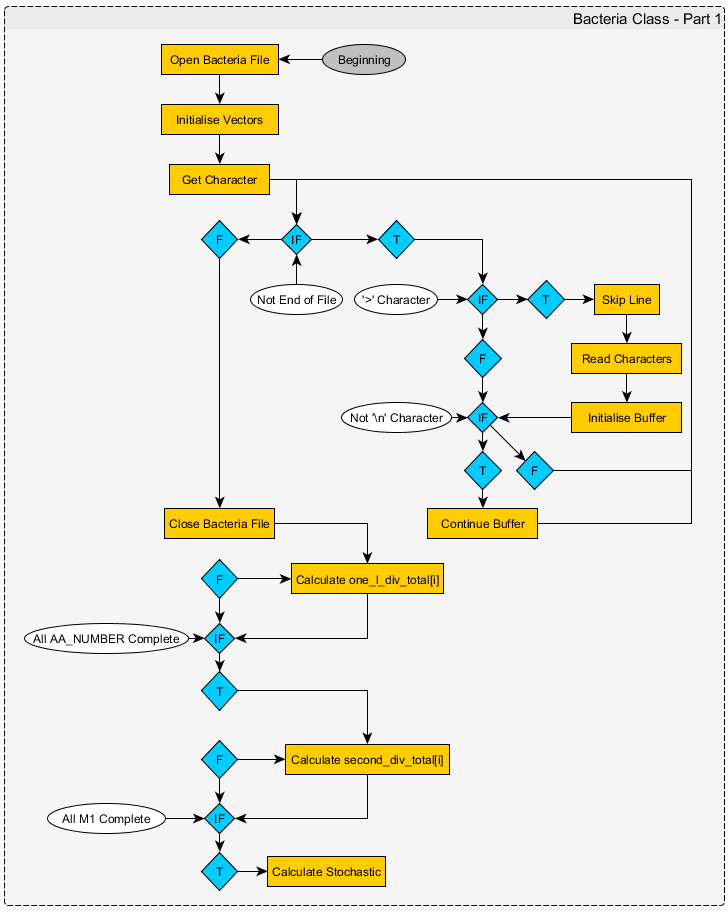


Figure 6: Calculate Variables in Bacteria Class – Part 1

The bacteria class proceeds to calculate other variables which can be seen below in Figure 7.

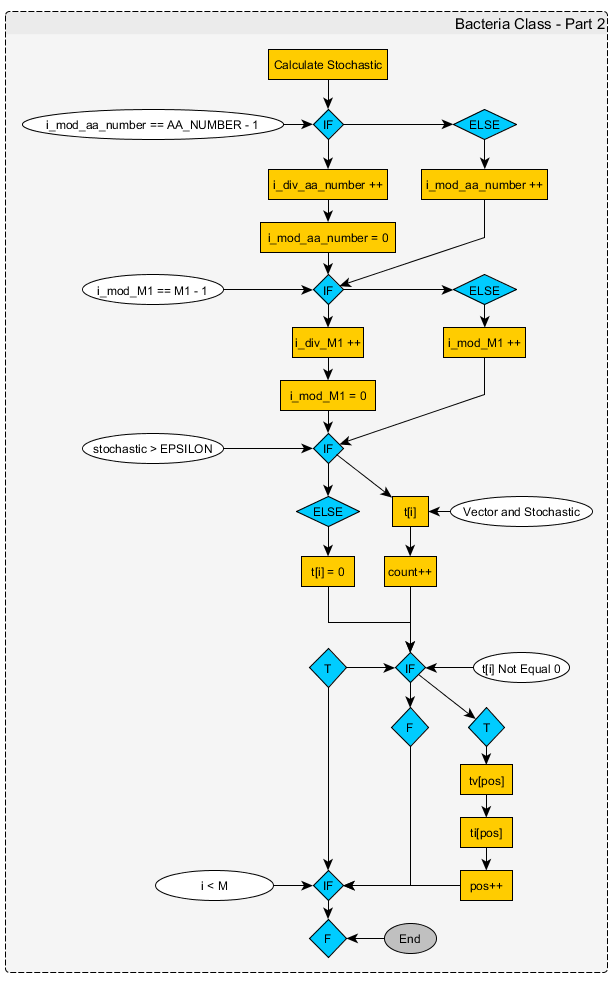


Figure 7: Other Values Calculated in Bacteria Class

The main importance of the bacteria class is the calculated values for tv[] and ti[]. These variables are used when comparing two different bacteria. Figure 8 below shows the software architecture for the rest of the bacteria class. The full software architecture diagram for the bacteria class can be found in the appendices. Bacteria Class – Part 1 software architecture can be found in section 7.4 of this report. Bacteria Class – Part 2 software architecture can be found in section 7.5 of this report.

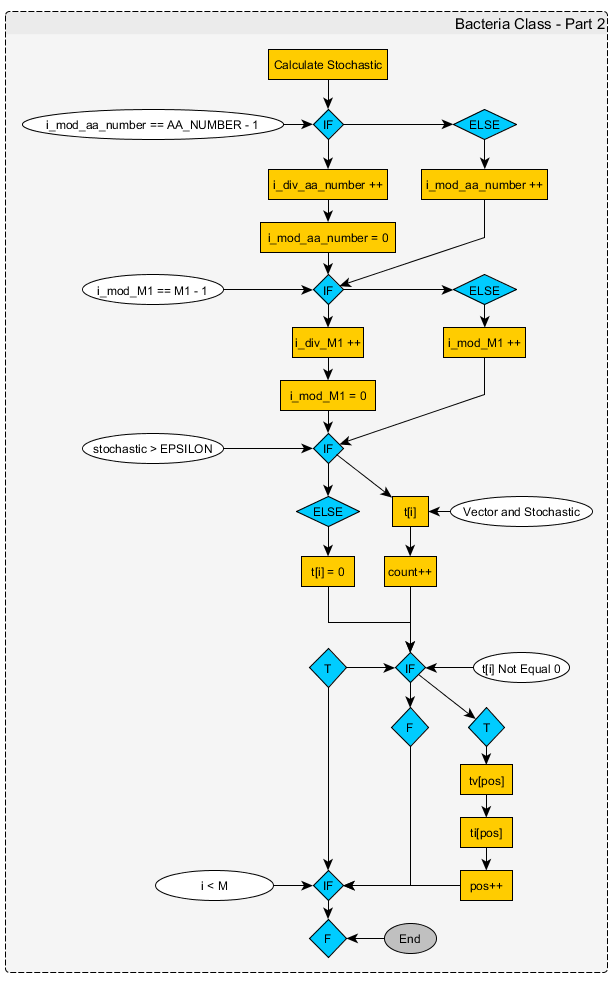


Figure 8: Bacteria Class Ending

**1.1.4 Compare Bacteria**

The compare bacteria function takes two bacteria as inputs and proceeds to calculate the correlation between the two bacteria based on their vectors, ti and tv variables. In this application, the two bacteria that are taken are two different bacteria. This occurs in the CompareAllBacteria function which includes a nested loop to ensure that every possible unique pair of bacteria are compared. The detail software architecture for the CompareBacteria function can be seen in Figure 9 on the next page and in section 7.6 of the appendices in this report.

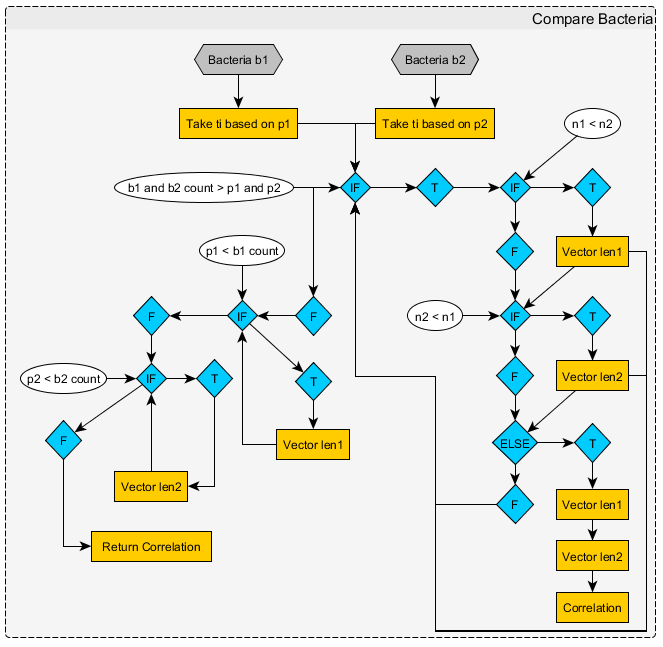


Figure 9: Detail View of Compare Bacteria

**2 Bioinformatics – Genome Similarity Using Frequency Vectors**

This section of the report discusses what parts of the original code, that were analysed, that could allow the application to perform faster while providing the same results. This includes sections of code that caused bottlenecks, performance issues, what could be parallelised and if the application can be scalable.

**2.1 Analysis**

ddfd

Your analysis of potential parallelism within the application. This might include identification of existing loops or control flow constructs where parallelism might be found. Which of these is likely to be of sufficient granularity to be worth exploiting? Is it scalable parallelism? A discussion of changes required to expose parallelism, such as replacing algorithms or code restructuring transformations.

Criteria: Demonstrates a deep understanding of the original application, its structure and performance issues/bottlenecks. (Must include identification and discussion of data and control dependencies and detailed before and after detailed profiling results).

about program

Loads all of the bacteria in

Compares 1 bacteria at a time to the rest. comparison only requires 1 calculation, x to y equals y to x Therefore increases in bacteria being compared faster with time

Therefore cannot break computing time into equal chunks

Even if did so, one of the bacteria files are huge, 10 times bigger than some others. therefore if broken into equal computation, some threads may take longer as the bacteria information is much bigger

Went through all for loops within the program

2 of the for loops took the majority of time to compute

Recommended to be parallelized

As it is 2 areas needing threads, threadpool could be used

While loop also required a lot of computing

Too complex to break apart

**2.2 Parallelisation Process**

How did you map computation and/or data to processors? Which parallelism abstractions or programming language constructs did you use to perform synchronization?

**2.3 Compilers, Software and Techniques**

**2.3.1 Software**

The integrated development environment (IDE) used when analysing the application and its code was Microsoft’s Visual Studio 2017. This IDE’s diagnostic tools were used to identify which section of code took the most amount of time to process. Furthermore, the diagnostic tools show the memory and CPU usage that is being used by the application is real-time. This assisted in analysing for-loop parallelisation as to how much memory was being used and the difference between the original and parallelised version of code’s CPU usage.

OpenMP 2.0 is included as part of Visual Studio 2017. This was enabled and chosen for the following reasons. Its simplicity, ease of use and integration in the IDE makes it a wonderful choice when parallelising for-loops and multi-threading other components of the application. The second reason is that I have not used the OpenMP API before. The third reason, OpenMP performed faster results than using POSIX threads.

**2.3.2 Compiler**

The compiler used was the default compiler that Microsoft has included in Visual Studio 2017. This version of the compiler was Visual C++ 14.1.

**2.3.2 Compiler**

OpenMP’s sections, parallel and for techniques were used make the program complete within a faster time frame during testing. Sections to perform two different tasks at the same time or splitting sections of work. Parallel and for were used to parallelise for-loops. Other techniques attempted were removing a nested for-loop for a while loop. Utilising an array to store the correlation variables before printing to the console in order.

**3 Code Implemented**

An explanation of the code that you added or modified to parallelize the application (including source code line count)

**4 Results**

Timing and profiling results, both before and after parallelization and a speedup graph

Criteria: Obtained very close to the best possible performance improvement for the application (must be more than 4 cores for excellent).

(Must include a correctly constructed speed-up graph).

**5 Reflection**

Reflect on your outcome – What have you learnt? How successful was your attempt? Do you think you’ve done as well as is possible? What might you have done differently?

**5.1 Issues Encounter**

The story of how you overcame performance problems/barriers (e.g. load imbalance, memory contention, granularity, data dependencies, etc) to improving parallel performance

Criteria: Demonstrated great skill and effort to achieve this outcome and overcome significant barriers to improved performance.

(Include interesting before and after code snippets)

**6 References**

[1] "Difference between .fas and .faa files? - SEQanswers", *Seqanswers.com*, 2017. [Online]. Available: <http://seqanswers.com/forums/showthread.php?t=21264>. [Accessed: 24- Oct- 2017].

**7 Appendix**

**7.1 Original Code**

The original code for the Bioinformatics – Genome similarity using Frequency Vectors (C++) application was provided by Dr Wayne Kelly on the CAB401 High Performance and Parallel Computing QUT Blackboard page. A copy of the original code can be seen below. All bacteria files and the Visual Studio 2017 project can be found at the link here <https://github.com/Starwolf-001/CAB401/tree/master/Bioinformatics_c_assignment>.

1 #include <stdio.h>

2 #include <string.h>

3 #include <time.h>

4 #include <math.h>

5 #include <iostream>

6

7 int number\_bacteria;

8 char\*\* bacteria\_name;

9 long M, M1, M2;

10 short code[27] = { 0, 2, 1, 2, 3, 4, 5, 6, 7, -1, 8, 9, 10, 11, -1, 12, 11 13, 14, 15, 16, 1, 17, 18, 5, 19, 3 };

12 #define encode(ch) code[ch-'A']

13 #define LEN 6

14 #define AA\_NUMBER 20

15 #define EPSILON 1e-010

16

17 void Init()

18 {

19 M2 = 1;

20 for (int i = 0; i<LEN - 2; i++) // M2 = AA\_NUMBER ^ (LEN-2);

21 M2 \*= AA\_NUMBER;

22 M1 = M2 \* AA\_NUMBER; // M1 = AA\_NUMBER ^ (LEN-1);

23 M = M1 \*AA\_NUMBER; // M = AA\_NUMBER ^ (LEN);

24 }

25

26 class Bacteria

27 {

28 private:

29 long\* vector;

30 long\* second;

31 long one\_l[AA\_NUMBER];

32 long indexs;

33 long total;

34 long total\_l;

35 long complement;

36

37 void InitVectors()

38 {

39 vector = new long[M];

40 second = new long[M1];

41 memset(vector, 0, M \* sizeof(long));

42 memset(second, 0, M1 \* sizeof(long));

43 memset(one\_l, 0, AA\_NUMBER \* sizeof(long));

44 total = 0;

45 total\_l = 0;

46 complement = 0;

47 }

48

49 void init\_buffer(char\* buffer)

50 {

51 complement++;

52 indexs = 0;

53 for (int i = 0; i<LEN - 1; i++)

54 {

55 short enc = encode(buffer[i]);

56 one\_l[enc]++;

57 total\_l++;

58 indexs = indexs \* AA\_NUMBER + enc;

59 }

60 second[indexs]++;

61 }

62

63 void cont\_buffer(char ch)

64 {

65 short enc = encode(ch);

66 one\_l[enc]++;

67 total\_l++;

68 long index = indexs \* AA\_NUMBER + enc;

69 vector[index]++;

70 total++;

71 indexs = (indexs % M2) \* AA\_NUMBER + enc;

72 second[indexs]++;

73 }

74

75 public:

76 long count;

77 double\* tv;

78 long \*ti;

79

80 Bacteria(char\* filename)

81 {

82 FILE \* bacteria\_file = fopen(filename, "r");

83 InitVectors();

84

85 char ch;

86 while ((ch = fgetc(bacteria\_file)) != EOF)

87 {

88 if (ch == '>')

89 {

90 // skip rest of line

91 while (fgetc(bacteria\_file) != '\n');

92 char buffer[LEN - 1];

93 fread(buffer, sizeof(char), LEN - 1, bacteria\_file);

94 init\_buffer(buffer);

95 }

96 else if (ch != '\n')

97 cont\_buffer(ch);

98 }

99

100 long total\_plus\_complement = total + complement;

101 double total\_div\_2 = total \* 0.5;

102 int i\_mod\_aa\_number = 0;

103 int i\_div\_aa\_number = 0;

104 long i\_mod\_M1 = 0;

105 long i\_div\_M1 = 0;

106

107 double one\_l\_div\_total[AA\_NUMBER];

108 for (int i = 0; i<AA\_NUMBER; i++)

109 one\_l\_div\_total[i] = (double)one\_l[i] / total\_l;

110

111 double\* second\_div\_total = new double[M1];

112 for (int i = 0; i<M1; i++)

113 second\_div\_total[i] = (double)second[i] / total\_plus\_complement;

114 count = 0;

115 double\* t = new double[M];

116

117 for (long i = 0; i<M; i++)

118 {

119 double p1 = second\_div\_total[i\_div\_aa\_number];

120 double p2 = one\_l\_div\_total[i\_mod\_aa\_number];

121 double p3 = second\_div\_total[i\_mod\_M1];

122 double p4 = one\_l\_div\_total[i\_div\_M1];

123 double stochastic = (p1 \* p2 + p3 \* p4) \* total\_div\_2;

124

125 if (i\_mod\_aa\_number == AA\_NUMBER - 1)

126 {

127 i\_mod\_aa\_number = 0;

128 i\_div\_aa\_number++;

129 }

130 else

131 i\_mod\_aa\_number++;

132

133 if (i\_mod\_M1 == M1 - 1)

134 {

135 i\_mod\_M1 = 0;

136 i\_div\_M1++;

137 }

138 else

139 i\_mod\_M1++;

140

141 if (stochastic > EPSILON)

142 {

143 t[i] = (vector[i] - stochastic) / stochastic;

144 count++;

145 }

146 else

147 t[i] = 0;

148 }

149

150 delete second\_div\_total;

151 delete vector;

152 delete second;

153

154 tv = new double[count];

155 ti = new long[count];

156

157 int pos = 0;

158 for (long i = 0; i<M; i++)

159 {

160 if (t[i] != 0)

161 {

162 tv[pos] = t[i];

163 ti[pos] = i;

164 pos++;

165 }

166 }

167 delete t;

168

169 fclose(bacteria\_file);

170 }

171 };

172

173 void ReadInputFile(char\* input\_name)

174 {

175 FILE\* input\_file = fopen(input\_name, "r");

176 fscanf(input\_file, "%d", &number\_bacteria);

177 bacteria\_name = new char\*[number\_bacteria];

178

179 for (long i = 0; i<number\_bacteria; i++)

180 {

181 bacteria\_name[i] = new char[20];

182 fscanf(input\_file, "%s", bacteria\_name[i]);

183 strcat(bacteria\_name[i], ".faa");

184 }

185 fclose(input\_file);

186 }

187

188 double CompareBacteria(Bacteria\* b1, Bacteria\* b2)

189 {

190 double correlation = 0;

191 double vector\_len1 = 0;

192 double vector\_len2 = 0;

193 long p1 = 0;

194 long p2 = 0;

195 while (p1 < b1->count && p2 < b2->count)

196 {

197 long n1 = b1->ti[p1];

198 long n2 = b2->ti[p2];

199 if (n1 < n2)

200 {

201 double t1 = b1->tv[p1];

202 vector\_len1 += (t1 \* t1);

203 p1++;

204 }

205 else if (n2 < n1)

206 {

207 double t2 = b2->tv[p2];

208 p2++;

209 vector\_len2 += (t2 \* t2);

210 }

211 else

212 {

213 double t1 = b1->tv[p1++];

214 double t2 = b2->tv[p2++];

215 vector\_len1 += (t1 \* t1);

216 vector\_len2 += (t2 \* t2);

217 correlation += t1 \* t2;

218 }

219 }

220 while (p1 < b1->count)

221 {

222 long n1 = b1->ti[p1];

223 double t1 = b1->tv[p1++];

224 vector\_len1 += (t1 \* t1);

225 }

226 while (p2 < b2->count)

227 {

228 long n2 = b2->ti[p2];

229 double t2 = b2->tv[p2++];

230 vector\_len2 += (t2 \* t2);

231 }

232

233 return correlation / (sqrt(vector\_len1) \* sqrt(vector\_len2));

234 }

235

236 void CompareAllBacteria()

237 {

238 Bacteria\*\* b = new Bacteria\*[number\_bacteria];

239 for (int i = 0; i<number\_bacteria; i++)

240 {

241 printf("load %d of %d\n", i + 1, number\_bacteria);

242 b[i] = new Bacteria(bacteria\_name[i]);

243 }

244

245 for (int i = 0; i<number\_bacteria - 1; i++)

246 for (int j = i + 1; j<number\_bacteria; j++)

247 {

248 printf("%2d %2d -> ", i, j);

249 double correlation = CompareBacteria(b[i], b[j]);

250 printf("%.20lf\n", correlation);

251 }

252 }

253

254 int main(int argc, char \* argv[])

255 {

256 time\_t t1 = time(NULL);

257

258 Init();

259 ReadInputFile(argv[1]);

260 CompareAllBacteria();

261

262 time\_t t2 = time(NULL);

263 printf("time elapsed: %d seconds\n", t2 - t1);

264

265 system("pause");

266

267 return 0;

277 }

**7.2 High Performance and Parallel Code**

This code for the Bioinformatics – Genome similarity using Frequency Vectors (C++) application was analysed and tested to provide the correct results in a short time as per the requirement for the CAB401 High Performance and Parallel Computing Assignment. This code can be seen below. All bacteria files, notes taken during testing, code and the Visual Studio 2017 project can be found at the link here <https://github.com/Starwolf-001/CAB401/tree/master/Bioinformatics_c_assignment_high_performance>.

1 #include <stdio.h>

2 #include <string.h>

3 #include <time.h>

4 #include <math.h>

5 #include <iostream>

6 #include <omp.h>

7

8 int number\_bacteria;

9 char\*\* bacteria\_name;

10 long M, M1, M2;

11 short code[27] = {0, 2, 1, 2, 3, 4, 5, 6, 7, -1, 8, 9, 10, 11, -1, 12, 13, 14, 15, 16, 1, 17, 18, 5, 19, 3};

12 #define encode(ch) code[ch-'A']

13 #define LEN 6

14 #define AA\_NUMBER 20

15 #define EPSILON 1e-010

16

17 void Init() {

18 M2 = 1;

24 for (int i = 0; i < LEN - 2; i++) { // M2 = AA\_NUMBER ^ (LEN-2);

25 M2 \*= AA\_NUMBER;

26 }

27 M1 = M2 \* AA\_NUMBER; // M1 = AA\_NUMBER ^ (LEN-1);

28 M = M1 \*AA\_NUMBER; // M = AA\_NUMBER ^ (LEN);

29 }

30

31 class Bacteria {

32 private:

33 long\* vector;

34 long\* second;

35 long one\_l[AA\_NUMBER];

36 long indexs;

37 long total;

38 long total\_l;

39 long complement;

40

41 void InitVectors() {

42 vector = new long[M];

43 second = new long[M1];

44 memset(vector, 0, M \* sizeof(long));

45 memset(second, 0, M1 \* sizeof(long));

46 memset(one\_l, 0, AA\_NUMBER \* sizeof(long));

47 total = 0;

48 total\_l = 0;

49 complement = 0;

50 }

51

52 void init\_buffer(char\* buffer) {

53 complement++;

54 indexs = 0;

59 for (int i = 0; i < LEN - 1; i++) {

60 short enc = encode(buffer[i]);

61 one\_l[enc]++;

62 total\_l++;

63 indexs = indexs \* AA\_NUMBER + enc;

64 }

65 second[indexs]++;

66 }

67

68 void cont\_buffer(char ch) {

69 short enc = encode(ch);

70 one\_l[enc]++;

71 total\_l++;

72 long index = indexs \* AA\_NUMBER + enc;

73 vector[index]++;

74 total++;

75 indexs = (indexs % M2) \* AA\_NUMBER + enc;

76 second[indexs]++;

77 }

78

79 public:

80 long count;

81 double\* tv;

82 long \*ti;

83

84 Bacteria(char\* filename) {

92 FILE \* bacteria\_file = fopen(filename, "r");

93 InitVectors();

94

95 char ch;

96 while ((ch = fgetc(bacteria\_file)) != EOF) {

97 if (ch == '>') {

98 while (fgetc(bacteria\_file) != '\n'); // skip rest of line

99

100 char buffer[LEN - 1];

101 fread(buffer, sizeof(char), LEN - 1, bacteria\_file);

102 init\_buffer(buffer);

103 }

104 else if (ch != '\n')

105 cont\_buffer(ch);

106 }

107 fclose(bacteria\_file);

108

109 long total\_plus\_complement = total + complement;

110 double total\_div\_2 = total \* 0.5;

111 int i\_mod\_aa\_number = 0;

112 int i\_div\_aa\_number = 0;

113 long i\_mod\_M1 = 0;

114 long i\_div\_M1 = 0;

115

116 double one\_l\_div\_total[AA\_NUMBER];

117 double\* second\_div\_total = new double[M1];

130 for (int i = 0; i < AA\_NUMBER; i++) {

131 one\_l\_div\_total[i] = (double)one\_l[i] / total\_l;

132 }

138 for (int i = 0; i < M1; i++) {

139 second\_div\_total[i] = (double)second[i] / total\_plus\_complement;

140 }

141

142 count = 0;

143 double\* t = new double[M];

144

156 for (long i = 0; i < M; i++) {

157 double p1 = second\_div\_total[i\_div\_aa\_number];

158 double p2 = one\_l\_div\_total[i\_mod\_aa\_number];

159 double p3 = second\_div\_total[i\_mod\_M1];

160 double p4 = one\_l\_div\_total[i\_div\_M1];

161 double stochastic = (p1 \* p2 + p3 \* p4) \* total\_div\_2;

162

163 if (i\_mod\_aa\_number == AA\_NUMBER - 1) {

164 i\_mod\_aa\_number = 0;

165 i\_div\_aa\_number++;

166 }

167 else {

168 i\_mod\_aa\_number++;

169 }

170

171 if (i\_mod\_M1 == M1 - 1) {

172 i\_mod\_M1 = 0;

173 i\_div\_M1++;

174 }

175 else {

176 i\_mod\_M1++;

177 }

178

179 if (stochastic > EPSILON) {

180 t[i] = (vector[i] - stochastic) / stochastic;

181 count++;

182 }

183 else {

184 t[i] = 0;

185 }

186 }

187

188 delete second\_div\_total;

189 delete vector;

190 delete second;

191

192 tv = new double[count];

193 ti = new long[count];

194

195 int pos = 0;

201 for (long i = 0; i < M; i++) {

202 if (t[i] != 0) {

203 tv[pos] = t[i];

204 ti[pos] = i;

205 pos++;

206 }

207 }

208 delete t;

209 }

210 };

211

212 void ReadInputFile(char\* input\_name) {

213 FILE\* input\_file = fopen(input\_name, "r");

214 fscanf(input\_file, "%d", &number\_bacteria);

215 bacteria\_name = new char\*[number\_bacteria];

220 for (long i = 0; i < number\_bacteria; i++) {

221 bacteria\_name[i] = new char[20];

222 fscanf(input\_file, "%s", bacteria\_name[i]);

223 strcat(bacteria\_name[i], ".faa");

224 }

225 fclose(input\_file);

226 }

227

228 double CompareBacteria(Bacteria\* b1, Bacteria\* b2) {

229 double correlation = 0;

230 double vector\_len1 = 0;

231 double vector\_len2 = 0;

232 long p1 = 0;

233 long p2 = 0;

237 while (p1 < b1->count && p2 < b2->count) {

238 long n1 = b1->ti[p1];

239 long n2 = b2->ti[p2];

240 if (n1 < n2) {

241 double t1 = b1->tv[p1];

242 vector\_len1 += (t1 \* t1);

243 p1++;

244 }

245 else if (n2 < n1) {

246 double t2 = b2->tv[p2];

247 p2++;

248 vector\_len2 += (t2 \* t2);

249 }

250 else {

251 double t1 = b1->tv[p1++];

252 double t2 = b2->tv[p2++];

253 vector\_len1 += (t1 \* t1);

254 vector\_len2 += (t2 \* t2);

255 correlation += t1 \* t2;

256 }

257 }

264 while (p1 < b1->count) {

265 long n1 = b1->ti[p1];

266 double t1 = b1->tv[p1++];

267 vector\_len1 += (t1 \* t1);

268 }

274 while (p2 < b2->count) {

275 long n2 = b2->ti[p2];

276 double t2 = b2->tv[p2++];

277 vector\_len2 += (t2 \* t2);

278 }

279 return correlation / (sqrt(vector\_len1) \* sqrt(vector\_len2));

280 }

281

282 void CompareAllBacteria() {

283 Bacteria\*\* b = new Bacteria\*[number\_bacteria];

291 #pragma omp parallel for

292 for (int i = 0; i < number\_bacteria; i++) {

293 printf("load %d of %d\n", i + 1, number\_bacteria);

294 b[i] = new Bacteria(bacteria\_name[i]);

295 }

296

342 #pragma omp parallel for

343 for (int i = 0; i < number\_bacteria - 1; i++) {

347 for (int j = i + 1; j < number\_bacteria; j++) {

348 double correlation = CompareBacteria(b[i], b[j]);

349 printf("%2d %2d -> %.20lf\n", i, j, correlation);

350 }

351 }

352 }

353

354 int main(int argc, char \* argv[]) {

355 time\_t t1 = time(NULL);

364 Init();

365 ReadInputFile(argv[1]);

366 CompareAllBacteria();

367

368 time\_t t2 = time(NULL);

369 printf("time elapsed: %d seconds\n", t2 - t1);

370

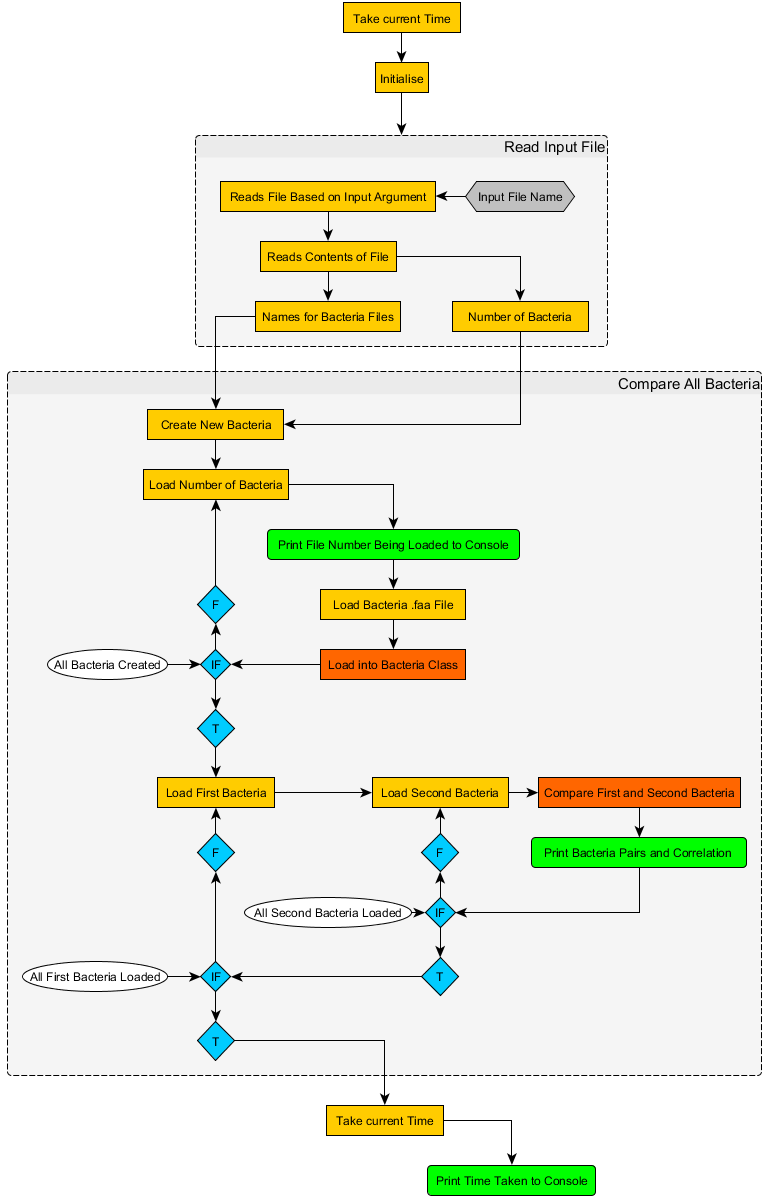
371 system("pause");

372

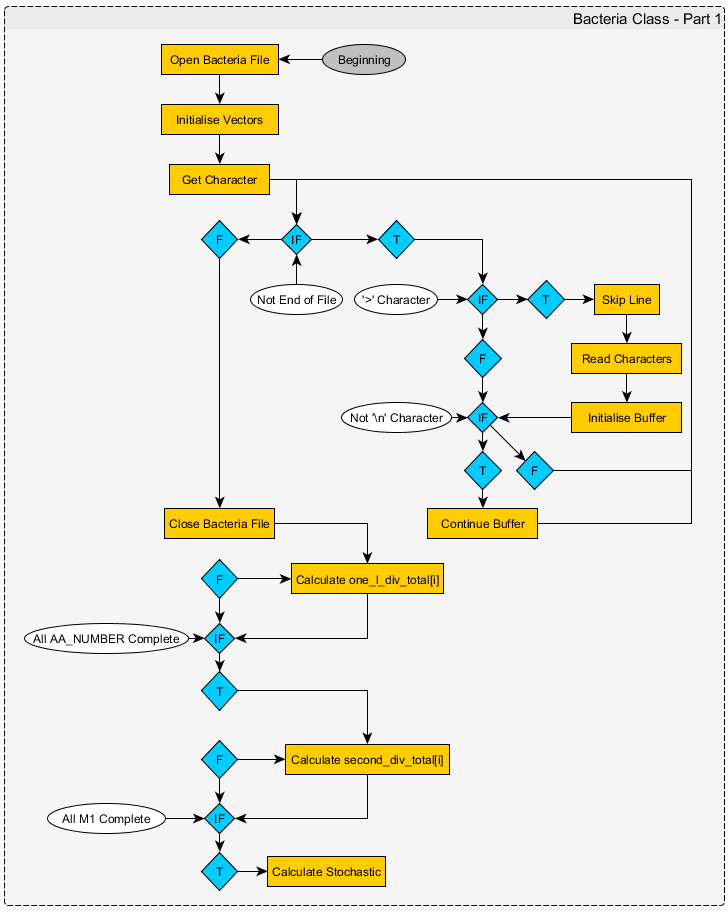
373 return 0;

374 }

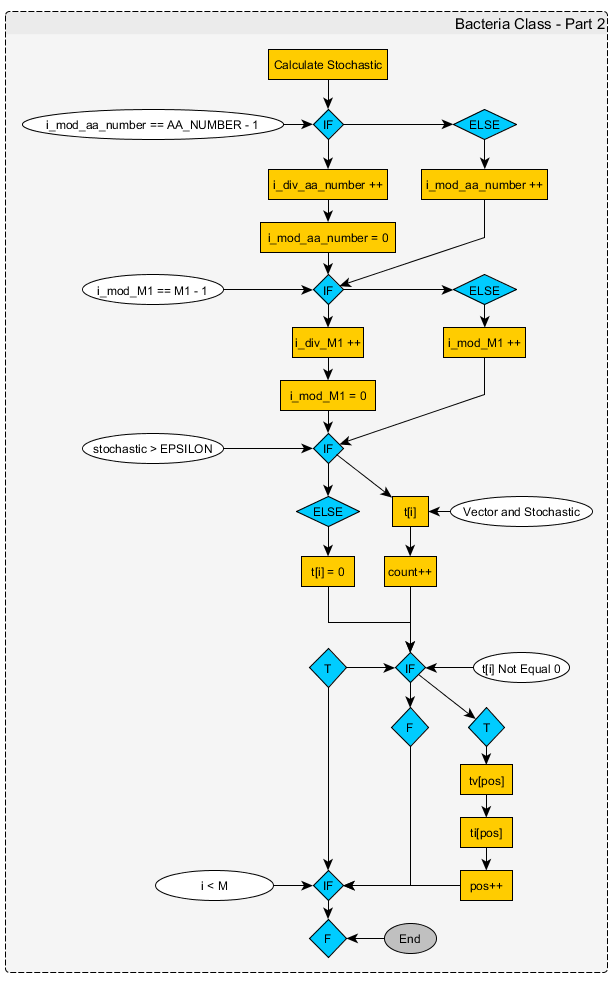
**7.3 Detailed Software Architecture**



**7.4 Bacteria Class – Part 1 Software Architecture**



**7.5 Bacteria Class – Part 2 Software Architecture**



**7.6 Compare Bacteria Software Architecture**

