Profiling

Caden Corontzos

I used callgrind to profile my code. Callgrind finds out how many instructions are associated with each line of code, which gives an indication of how long each line will take. I looked at some of the trouble spots that Eitan and I found to see if we could see a noticable difference in runtime before and after each change.

I used the two groups of DNA I had found the other day as benchmarks.

Here is the data before any optimizations.

Table 1: Corpus 1

File.Name	Original.File.Size	Compressed.Size	Compression.Ratio	Compresssion.Time	Decompression.Time
DNACorpus1/chmpxx	121024	43516	2.781	55	18
DNACorpus1/chntxx	155844	58336	2.671	77	27
DNACorpus1/hehcmv	229354	85526	2.682	118	36
DNACorpus1/humdyst	38770	15300	2.534	35	7
DNACorpus1/humghcs	66495	25552	2.602	33	12
DNACorpus1/humhbb	73308	28134	2.606	37	12
DNACorpus1/humhdab	58864	22699	2.593	30	32
DNACorpus1/humprtb	56737	21902	2.590	32	17
DNACorpus1/mpomtcg	186609	70254	2.656	122	46
DNACorpus1/mtpacga	100314	36862	2.721	73	18
DNACorpus1/vaccg	191737	70067	2.736	97	30

Table 2: Corpus 2

File.Name	Original.File.Size	Compressed.Size	Compression.Ratio	Compresssion.Time	Decompression.Time
DNACorpus2/AeCa	1591049	556535	2.859	1221	289
DNACorpus2/AgPh	43970	17442	2.521	21	8
DNACorpus2/AnCa	142189675	43665091	3.256	160561	33570
DNACorpus2/BuEb	18940	7893	2.400	34	7
DNACorpus2/DaRe	62565020	19586457	3.194	58734	16327
DNACorpus2/DrMe	32181429	10619042	3.031	29080	9883
DNACorpus2/EnIn	26403087	8609993	3.067	23231	7525
DNACorpus2/EsCo	4641652	1593404	2.913	3372	1110
DNACorpus2/GaGa	148532294	46851765	3.170	141631	27250
DNACorpus2/HaHi	3890005	1306708	2.977	3067	955
DNACorpus2/HePy	1667825	566972	2.942	1251	271
DNACorpus2/HoSa	189752667	57200209	3.317	168619	35951
DNACorpus2/OrSa	43262523	14148071	3.058	31413	7275
DNACorpus2/PlFa	8986712	2895744	3.103	6721	2041
DNACorpus2/ScPo	10652155	3590856	2.966	8229	2277
DNACorpus2/WaMe	9144432	3112000	2.938	6779	1994
DNACorpus2/YeMi	73689	27235	2.706	38	18

I then tested my implementation with callgrind. I encoded HaHi from DNA Corpus 2 to see what lines are taking long.

The first change we want to make was to create a type for codewords. They were previously just int, but we want to make them fixed at uint64 and have a special type for them.

I ran callgrind before and after making this change. I found that the amount of instructions was virtually identical, which makes sense since we didn't really change the structure of the code. I think that having a fixed codeword size will show some time saved once we turn compiler optimizations back on, but without that it didn't change much. I rand the corpus again just to confirm.

Table 3: Corpus 1

File.Name	Original.File.Size	Compressed.Size	Compression.Ratio	Compresssion.Time	Decompression.Time
DNACorpus1/chmpxx	121024	43516	2.781	60	22
DNACorpus1/chntxx	155844	58336	2.671	77	30
DNACorpus1/hehcmv	229354	85526	2.682	120	50
DNACorpus1/humdyst	38770	15300	2.534	20	16
DNACorpus1/humghcs	66495	25552	2.602	33	15
DNACorpus1/humhbb	73308	28134	2.606	35	23
DNACorpus1/humhdab	58864	22699	2.593	34	14
DNACorpus1/humprtb	56737	21902	2.590	32	19
DNACorpus1/mpomtcg	186609	70254	2.656	114	30
DNACorpus1/mtpacga	100314	36862	2.721	54	21
DNACorpus1/vaccg	191737	70067	2.736	96	28

Table 4: Corpus 2

File.Name	Original.File.Size	Compressed.Size	Compression.Ratio	Compresssion.Time	Decompression.Time
DNACorpus2/AeCa	1591049	556535	2.859	1027	359
DNACorpus2/AgPh	43970	17442	2.521	27	10
DNACorpus2/AnCa	142189675	43665091	3.256	130129	30638
DNACorpus2/BuEb	18940	7893	2.400	11	7
DNACorpus2/DaRe	62565020	19586457	3.194	55746	14370
DNACorpus2/DrMe	32181429	10619042	3.031	28462	5926
DNACorpus2/EnIn	26403087	8609993	3.067	22359	7255
DNACorpus2/EsCo	4641652	1593404	2.913	3806	1415
DNACorpus2/GaGa	148532294	46851765	3.170	141216	31454
DNACorpus2/HaHi	3890005	1306708	2.977	3001	692
DNACorpus2/HePy	1667825	566972	2.942	1197	333
DNACorpus2/HoSa	189752667	57200209	3.317	182781	41298
DNACorpus2/OrSa	43262523	14148071	3.058	35052	7914
DNACorpus2/PlFa	8986712	2895744	3.103	5889	2225
DNACorpus2/ScPo	10652155	3590856	2.966	8702	2699
DNACorpus2/WaMe	9144432	3112000	2.938	6045	1814
DNACorpus2/YeMi	73689	27235	2.706	36	14

These times look pretty similar to the first run.

One thing I noticed in the callgrind output was that there was significant time being spent each time we check if something is in the dictionary.

Here is the callgrind output for that line.

```
105,030,135 ( 0.18%)          if (dictionary.find(currentBlock + next_character) != dictionary.end()){
13,537,450,317 (22.83%)  => /usr/include/c++/9/bits/basic_string.h:std::__cxx11::basic_string<char, std
11,653,779,430 (19.65%)  => /usr/include/c++/9/bits/unordered_map.h:std::unordered_map<std::__cxx11::bas
2,108,383,120 ( 3.56%)  => /usr/include/c++/9/bits/basic_string.h:std::__cxx11::basic_string<char, std:
956,941,242 ( 1.61%)  => /usr/include/c++/9/bits/unordered_map.h:std::unordered_map<std::__cxx11::basic_
241,180,314 ( 0.41%)  => /usr/include/c++/9/bits/hashtable_policy.h:bool std::__detail::operator!=<std:
```

As shown, this line is taking a significant amount of instructions, and it needs to pull the end() of the dictionary each time it is ran. If we use cend() instead and save that iterator in a variable called end, we can save a significant amount of instructions.

```
89,470,115 ( 0.61%)          if (dictionary.find(currentBlock + next_character) != end ){
3,353,009,053 (22.78%)  => /usr/include/c++/9/bits/basic_string.h:std::__cxx11::basic_string<char, std:
2,833,786,025 (19.26%)  => /usr/include/c++/9/bits/unordered_map.h:std::unordered_map<std::__cxx11::bas
420,120,704 ( 2.85%)  => /usr/include/c++/9/bits/basic_string.h:std::__cxx11::basic_string<char, std::c
50,570,065 ( 0.34%)  => /usr/include/c++/9/bits/hashtable_policy.h:bool std::__detail::operator!=<std::
```

As you can see, the line itself takes less instructions, and it no has to pull up the end of the dictionary every time. This significantly decreased the total instructions in the program run. I then ran the corpus again to check the times.

Table 5: Corpus 1

File.Name	Original.File.Size	Compressed.Size	Compression.Ratio	Compresssion.Time	Decompression.Time
DNACorpus1/chmpxx	121024	43516	2.781	66	24
DNACorpus1/chntxx	155844	58336	2.671	72	61
DNACorpus1/hehcmv	229354	85526	2.682	127	65
DNACorpus1/humdyst	38770	15300	2.534	25	5
DNACorpus1/humghcs	66495	25552	2.602	30	20
DNACorpus1/humhbb	73308	28134	2.606	47	19
DNACorpus1/humhdab	58864	22699	2.593	41	14
DNACorpus1/humprtb	56737	21902	2.590	26	14
DNACorpus1/mpomtcg	186609	70254	2.656	110	34
DNACorpus1/mtpacga	100314	36862	2.721	44	18
DNACorpus1/vaccg	191737	70067	2.736	92	36

Table 6: Corpus 2

File.Name	Original.File.Size	Compressed.Size	Compression.Ratio	Compresssion.Time	Decompression.Time
DNACorpus2/AeCa	1591049	556535	2.859	982	328
DNACorpus2/AgPh	43970	17442	2.521	22	12
DNACorpus2/AnCa	142189675	43665091	3.256	114457	28818
DNACorpus2/BuEb	18940	7893	2.400	11	6
DNACorpus2/DaRe	62565020	19586457	3.194	51590	18192
DNACorpus2/DrMe	32181429	10619042	3.031	21125	5173
DNACorpus2/EnIn	26403087	8609993	3.067	15515	4387
DNACorpus2/EsCo	4641652	1593404	2.913	2555	775
DNACorpus2/GaGa	148532294	46851765	3.170	116135	33439
DNACorpus2/HaHi	3890005	1306708	2.977	2367	950
DNACorpus2/HePy	1667825	566972	2.942	1073	295
DNACorpus2/HoSa	189752667	57200209	3.317	154695	33068
DNACorpus2/OrSa	43262523	14148071	3.058	33018	7954
DNACorpus2/PlFa	8986712	2895744	3.103	6083	2212
DNACorpus2/ScPo	10652155	3590856	2.966	6358	1648
DNACorpus2/WaMe	9144432	3112000	2.938	5485	2338
DNACorpus2/YeMi	73689	27235	2.706	55	26

It's hard to see in the smaller files but if you look at the larger files in Corpus 2 you can see that the runtime has been reduced.

Another thing that I noticed from the callgrind output was that a lot of time/instructions are being spent on string concatenation. Every time we have already seen a sequence, we have to concatenate a character. I also noticed that I was doing this concatenation multiple times without needing to.

```
        if (dictionary.find(currentBlock + next_character) != end ){
            currentBlock = currentBlock + next_character;
        }
        else{

            // other code here ommitted

            dictionary[currentBlock + next_character] = codeword;
        }
```

If I just concatenate them and save the output into a new string, that will save me from doing the concatenation 2 more times.