1 Software

1.1 Optimizations

2 Hardware

hello

3 Appendix

3.1 Software

3.1.1 Main

```
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include "cmd_line.h"
#include "md5_sam.h"
#include <omp.h>
#include <math.h>
#include <time.h>
using namespace std;

//
#define N E7
#define _threads_ 16
#define LEN7

// password to crack
// 'aaaaaaa'
// #define passwd1 0x5d793fc5b00a2348
// #define passwd2 0xc3fb9ab59e5ca98a

// 'zzzzzzz'
#define passwd1 0xf0e8fb430bbdde6a
#define passwd2 0xe9c879a518fd895f




int main(int argc, char *argv[])
{
    transform_password(passwd1, passwd2);

    union Block in_block;
    union Hash hash; // hashes are initialized in constructor

    // For loop parameters
    uint64_t i;



    /*
        parallel required to make multiple threads

        for looks at the following and parallelizes the for loop

        Discarded for loop and went for sections instead.

        firstprivate uses the initial values of the variable and
        makes it private.

        num_threads determines the number of parallel instances.
        This is controlled by the _threads_ macro.
```

```
53
        The N macro determines the simulation length.
55  */
    #pragma omp parallel sections num_threads(_threads_) private(in_block,hash,i)
57  {
        /* Optimization.
59          Hard code in the values in the loop. Also choose the
            number of times we unwrap wisely (see write_pass)
61      */
        #pragma omp section
63      {
            uint64_t tmp = 0;
65          for (i = 0; i < N / _threads_; i+=26)
            {
67              write_first_pass(&in_block,   i+ 0); tmp += G_MD5(&in_block, &hash,
                    i+ 0); // Perform MD5 sum
                write_pass(&in_block,   i+ 1); tmp += F_MD5(&in_block, &hash, i+ 1);
                    // Perform MD5 sum
69              write_pass(&in_block,   i+ 2); tmp += F_MD5(&in_block, &hash, i+ 2);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 3); tmp += F_MD5(&in_block, &hash, i+ 3);
                    // Perform MD5 sum
71              write_pass(&in_block,   i+ 4); tmp += F_MD5(&in_block, &hash, i+ 4);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 5); tmp += F_MD5(&in_block, &hash, i+ 5);
                    // Perform MD5 sum
73              write_pass(&in_block,   i+ 6); tmp += F_MD5(&in_block, &hash, i+ 6);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 7); tmp += F_MD5(&in_block, &hash, i+ 7);
                    // Perform MD5 sum
75              write_pass(&in_block,   i+ 8); tmp += F_MD5(&in_block, &hash, i+ 8);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 9); tmp += F_MD5(&in_block, &hash, i+ 9);
                    // Perform MD5 sum
77
                write_pass(&in_block,   i+10); tmp += F_MD5(&in_block, &hash, i+10);
                    // Perform MD5 sum
79              write_pass(&in_block,   i+11); tmp += F_MD5(&in_block, &hash, i+11);
                    // Perform MD5 sum
                write_pass(&in_block,   i+12); tmp += F_MD5(&in_block, &hash, i+12);
                    // Perform MD5 sum
81              write_pass(&in_block,   i+13); tmp += F_MD5(&in_block, &hash, i+13);
                    // Perform MD5 sum
                write_pass(&in_block,   i+14); tmp += F_MD5(&in_block, &hash, i+14);
                    // Perform MD5 sum
83              write_pass(&in_block,   i+15); tmp += F_MD5(&in_block, &hash, i+15);
                    // Perform MD5 sum
                write_pass(&in_block,   i+16); tmp += F_MD5(&in_block, &hash, i+16);
                    // Perform MD5 sum
85              write_pass(&in_block,   i+17); tmp += F_MD5(&in_block, &hash, i+17);
                    // Perform MD5 sum
                write_pass(&in_block,   i+18); tmp += F_MD5(&in_block, &hash, i+18);
                    // Perform MD5 sum
87              write_pass(&in_block,   i+19); tmp += F_MD5(&in_block, &hash, i+19);
                    // Perform MD5 sum
```

```
89                  write_pass(&in_block,   i+20); tmp += F_MD5(&in_block, &hash, i+20);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+21); tmp += F_MD5(&in_block, &hash, i+21);
                        // Perform MD5 sum
91                  write_pass(&in_block,   i+22); tmp += F_MD5(&in_block, &hash, i+22);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+23); tmp += F_MD5(&in_block, &hash, i+23);
                        // Perform MD5 sum
93                  write_pass(&in_block,   i+24); tmp += F_MD5(&in_block, &hash, i+24);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+25); tmp += F_MD5(&in_block, &hash, i+25);
                        // Perform MD5 sum
95              }
                if (tmp != 0)
97              {
                    write_first_pass(&in_block, tmp);
99                  printf("found.␣%s\n", in_block._8);
                }
101         }
            // for 1 thread, comment after here.
103         #pragma omp section
            {
105             uint64_t tmp = 0;
                for (i = N / _threads_; i < 2 * N / _threads_; i+=26)
107             {
                    write_first_pass(&in_block,   i+ 0); tmp += G_MD5(&in_block, &hash,
                       i+ 0); // Perform MD5 sum
109                 write_pass(&in_block,   i+ 1); tmp += F_MD5(&in_block, &hash, i+ 1);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 2); tmp += F_MD5(&in_block, &hash, i+ 2);
                        // Perform MD5 sum
111                 write_pass(&in_block,   i+ 3); tmp += F_MD5(&in_block, &hash, i+ 3);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 4); tmp += F_MD5(&in_block, &hash, i+ 4);
                        // Perform MD5 sum
113                 write_pass(&in_block,   i+ 5); tmp += F_MD5(&in_block, &hash, i+ 5);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 6); tmp += F_MD5(&in_block, &hash, i+ 6);
                        // Perform MD5 sum
115                 write_pass(&in_block,   i+ 7); tmp += F_MD5(&in_block, &hash, i+ 7);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 8); tmp += F_MD5(&in_block, &hash, i+ 8);
                        // Perform MD5 sum
117                 write_pass(&in_block,   i+ 9); tmp += F_MD5(&in_block, &hash, i+ 9);
                        // Perform MD5 sum

119                 write_pass(&in_block,   i+10); tmp += F_MD5(&in_block, &hash, i+10);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+11); tmp += F_MD5(&in_block, &hash, i+11);
                        // Perform MD5 sum
121                 write_pass(&in_block,   i+12); tmp += F_MD5(&in_block, &hash, i+12);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+13); tmp += F_MD5(&in_block, &hash, i+13);
                        // Perform MD5 sum
123                 write_pass(&in_block,   i+14); tmp += F_MD5(&in_block, &hash, i+14);
                        // Perform MD5 sum
```

```
                write_pass(&in_block,   i+15); tmp += F_MD5(&in_block, &hash, i+15);
                    // Perform MD5 sum
125             write_pass(&in_block,   i+16); tmp += F_MD5(&in_block, &hash, i+16);
                    // Perform MD5 sum
                write_pass(&in_block,   i+17); tmp += F_MD5(&in_block, &hash, i+17);
                    // Perform MD5 sum
127             write_pass(&in_block,   i+18); tmp += F_MD5(&in_block, &hash, i+18);
                    // Perform MD5 sum
                write_pass(&in_block,   i+19); tmp += F_MD5(&in_block, &hash, i+19);
                    // Perform MD5 sum
129
                write_pass(&in_block,   i+20); tmp += F_MD5(&in_block, &hash, i+20);
                    // Perform MD5 sum
131             write_pass(&in_block,   i+21); tmp += F_MD5(&in_block, &hash, i+21);
                    // Perform MD5 sum
                write_pass(&in_block,   i+22); tmp += F_MD5(&in_block, &hash, i+22);
                    // Perform MD5 sum
133             write_pass(&in_block,   i+23); tmp += F_MD5(&in_block, &hash, i+23);
                    // Perform MD5 sum
                write_pass(&in_block,   i+24); tmp += F_MD5(&in_block, &hash, i+24);
                    // Perform MD5 sum
135             write_pass(&in_block,   i+25); tmp += F_MD5(&in_block, &hash, i+25);
                    // Perform MD5 sum
            }
137         if (tmp != 0)
            {
139             write_first_pass(&in_block, tmp);
                printf("found.␣%s\n\n", in_block._8);
141         }
        }
143     // for 2 threads, comment after here.
        #pragma omp section
145     {
            uint64_t tmp = 0;
147         for (i = 2 * N / _threads_; i < 3 * N / _threads_; i+=26)
            {
149             write_first_pass(&in_block,   i+ 0); tmp += G_MD5(&in_block, &hash,
                    i+ 0); // Perform MD5 sum
                write_pass(&in_block,   i+ 1); tmp += F_MD5(&in_block, &hash, i+ 1);
                    // Perform MD5 sum
151             write_pass(&in_block,   i+ 2); tmp += F_MD5(&in_block, &hash, i+ 2);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 3); tmp += F_MD5(&in_block, &hash, i+ 3);
                    // Perform MD5 sum
153             write_pass(&in_block,   i+ 4); tmp += F_MD5(&in_block, &hash, i+ 4);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 5); tmp += F_MD5(&in_block, &hash, i+ 5);
                    // Perform MD5 sum
155             write_pass(&in_block,   i+ 6); tmp += F_MD5(&in_block, &hash, i+ 6);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 7); tmp += F_MD5(&in_block, &hash, i+ 7);
                    // Perform MD5 sum
157             write_pass(&in_block,   i+ 8); tmp += F_MD5(&in_block, &hash, i+ 8);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 9); tmp += F_MD5(&in_block, &hash, i+ 9);
                    // Perform MD5 sum
159
```

```
                write_pass(&in_block,   i+10); tmp += F_MD5(&in_block, &hash, i+10);
                    // Perform MD5 sum
161             write_pass(&in_block,   i+11); tmp += F_MD5(&in_block, &hash, i+11);
                    // Perform MD5 sum
                write_pass(&in_block,   i+12); tmp += F_MD5(&in_block, &hash, i+12);
                    // Perform MD5 sum
163             write_pass(&in_block,   i+13); tmp += F_MD5(&in_block, &hash, i+13);
                    // Perform MD5 sum
                write_pass(&in_block,   i+14); tmp += F_MD5(&in_block, &hash, i+14);
                    // Perform MD5 sum
165             write_pass(&in_block,   i+15); tmp += F_MD5(&in_block, &hash, i+15);
                    // Perform MD5 sum
                write_pass(&in_block,   i+16); tmp += F_MD5(&in_block, &hash, i+16);
                    // Perform MD5 sum
167             write_pass(&in_block,   i+17); tmp += F_MD5(&in_block, &hash, i+17);
                    // Perform MD5 sum
                write_pass(&in_block,   i+18); tmp += F_MD5(&in_block, &hash, i+18);
                    // Perform MD5 sum
169             write_pass(&in_block,   i+19); tmp += F_MD5(&in_block, &hash, i+19);
                    // Perform MD5 sum

171             write_pass(&in_block,   i+20); tmp += F_MD5(&in_block, &hash, i+20);
                    // Perform MD5 sum
                write_pass(&in_block,   i+21); tmp += F_MD5(&in_block, &hash, i+21);
                    // Perform MD5 sum
173             write_pass(&in_block,   i+22); tmp += F_MD5(&in_block, &hash, i+22);
                    // Perform MD5 sum
                write_pass(&in_block,   i+23); tmp += F_MD5(&in_block, &hash, i+23);
                    // Perform MD5 sum
175             write_pass(&in_block,   i+24); tmp += F_MD5(&in_block, &hash, i+24);
                    // Perform MD5 sum
                write_pass(&in_block,   i+25); tmp += F_MD5(&in_block, &hash, i+25);
                    // Perform MD5 sum
177         }
            if (tmp != 0)
179         {
                write_first_pass(&in_block, tmp);
181             printf("found.␣%s\n", in_block._8);
            }
183     }
        #pragma omp section
185     {
            uint64_t tmp = 0;
187         for (i = 3 * N / _threads_; i < 4 * N / _threads_; i+=26)
            {
189             write_first_pass(&in_block,   i+ 0); tmp += G_MD5(&in_block, &hash,
                    i+ 0); // Perform MD5 sum
                write_pass(&in_block,   i+ 1); tmp += F_MD5(&in_block, &hash, i+ 1);
                    // Perform MD5 sum
191             write_pass(&in_block,   i+ 2); tmp += F_MD5(&in_block, &hash, i+ 2);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 3); tmp += F_MD5(&in_block, &hash, i+ 3);
                    // Perform MD5 sum
193             write_pass(&in_block,   i+ 4); tmp += F_MD5(&in_block, &hash, i+ 4);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 5); tmp += F_MD5(&in_block, &hash, i+ 5);
                    // Perform MD5 sum
```

```
                write_pass(&in_block,   i+ 6); tmp += F_MD5(&in_block, &hash, i+ 6);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 7); tmp += F_MD5(&in_block, &hash, i+ 7);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 8); tmp += F_MD5(&in_block, &hash, i+ 8);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 9); tmp += F_MD5(&in_block, &hash, i+ 9);
                    // Perform MD5 sum

                write_pass(&in_block,   i+10); tmp += F_MD5(&in_block, &hash, i+10);
                    // Perform MD5 sum
                write_pass(&in_block,   i+11); tmp += F_MD5(&in_block, &hash, i+11);
                    // Perform MD5 sum
                write_pass(&in_block,   i+12); tmp += F_MD5(&in_block, &hash, i+12);
                    // Perform MD5 sum
                write_pass(&in_block,   i+13); tmp += F_MD5(&in_block, &hash, i+13);
                    // Perform MD5 sum
                write_pass(&in_block,   i+14); tmp += F_MD5(&in_block, &hash, i+14);
                    // Perform MD5 sum
                write_pass(&in_block,   i+15); tmp += F_MD5(&in_block, &hash, i+15);
                    // Perform MD5 sum
                write_pass(&in_block,   i+16); tmp += F_MD5(&in_block, &hash, i+16);
                    // Perform MD5 sum
                write_pass(&in_block,   i+17); tmp += F_MD5(&in_block, &hash, i+17);
                    // Perform MD5 sum
                write_pass(&in_block,   i+18); tmp += F_MD5(&in_block, &hash, i+18);
                    // Perform MD5 sum
                write_pass(&in_block,   i+19); tmp += F_MD5(&in_block, &hash, i+19);
                    // Perform MD5 sum

                write_pass(&in_block,   i+20); tmp += F_MD5(&in_block, &hash, i+20);
                    // Perform MD5 sum
                write_pass(&in_block,   i+21); tmp += F_MD5(&in_block, &hash, i+21);
                    // Perform MD5 sum
                write_pass(&in_block,   i+22); tmp += F_MD5(&in_block, &hash, i+22);
                    // Perform MD5 sum
                write_pass(&in_block,   i+23); tmp += F_MD5(&in_block, &hash, i+23);
                    // Perform MD5 sum
                write_pass(&in_block,   i+24); tmp += F_MD5(&in_block, &hash, i+24);
                    // Perform MD5 sum
                write_pass(&in_block,   i+25); tmp += F_MD5(&in_block, &hash, i+25);
                    // Perform MD5 sum
            }
            if (tmp != 0)
            {
                write_first_pass(&in_block, tmp);
                printf("found. %s\n", in_block._8);
            }
        }
        // for 4 threads, comment after here

        #pragma omp section
        {
            uint64_t tmp = 0;
            for (i = 4 * N / _threads_; i < 5 * N / _threads_; i+=26)
            {
```

```
231                 write_first_pass(&in_block,   i+ 0); tmp += G_MD5(&in_block, &hash,
                        i+ 0); // Perform MD5 sum
                    write_pass(&in_block,   i+ 1); tmp += F_MD5(&in_block, &hash, i+ 1);
                        // Perform MD5 sum
233                 write_pass(&in_block,   i+ 2); tmp += F_MD5(&in_block, &hash, i+ 2);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 3); tmp += F_MD5(&in_block, &hash, i+ 3);
                        // Perform MD5 sum
235                 write_pass(&in_block,   i+ 4); tmp += F_MD5(&in_block, &hash, i+ 4);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 5); tmp += F_MD5(&in_block, &hash, i+ 5);
                        // Perform MD5 sum
237                 write_pass(&in_block,   i+ 6); tmp += F_MD5(&in_block, &hash, i+ 6);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 7); tmp += F_MD5(&in_block, &hash, i+ 7);
                        // Perform MD5 sum
239                 write_pass(&in_block,   i+ 8); tmp += F_MD5(&in_block, &hash, i+ 8);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 9); tmp += F_MD5(&in_block, &hash, i+ 9);
                        // Perform MD5 sum
241
                    write_pass(&in_block,   i+10); tmp += F_MD5(&in_block, &hash, i+10);
                        // Perform MD5 sum
243                 write_pass(&in_block,   i+11); tmp += F_MD5(&in_block, &hash, i+11);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+12); tmp += F_MD5(&in_block, &hash, i+12);
                        // Perform MD5 sum
245                 write_pass(&in_block,   i+13); tmp += F_MD5(&in_block, &hash, i+13);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+14); tmp += F_MD5(&in_block, &hash, i+14);
                        // Perform MD5 sum
247                 write_pass(&in_block,   i+15); tmp += F_MD5(&in_block, &hash, i+15);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+16); tmp += F_MD5(&in_block, &hash, i+16);
                        // Perform MD5 sum
249                 write_pass(&in_block,   i+17); tmp += F_MD5(&in_block, &hash, i+17);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+18); tmp += F_MD5(&in_block, &hash, i+18);
                        // Perform MD5 sum
251                 write_pass(&in_block,   i+19); tmp += F_MD5(&in_block, &hash, i+19);
                        // Perform MD5 sum

253                 write_pass(&in_block,   i+20); tmp += F_MD5(&in_block, &hash, i+20);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+21); tmp += F_MD5(&in_block, &hash, i+21);
                        // Perform MD5 sum
255                 write_pass(&in_block,   i+22); tmp += F_MD5(&in_block, &hash, i+22);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+23); tmp += F_MD5(&in_block, &hash, i+23);
                        // Perform MD5 sum
257                 write_pass(&in_block,   i+24); tmp += F_MD5(&in_block, &hash, i+24);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+25); tmp += F_MD5(&in_block, &hash, i+25);
                        // Perform MD5 sum
259             }
                if (tmp != 0)
261             {
```

```
                write_first_pass(&in_block, tmp);
263             printf("found. %s\n", in_block._8);
            }
265     }
        #pragma omp section
267     {
            uint64_t tmp = 0;
269         for (i = 5 * N / _threads_; i < 6 * N / _threads_; i+=26)
            {
271             write_first_pass(&in_block,   i+ 0); tmp += G_MD5(&in_block, &hash,
                    i+ 0); // Perform MD5 sum
                write_pass(&in_block,   i+ 1); tmp += F_MD5(&in_block, &hash, i+ 1);
                    // Perform MD5 sum
273             write_pass(&in_block,   i+ 2); tmp += F_MD5(&in_block, &hash, i+ 2);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 3); tmp += F_MD5(&in_block, &hash, i+ 3);
                    // Perform MD5 sum
275             write_pass(&in_block,   i+ 4); tmp += F_MD5(&in_block, &hash, i+ 4);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 5); tmp += F_MD5(&in_block, &hash, i+ 5);
                    // Perform MD5 sum
277             write_pass(&in_block,   i+ 6); tmp += F_MD5(&in_block, &hash, i+ 6);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 7); tmp += F_MD5(&in_block, &hash, i+ 7);
                    // Perform MD5 sum
279             write_pass(&in_block,   i+ 8); tmp += F_MD5(&in_block, &hash, i+ 8);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 9); tmp += F_MD5(&in_block, &hash, i+ 9);
                    // Perform MD5 sum
281
                write_pass(&in_block,   i+10); tmp += F_MD5(&in_block, &hash, i+10);
                    // Perform MD5 sum
283             write_pass(&in_block,   i+11); tmp += F_MD5(&in_block, &hash, i+11);
                    // Perform MD5 sum
                write_pass(&in_block,   i+12); tmp += F_MD5(&in_block, &hash, i+12);
                    // Perform MD5 sum
285             write_pass(&in_block,   i+13); tmp += F_MD5(&in_block, &hash, i+13);
                    // Perform MD5 sum
                write_pass(&in_block,   i+14); tmp += F_MD5(&in_block, &hash, i+14);
                    // Perform MD5 sum
287             write_pass(&in_block,   i+15); tmp += F_MD5(&in_block, &hash, i+15);
                    // Perform MD5 sum
                write_pass(&in_block,   i+16); tmp += F_MD5(&in_block, &hash, i+16);
                    // Perform MD5 sum
289             write_pass(&in_block,   i+17); tmp += F_MD5(&in_block, &hash, i+17);
                    // Perform MD5 sum
                write_pass(&in_block,   i+18); tmp += F_MD5(&in_block, &hash, i+18);
                    // Perform MD5 sum
291             write_pass(&in_block,   i+19); tmp += F_MD5(&in_block, &hash, i+19);
                    // Perform MD5 sum

293             write_pass(&in_block,   i+20); tmp += F_MD5(&in_block, &hash, i+20);
                    // Perform MD5 sum
                write_pass(&in_block,   i+21); tmp += F_MD5(&in_block, &hash, i+21);
                    // Perform MD5 sum
295             write_pass(&in_block,   i+22); tmp += F_MD5(&in_block, &hash, i+22);
                    // Perform MD5 sum
```

```
                write_pass(&in_block,   i+23); tmp += F_MD5(&in_block, &hash, i+23);
                    // Perform MD5 sum
297             write_pass(&in_block,   i+24); tmp += F_MD5(&in_block, &hash, i+24);
                    // Perform MD5 sum
                write_pass(&in_block,   i+25); tmp += F_MD5(&in_block, &hash, i+25);
                    // Perform MD5 sum
299         }
            if (tmp != 0)
301         {
                write_first_pass(&in_block, tmp);
303             printf("found. %s\n", in_block._8);
            }
305     }
        #pragma omp section
307     {
            uint64_t tmp = 0;
309         for (i = 6 * N / _threads_; i < 7 * N / _threads_; i+=26)
            {
311             write_first_pass(&in_block,   i+ 0); tmp += G_MD5(&in_block, &hash,
                   i+ 0); // Perform MD5 sum
                write_pass(&in_block,   i+ 1); tmp += F_MD5(&in_block, &hash, i+ 1);
                    // Perform MD5 sum
313             write_pass(&in_block,   i+ 2); tmp += F_MD5(&in_block, &hash, i+ 2);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 3); tmp += F_MD5(&in_block, &hash, i+ 3);
                    // Perform MD5 sum
315             write_pass(&in_block,   i+ 4); tmp += F_MD5(&in_block, &hash, i+ 4);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 5); tmp += F_MD5(&in_block, &hash, i+ 5);
                    // Perform MD5 sum
317             write_pass(&in_block,   i+ 6); tmp += F_MD5(&in_block, &hash, i+ 6);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 7); tmp += F_MD5(&in_block, &hash, i+ 7);
                    // Perform MD5 sum
319             write_pass(&in_block,   i+ 8); tmp += F_MD5(&in_block, &hash, i+ 8);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 9); tmp += F_MD5(&in_block, &hash, i+ 9);
                    // Perform MD5 sum
321
                write_pass(&in_block,   i+10); tmp += F_MD5(&in_block, &hash, i+10);
                    // Perform MD5 sum
323             write_pass(&in_block,   i+11); tmp += F_MD5(&in_block, &hash, i+11);
                    // Perform MD5 sum
                write_pass(&in_block,   i+12); tmp += F_MD5(&in_block, &hash, i+12);
                    // Perform MD5 sum
325             write_pass(&in_block,   i+13); tmp += F_MD5(&in_block, &hash, i+13);
                    // Perform MD5 sum
                write_pass(&in_block,   i+14); tmp += F_MD5(&in_block, &hash, i+14);
                    // Perform MD5 sum
327             write_pass(&in_block,   i+15); tmp += F_MD5(&in_block, &hash, i+15);
                    // Perform MD5 sum
                write_pass(&in_block,   i+16); tmp += F_MD5(&in_block, &hash, i+16);
                    // Perform MD5 sum
329             write_pass(&in_block,   i+17); tmp += F_MD5(&in_block, &hash, i+17);
                    // Perform MD5 sum
                write_pass(&in_block,   i+18); tmp += F_MD5(&in_block, &hash, i+18);
                    // Perform MD5 sum
```

```
331                 write_pass(&in_block,   i+19); tmp += F_MD5(&in_block, &hash, i+19);
                        // Perform MD5 sum

333                 write_pass(&in_block,   i+20); tmp += F_MD5(&in_block, &hash, i+20);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+21); tmp += F_MD5(&in_block, &hash, i+21);
                        // Perform MD5 sum
335                 write_pass(&in_block,   i+22); tmp += F_MD5(&in_block, &hash, i+22);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+23); tmp += F_MD5(&in_block, &hash, i+23);
                        // Perform MD5 sum
337                 write_pass(&in_block,   i+24); tmp += F_MD5(&in_block, &hash, i+24);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+25); tmp += F_MD5(&in_block, &hash, i+25);
                        // Perform MD5 sum
339             }
                if (tmp != 0)
341             {
                    write_first_pass(&in_block, tmp);
343                 printf("found.␣%s\n", in_block._8);
                }
345         }

347         #pragma omp section
            {
349             uint64_t tmp = 0;
                for (i = 7 * N / _threads_; i < 8 * N / _threads_; i+=26)
351             {
                    write_first_pass(&in_block,   i+ 0); tmp += G_MD5(&in_block, &hash,
                       i+ 0); // Perform MD5 sum
353                 write_pass(&in_block,   i+ 1); tmp += F_MD5(&in_block, &hash, i+ 1);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 2); tmp += F_MD5(&in_block, &hash, i+ 2);
                        // Perform MD5 sum
355                 write_pass(&in_block,   i+ 3); tmp += F_MD5(&in_block, &hash, i+ 3);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 4); tmp += F_MD5(&in_block, &hash, i+ 4);
                        // Perform MD5 sum
357                 write_pass(&in_block,   i+ 5); tmp += F_MD5(&in_block, &hash, i+ 5);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 6); tmp += F_MD5(&in_block, &hash, i+ 6);
                        // Perform MD5 sum
359                 write_pass(&in_block,   i+ 7); tmp += F_MD5(&in_block, &hash, i+ 7);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 8); tmp += F_MD5(&in_block, &hash, i+ 8);
                        // Perform MD5 sum
361                 write_pass(&in_block,   i+ 9); tmp += F_MD5(&in_block, &hash, i+ 9);
                        // Perform MD5 sum

363                 write_pass(&in_block,   i+10); tmp += F_MD5(&in_block, &hash, i+10);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+11); tmp += F_MD5(&in_block, &hash, i+11);
                        // Perform MD5 sum
365                 write_pass(&in_block,   i+12); tmp += F_MD5(&in_block, &hash, i+12);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+13); tmp += F_MD5(&in_block, &hash, i+13);
                        // Perform MD5 sum
```

```
367                 write_pass(&in_block,   i+14); tmp += F_MD5(&in_block, &hash, i+14);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+15); tmp += F_MD5(&in_block, &hash, i+15);
                        // Perform MD5 sum
369                 write_pass(&in_block,   i+16); tmp += F_MD5(&in_block, &hash, i+16);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+17); tmp += F_MD5(&in_block, &hash, i+17);
                        // Perform MD5 sum
371                 write_pass(&in_block,   i+18); tmp += F_MD5(&in_block, &hash, i+18);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+19); tmp += F_MD5(&in_block, &hash, i+19);
                        // Perform MD5 sum
373
                    write_pass(&in_block,   i+20); tmp += F_MD5(&in_block, &hash, i+20);
                        // Perform MD5 sum
375                 write_pass(&in_block,   i+21); tmp += F_MD5(&in_block, &hash, i+21);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+22); tmp += F_MD5(&in_block, &hash, i+22);
                        // Perform MD5 sum
377                 write_pass(&in_block,   i+23); tmp += F_MD5(&in_block, &hash, i+23);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+24); tmp += F_MD5(&in_block, &hash, i+24);
                        // Perform MD5 sum
379                 write_pass(&in_block,   i+25); tmp += F_MD5(&in_block, &hash, i+25);
                        // Perform MD5 sum
                }
381             if (tmp != 0)
                {
383                 write_first_pass(&in_block, tmp);
                    printf("found.␣%s\n", in_block._8);
385             }
            }
387         // for 8 threads, comment after here
            #pragma omp section
389         {
                uint64_t tmp = 0;
391             for (i = 8 * N / _threads_; i < 9 * N / _threads_; i+=26)
                {
393                 write_first_pass(&in_block,   i+ 0); tmp += G_MD5(&in_block, &hash,
                        i+ 0); // Perform MD5 sum
                    write_pass(&in_block,   i+ 1); tmp += F_MD5(&in_block, &hash, i+ 1);
                        // Perform MD5 sum
395                 write_pass(&in_block,   i+ 2); tmp += F_MD5(&in_block, &hash, i+ 2);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 3); tmp += F_MD5(&in_block, &hash, i+ 3);
                        // Perform MD5 sum
397                 write_pass(&in_block,   i+ 4); tmp += F_MD5(&in_block, &hash, i+ 4);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 5); tmp += F_MD5(&in_block, &hash, i+ 5);
                        // Perform MD5 sum
399                 write_pass(&in_block,   i+ 6); tmp += F_MD5(&in_block, &hash, i+ 6);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 7); tmp += F_MD5(&in_block, &hash, i+ 7);
                        // Perform MD5 sum
401                 write_pass(&in_block,   i+ 8); tmp += F_MD5(&in_block, &hash, i+ 8);
                        // Perform MD5 sum
```

```
                write_pass(&in_block,   i+ 9); tmp += F_MD5(&in_block, &hash, i+ 9);
                    // Perform MD5 sum

                write_pass(&in_block,   i+10); tmp += F_MD5(&in_block, &hash, i+10);
                    // Perform MD5 sum
                write_pass(&in_block,   i+11); tmp += F_MD5(&in_block, &hash, i+11);
                    // Perform MD5 sum
                write_pass(&in_block,   i+12); tmp += F_MD5(&in_block, &hash, i+12);
                    // Perform MD5 sum
                write_pass(&in_block,   i+13); tmp += F_MD5(&in_block, &hash, i+13);
                    // Perform MD5 sum
                write_pass(&in_block,   i+14); tmp += F_MD5(&in_block, &hash, i+14);
                    // Perform MD5 sum
                write_pass(&in_block,   i+15); tmp += F_MD5(&in_block, &hash, i+15);
                    // Perform MD5 sum
                write_pass(&in_block,   i+16); tmp += F_MD5(&in_block, &hash, i+16);
                    // Perform MD5 sum
                write_pass(&in_block,   i+17); tmp += F_MD5(&in_block, &hash, i+17);
                    // Perform MD5 sum
                write_pass(&in_block,   i+18); tmp += F_MD5(&in_block, &hash, i+18);
                    // Perform MD5 sum
                write_pass(&in_block,   i+19); tmp += F_MD5(&in_block, &hash, i+19);
                    // Perform MD5 sum

                write_pass(&in_block,   i+20); tmp += F_MD5(&in_block, &hash, i+20);
                    // Perform MD5 sum
                write_pass(&in_block,   i+21); tmp += F_MD5(&in_block, &hash, i+21);
                    // Perform MD5 sum
                write_pass(&in_block,   i+22); tmp += F_MD5(&in_block, &hash, i+22);
                    // Perform MD5 sum
                write_pass(&in_block,   i+23); tmp += F_MD5(&in_block, &hash, i+23);
                    // Perform MD5 sum
                write_pass(&in_block,   i+24); tmp += F_MD5(&in_block, &hash, i+24);
                    // Perform MD5 sum
                write_pass(&in_block,   i+25); tmp += F_MD5(&in_block, &hash, i+25);
                    // Perform MD5 sum
            }
            if (tmp != 0)
            {
                write_first_pass(&in_block, tmp);
                printf("found. %s\n", in_block._8);
            }
        }
        #pragma omp section
        {
            uint64_t tmp = 0;
            for (i = 9 * N / _threads_; i < 10 * N / _threads_; i+=26)
            {
                write_first_pass(&in_block,   i+ 0); tmp += G_MD5(&in_block, &hash,
                    i+ 0); // Perform MD5 sum
                write_pass(&in_block,   i+ 1); tmp += F_MD5(&in_block, &hash, i+ 1);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 2); tmp += F_MD5(&in_block, &hash, i+ 2);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 3); tmp += F_MD5(&in_block, &hash, i+ 3);
                    // Perform MD5 sum
```

```
437                 write_pass(&in_block,   i+ 4); tmp += F_MD5(&in_block, &hash, i+ 4);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 5); tmp += F_MD5(&in_block, &hash, i+ 5);
                        // Perform MD5 sum
439                 write_pass(&in_block,   i+ 6); tmp += F_MD5(&in_block, &hash, i+ 6);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 7); tmp += F_MD5(&in_block, &hash, i+ 7);
                        // Perform MD5 sum
441                 write_pass(&in_block,   i+ 8); tmp += F_MD5(&in_block, &hash, i+ 8);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 9); tmp += F_MD5(&in_block, &hash, i+ 9);
                        // Perform MD5 sum
443
                    write_pass(&in_block,   i+10); tmp += F_MD5(&in_block, &hash, i+10);
                        // Perform MD5 sum
445                 write_pass(&in_block,   i+11); tmp += F_MD5(&in_block, &hash, i+11);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+12); tmp += F_MD5(&in_block, &hash, i+12);
                        // Perform MD5 sum
447                 write_pass(&in_block,   i+13); tmp += F_MD5(&in_block, &hash, i+13);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+14); tmp += F_MD5(&in_block, &hash, i+14);
                        // Perform MD5 sum
449                 write_pass(&in_block,   i+15); tmp += F_MD5(&in_block, &hash, i+15);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+16); tmp += F_MD5(&in_block, &hash, i+16);
                        // Perform MD5 sum
451                 write_pass(&in_block,   i+17); tmp += F_MD5(&in_block, &hash, i+17);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+18); tmp += F_MD5(&in_block, &hash, i+18);
                        // Perform MD5 sum
453                 write_pass(&in_block,   i+19); tmp += F_MD5(&in_block, &hash, i+19);
                        // Perform MD5 sum

455                 write_pass(&in_block,   i+20); tmp += F_MD5(&in_block, &hash, i+20);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+21); tmp += F_MD5(&in_block, &hash, i+21);
                        // Perform MD5 sum
457                 write_pass(&in_block,   i+22); tmp += F_MD5(&in_block, &hash, i+22);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+23); tmp += F_MD5(&in_block, &hash, i+23);
                        // Perform MD5 sum
459                 write_pass(&in_block,   i+24); tmp += F_MD5(&in_block, &hash, i+24);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+25); tmp += F_MD5(&in_block, &hash, i+25);
                        // Perform MD5 sum
461             }
                if (tmp != 0)
463             {
                    write_first_pass(&in_block, tmp);
465                 printf("found.␣%s\n", in_block._8);
                }
467         }
            #pragma omp section
469         {
                uint64_t tmp = 0;
471             for (i = 10 * N / _threads_; i < 11 * N / _threads_; i+=26)
```

```
            {
473             write_first_pass(&in_block,   i+ 0); tmp += G_MD5(&in_block, &hash,
                    i+ 0); // Perform MD5 sum
                write_pass(&in_block,   i+ 1); tmp += F_MD5(&in_block, &hash, i+ 1);
                    // Perform MD5 sum
475             write_pass(&in_block,   i+ 2); tmp += F_MD5(&in_block, &hash, i+ 2);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 3); tmp += F_MD5(&in_block, &hash, i+ 3);
                    // Perform MD5 sum
477             write_pass(&in_block,   i+ 4); tmp += F_MD5(&in_block, &hash, i+ 4);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 5); tmp += F_MD5(&in_block, &hash, i+ 5);
                    // Perform MD5 sum
479             write_pass(&in_block,   i+ 6); tmp += F_MD5(&in_block, &hash, i+ 6);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 7); tmp += F_MD5(&in_block, &hash, i+ 7);
                    // Perform MD5 sum
481             write_pass(&in_block,   i+ 8); tmp += F_MD5(&in_block, &hash, i+ 8);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 9); tmp += F_MD5(&in_block, &hash, i+ 9);
                    // Perform MD5 sum
483
                write_pass(&in_block,   i+10); tmp += F_MD5(&in_block, &hash, i+10);
                    // Perform MD5 sum
485             write_pass(&in_block,   i+11); tmp += F_MD5(&in_block, &hash, i+11);
                    // Perform MD5 sum
                write_pass(&in_block,   i+12); tmp += F_MD5(&in_block, &hash, i+12);
                    // Perform MD5 sum
487             write_pass(&in_block,   i+13); tmp += F_MD5(&in_block, &hash, i+13);
                    // Perform MD5 sum
                write_pass(&in_block,   i+14); tmp += F_MD5(&in_block, &hash, i+14);
                    // Perform MD5 sum
489             write_pass(&in_block,   i+15); tmp += F_MD5(&in_block, &hash, i+15);
                    // Perform MD5 sum
                write_pass(&in_block,   i+16); tmp += F_MD5(&in_block, &hash, i+16);
                    // Perform MD5 sum
491             write_pass(&in_block,   i+17); tmp += F_MD5(&in_block, &hash, i+17);
                    // Perform MD5 sum
                write_pass(&in_block,   i+18); tmp += F_MD5(&in_block, &hash, i+18);
                    // Perform MD5 sum
493             write_pass(&in_block,   i+19); tmp += F_MD5(&in_block, &hash, i+19);
                    // Perform MD5 sum

495             write_pass(&in_block,   i+20); tmp += F_MD5(&in_block, &hash, i+20);
                    // Perform MD5 sum
                write_pass(&in_block,   i+21); tmp += F_MD5(&in_block, &hash, i+21);
                    // Perform MD5 sum
497             write_pass(&in_block,   i+22); tmp += F_MD5(&in_block, &hash, i+22);
                    // Perform MD5 sum
                write_pass(&in_block,   i+23); tmp += F_MD5(&in_block, &hash, i+23);
                    // Perform MD5 sum
499             write_pass(&in_block,   i+24); tmp += F_MD5(&in_block, &hash, i+24);
                    // Perform MD5 sum
                write_pass(&in_block,   i+25); tmp += F_MD5(&in_block, &hash, i+25);
                    // Perform MD5 sum
501         }
            if (tmp != 0)
```

```
503             {
                    write_first_pass(&in_block, tmp);
505                 printf("found. %s\n", in_block._8);
                }
507         }
            #pragma omp section
509         {
                uint64_t tmp = 0;
511             for (i = 11 * N / _threads_; i < 12 * N / _threads_; i+=26)
                {
513                 write_first_pass(&in_block,   i+ 0); tmp += G_MD5(&in_block, &hash,
                        i+ 0); // Perform MD5 sum
                    write_pass(&in_block,   i+ 1); tmp += F_MD5(&in_block, &hash, i+ 1);
                        // Perform MD5 sum
515                 write_pass(&in_block,   i+ 2); tmp += F_MD5(&in_block, &hash, i+ 2);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 3); tmp += F_MD5(&in_block, &hash, i+ 3);
                        // Perform MD5 sum
517                 write_pass(&in_block,   i+ 4); tmp += F_MD5(&in_block, &hash, i+ 4);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 5); tmp += F_MD5(&in_block, &hash, i+ 5);
                        // Perform MD5 sum
519                 write_pass(&in_block,   i+ 6); tmp += F_MD5(&in_block, &hash, i+ 6);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 7); tmp += F_MD5(&in_block, &hash, i+ 7);
                        // Perform MD5 sum
521                 write_pass(&in_block,   i+ 8); tmp += F_MD5(&in_block, &hash, i+ 8);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 9); tmp += F_MD5(&in_block, &hash, i+ 9);
                        // Perform MD5 sum
523
                    write_pass(&in_block,   i+10); tmp += F_MD5(&in_block, &hash, i+10);
                        // Perform MD5 sum
525                 write_pass(&in_block,   i+11); tmp += F_MD5(&in_block, &hash, i+11);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+12); tmp += F_MD5(&in_block, &hash, i+12);
                        // Perform MD5 sum
527                 write_pass(&in_block,   i+13); tmp += F_MD5(&in_block, &hash, i+13);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+14); tmp += F_MD5(&in_block, &hash, i+14);
                        // Perform MD5 sum
529                 write_pass(&in_block,   i+15); tmp += F_MD5(&in_block, &hash, i+15);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+16); tmp += F_MD5(&in_block, &hash, i+16);
                        // Perform MD5 sum
531                 write_pass(&in_block,   i+17); tmp += F_MD5(&in_block, &hash, i+17);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+18); tmp += F_MD5(&in_block, &hash, i+18);
                        // Perform MD5 sum
533                 write_pass(&in_block,   i+19); tmp += F_MD5(&in_block, &hash, i+19);
                        // Perform MD5 sum

535                 write_pass(&in_block,   i+20); tmp += F_MD5(&in_block, &hash, i+20);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+21); tmp += F_MD5(&in_block, &hash, i+21);
                        // Perform MD5 sum
```

```
537                 write_pass(&in_block,   i+22); tmp += F_MD5(&in_block, &hash, i+22);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+23); tmp += F_MD5(&in_block, &hash, i+23);
                        // Perform MD5 sum
539                 write_pass(&in_block,   i+24); tmp += F_MD5(&in_block, &hash, i+24);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+25); tmp += F_MD5(&in_block, &hash, i+25);
                        // Perform MD5 sum
541             }
                if (tmp != 0)
543             {
                    write_first_pass(&in_block, tmp);
545                 printf("found. %s\n", in_block._8);
                }
547         }
            #pragma omp section
549         {
                uint64_t tmp = 0;
551             for (i = 12 * N / _threads_; i < 13 * N / _threads_; i+=26)
                {
553                 write_first_pass(&in_block,   i+ 0); tmp += G_MD5(&in_block, &hash,
                       i+ 0); // Perform MD5 sum
                    write_pass(&in_block,   i+ 1); tmp += F_MD5(&in_block, &hash, i+ 1);
                        // Perform MD5 sum
555                 write_pass(&in_block,   i+ 2); tmp += F_MD5(&in_block, &hash, i+ 2);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 3); tmp += F_MD5(&in_block, &hash, i+ 3);
                        // Perform MD5 sum
557                 write_pass(&in_block,   i+ 4); tmp += F_MD5(&in_block, &hash, i+ 4);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 5); tmp += F_MD5(&in_block, &hash, i+ 5);
                        // Perform MD5 sum
559                 write_pass(&in_block,   i+ 6); tmp += F_MD5(&in_block, &hash, i+ 6);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 7); tmp += F_MD5(&in_block, &hash, i+ 7);
                        // Perform MD5 sum
561                 write_pass(&in_block,   i+ 8); tmp += F_MD5(&in_block, &hash, i+ 8);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 9); tmp += F_MD5(&in_block, &hash, i+ 9);
                        // Perform MD5 sum
563
                    write_pass(&in_block,   i+10); tmp += F_MD5(&in_block, &hash, i+10);
                        // Perform MD5 sum
565                 write_pass(&in_block,   i+11); tmp += F_MD5(&in_block, &hash, i+11);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+12); tmp += F_MD5(&in_block, &hash, i+12);
                        // Perform MD5 sum
567                 write_pass(&in_block,   i+13); tmp += F_MD5(&in_block, &hash, i+13);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+14); tmp += F_MD5(&in_block, &hash, i+14);
                        // Perform MD5 sum
569                 write_pass(&in_block,   i+15); tmp += F_MD5(&in_block, &hash, i+15);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+16); tmp += F_MD5(&in_block, &hash, i+16);
                        // Perform MD5 sum
571                 write_pass(&in_block,   i+17); tmp += F_MD5(&in_block, &hash, i+17);
                        // Perform MD5 sum
```

```
                write_pass(&in_block,   i+18); tmp += F_MD5(&in_block, &hash, i+18);
                    // Perform MD5 sum
573             write_pass(&in_block,   i+19); tmp += F_MD5(&in_block, &hash, i+19);
                    // Perform MD5 sum

575             write_pass(&in_block,   i+20); tmp += F_MD5(&in_block, &hash, i+20);
                    // Perform MD5 sum
                write_pass(&in_block,   i+21); tmp += F_MD5(&in_block, &hash, i+21);
                    // Perform MD5 sum
577             write_pass(&in_block,   i+22); tmp += F_MD5(&in_block, &hash, i+22);
                    // Perform MD5 sum
                write_pass(&in_block,   i+23); tmp += F_MD5(&in_block, &hash, i+23);
                    // Perform MD5 sum
579             write_pass(&in_block,   i+24); tmp += F_MD5(&in_block, &hash, i+24);
                    // Perform MD5 sum
                write_pass(&in_block,   i+25); tmp += F_MD5(&in_block, &hash, i+25);
                    // Perform MD5 sum
581         }
            if (tmp != 0)
583         {
                write_first_pass(&in_block, tmp);
585             printf("found.␣%s\n", in_block._8);
            }
587     }
        #pragma omp section
589     {
            uint64_t tmp = 0;
591         for (i = 13 * N / _threads_; i < 14 * N / _threads_; i+=26)
            {
593             write_first_pass(&in_block,   i+ 0); tmp += G_MD5(&in_block, &hash,
                   i+ 0); // Perform MD5 sum
                write_pass(&in_block,   i+ 1); tmp += F_MD5(&in_block, &hash, i+ 1);
                    // Perform MD5 sum
595             write_pass(&in_block,   i+ 2); tmp += F_MD5(&in_block, &hash, i+ 2);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 3); tmp += F_MD5(&in_block, &hash, i+ 3);
                    // Perform MD5 sum
597             write_pass(&in_block,   i+ 4); tmp += F_MD5(&in_block, &hash, i+ 4);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 5); tmp += F_MD5(&in_block, &hash, i+ 5);
                    // Perform MD5 sum
599             write_pass(&in_block,   i+ 6); tmp += F_MD5(&in_block, &hash, i+ 6);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 7); tmp += F_MD5(&in_block, &hash, i+ 7);
                    // Perform MD5 sum
601             write_pass(&in_block,   i+ 8); tmp += F_MD5(&in_block, &hash, i+ 8);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 9); tmp += F_MD5(&in_block, &hash, i+ 9);
                    // Perform MD5 sum
603
                write_pass(&in_block,   i+10); tmp += F_MD5(&in_block, &hash, i+10);
                    // Perform MD5 sum
605             write_pass(&in_block,   i+11); tmp += F_MD5(&in_block, &hash, i+11);
                    // Perform MD5 sum
                write_pass(&in_block,   i+12); tmp += F_MD5(&in_block, &hash, i+12);
                    // Perform MD5 sum
```

```
607                 write_pass(&in_block,   i+13); tmp += F_MD5(&in_block, &hash, i+13);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+14); tmp += F_MD5(&in_block, &hash, i+14);
                        // Perform MD5 sum
609                 write_pass(&in_block,   i+15); tmp += F_MD5(&in_block, &hash, i+15);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+16); tmp += F_MD5(&in_block, &hash, i+16);
                        // Perform MD5 sum
611                 write_pass(&in_block,   i+17); tmp += F_MD5(&in_block, &hash, i+17);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+18); tmp += F_MD5(&in_block, &hash, i+18);
                        // Perform MD5 sum
613                 write_pass(&in_block,   i+19); tmp += F_MD5(&in_block, &hash, i+19);
                        // Perform MD5 sum

615                 write_pass(&in_block,   i+20); tmp += F_MD5(&in_block, &hash, i+20);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+21); tmp += F_MD5(&in_block, &hash, i+21);
                        // Perform MD5 sum
617                 write_pass(&in_block,   i+22); tmp += F_MD5(&in_block, &hash, i+22);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+23); tmp += F_MD5(&in_block, &hash, i+23);
                        // Perform MD5 sum
619                 write_pass(&in_block,   i+24); tmp += F_MD5(&in_block, &hash, i+24);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+25); tmp += F_MD5(&in_block, &hash, i+25);
                        // Perform MD5 sum
621             }
                if (tmp != 0)
623             {
                    write_first_pass(&in_block, tmp);
625                 printf("found.␣%s\n", in_block._8);
                }
627         }
            #pragma omp section
629         {
                uint64_t tmp = 0;
631             for (i = 14 * N / _threads_; i < 15 * N / _threads_; i+=26)
                {
633                 write_first_pass(&in_block,   i+ 0); tmp += G_MD5(&in_block, &hash,
                       i+ 0); // Perform MD5 sum
                    write_pass(&in_block,   i+ 1); tmp += F_MD5(&in_block, &hash, i+ 1);
                        // Perform MD5 sum
635                 write_pass(&in_block,   i+ 2); tmp += F_MD5(&in_block, &hash, i+ 2);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 3); tmp += F_MD5(&in_block, &hash, i+ 3);
                        // Perform MD5 sum
637                 write_pass(&in_block,   i+ 4); tmp += F_MD5(&in_block, &hash, i+ 4);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 5); tmp += F_MD5(&in_block, &hash, i+ 5);
                        // Perform MD5 sum
639                 write_pass(&in_block,   i+ 6); tmp += F_MD5(&in_block, &hash, i+ 6);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 7); tmp += F_MD5(&in_block, &hash, i+ 7);
                        // Perform MD5 sum
641                 write_pass(&in_block,   i+ 8); tmp += F_MD5(&in_block, &hash, i+ 8);
                        // Perform MD5 sum
```

```
                write_pass(&in_block,   i+ 9); tmp += F_MD5(&in_block, &hash, i+ 9);
                    // Perform MD5 sum

                write_pass(&in_block,   i+10); tmp += F_MD5(&in_block, &hash, i+10);
                    // Perform MD5 sum
                write_pass(&in_block,   i+11); tmp += F_MD5(&in_block, &hash, i+11);
                    // Perform MD5 sum
                write_pass(&in_block,   i+12); tmp += F_MD5(&in_block, &hash, i+12);
                    // Perform MD5 sum
                write_pass(&in_block,   i+13); tmp += F_MD5(&in_block, &hash, i+13);
                    // Perform MD5 sum
                write_pass(&in_block,   i+14); tmp += F_MD5(&in_block, &hash, i+14);
                    // Perform MD5 sum
                write_pass(&in_block,   i+15); tmp += F_MD5(&in_block, &hash, i+15);
                    // Perform MD5 sum
                write_pass(&in_block,   i+16); tmp += F_MD5(&in_block, &hash, i+16);
                    // Perform MD5 sum
                write_pass(&in_block,   i+17); tmp += F_MD5(&in_block, &hash, i+17);
                    // Perform MD5 sum
                write_pass(&in_block,   i+18); tmp += F_MD5(&in_block, &hash, i+18);
                    // Perform MD5 sum
                write_pass(&in_block,   i+19); tmp += F_MD5(&in_block, &hash, i+19);
                    // Perform MD5 sum

                write_pass(&in_block,   i+20); tmp += F_MD5(&in_block, &hash, i+20);
                    // Perform MD5 sum
                write_pass(&in_block,   i+21); tmp += F_MD5(&in_block, &hash, i+21);
                    // Perform MD5 sum
                write_pass(&in_block,   i+22); tmp += F_MD5(&in_block, &hash, i+22);
                    // Perform MD5 sum
                write_pass(&in_block,   i+23); tmp += F_MD5(&in_block, &hash, i+23);
                    // Perform MD5 sum
                write_pass(&in_block,   i+24); tmp += F_MD5(&in_block, &hash, i+24);
                    // Perform MD5 sum
                write_pass(&in_block,   i+25); tmp += F_MD5(&in_block, &hash, i+25);
                    // Perform MD5 sum
            }
            if (tmp != 0)
            {
                write_first_pass(&in_block, tmp);
                printf("found.␣%s\n", in_block._8);
            }
        }
        #pragma omp section
        {
            uint64_t tmp = 0;
            for (i = 15 * N / _threads_; i < 16 * N / _threads_; i+=26)
            {
                write_first_pass(&in_block,   i+ 0); tmp += G_MD5(&in_block, &hash,
                    i+ 0); // Perform MD5 sum
                write_pass(&in_block,   i+ 1); tmp += F_MD5(&in_block, &hash, i+ 1);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 2); tmp += F_MD5(&in_block, &hash, i+ 2);
                    // Perform MD5 sum
                write_pass(&in_block,   i+ 3); tmp += F_MD5(&in_block, &hash, i+ 3);
                    // Perform MD5 sum
```

```
677                 write_pass(&in_block,   i+ 4); tmp += F_MD5(&in_block, &hash, i+ 4);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 5); tmp += F_MD5(&in_block, &hash, i+ 5);
                        // Perform MD5 sum
679                 write_pass(&in_block,   i+ 6); tmp += F_MD5(&in_block, &hash, i+ 6);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 7); tmp += F_MD5(&in_block, &hash, i+ 7);
                        // Perform MD5 sum
681                 write_pass(&in_block,   i+ 8); tmp += F_MD5(&in_block, &hash, i+ 8);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+ 9); tmp += F_MD5(&in_block, &hash, i+ 9);
                        // Perform MD5 sum
683
                    write_pass(&in_block,   i+10); tmp += F_MD5(&in_block, &hash, i+10);
                        // Perform MD5 sum
685                 write_pass(&in_block,   i+11); tmp += F_MD5(&in_block, &hash, i+11);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+12); tmp += F_MD5(&in_block, &hash, i+12);
                        // Perform MD5 sum
687                 write_pass(&in_block,   i+13); tmp += F_MD5(&in_block, &hash, i+13);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+14); tmp += F_MD5(&in_block, &hash, i+14);
                        // Perform MD5 sum
689                 write_pass(&in_block,   i+15); tmp += F_MD5(&in_block, &hash, i+15);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+16); tmp += F_MD5(&in_block, &hash, i+16);
                        // Perform MD5 sum
691                 write_pass(&in_block,   i+17); tmp += F_MD5(&in_block, &hash, i+17);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+18); tmp += F_MD5(&in_block, &hash, i+18);
                        // Perform MD5 sum
693                 write_pass(&in_block,   i+19); tmp += F_MD5(&in_block, &hash, i+19);
                        // Perform MD5 sum

695                 write_pass(&in_block,   i+20); tmp += F_MD5(&in_block, &hash, i+20);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+21); tmp += F_MD5(&in_block, &hash, i+21);
                        // Perform MD5 sum
697                 write_pass(&in_block,   i+22); tmp += F_MD5(&in_block, &hash, i+22);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+23); tmp += F_MD5(&in_block, &hash, i+23);
                        // Perform MD5 sum
699                 write_pass(&in_block,   i+24); tmp += F_MD5(&in_block, &hash, i+24);
                        // Perform MD5 sum
                    write_pass(&in_block,   i+25); tmp += F_MD5(&in_block, &hash, i+25);
                        // Perform MD5 sum
701             }
                if (tmp != 0)
703             {
                    write_first_pass(&in_block, tmp);
705                 printf("found.␣%s\n", in_block._8);
                }
707         }
            //
709     }

711     printf("%lu␣hashes\n", N);
```

```
    return 0;
}

/*
    Optimization.
    Use first block every time.
    456976 unique length 4 passwords.
    Each one will be used 17576 times. (vectorised 4394)
    Make each loop 17576 long. The first
    line of the loop calculates the value and stores it.
    The next 17575 iterations skip that one.

    This can also be used for the write_pass function.
    The only values that need to be updated are [4-6]
*/
#ifdef LEN7
void write_pass(union Block *in_block, uint64_t i)
{
    in_block->_8[6] = alph(i % E1);
    i /= E1;
    in_block->_8[5] = alph(i % E1);
    i /= E1;
    in_block->_8[4] = alph(i % E1);
    i /= E1;

    // these are updated rarely.

    /*in_block->_8[7] = 0x80;
    in_block->_8[3] = alph(i % E1);
    i /= E1;
    in_block->_8[2] = alph(i % E1);
    i /= E1;
    in_block->_8[1] = alph(i % E1);
    i /= E1;
    in_block->_8[0] = alph(i % E1);
    i /= E1;
    */
    // memset(&in_block->_8[8], 0, 4); optimized out.
    // in_block->_32[3] = 56; optimized out.

}
void write_first_pass(union Block *in_block, uint64_t i)
{
    in_block->_8[6] = alph(i % E1);
    i /= E1;
    in_block->_8[5] = alph(i % E1);
    i /= E1;
    in_block->_8[4] = alph(i % E1);
    i /= E1;

    // these are updated rarely.
    in_block->_8[7] = 0x80;
    in_block->_8[3] = alph(i % E1);
    i /= E1;
    in_block->_8[2] = alph(i % E1);
    i /= E1;
    in_block->_8[1] = alph(i % E1);
```

```
769     i /= E1;
        in_block->_8[0] = alph(i % E1);
771     i /= E1;
        // memset(&in_block->_8[8], 0, 4); optimized out.
773     // in_block->_32[3] = 56; optimized out.

775 }
    #endif
777 #ifndef LEN7
    /*
779     Password generator.
        Accepts a block and iterator value,
781     Outputs a password of length N (macro)

783     Implemented as a binary search for speed.

785     Future optimization: write in 32 bit words.

787     Is there a better way to evenly generate these passwords?
    */
789 void write_pass(union Block *in_block, uint64_t i)
    {
791     int len;
        // if (i < D1)
793     // else if (i < D2)
        // else if (i < D3)
795     // else if (i < D4)
        // else if (i < D5)
797     // else if (i < D6)
        // else if (i < D7)
799     // else if (i < D8)
        // else if (i < D9)
801     // else

803     if (i < D5)
        {
805         if (i < D2)
            {
807             if (i < D1) // D1
                {
809                 in_block->_8[0] = alph(i);
                    len = 1;
811                 in_block->_8[len] = 0x80;
                    memset(&in_block->_8[len + 1], 0, 11 - len);
813                 in_block->_32[3] = len << 3;
                }
815             else // D2
                {
817                 in_block->_8[0] = alph((i - D1) % E1);
                    in_block->_8[1] = alph((i - D1) / E1);
819                 len = 2;
                    in_block->_8[len] = 0x80;
821                 memset(&in_block->_8[len + 1], 0, 11 - len);
                    in_block->_32[3] = len << 3;
823             }

825         }
```

```
        else
827     {
            if (i < D4)
829         {
                if (i < D3) // D3
831             {
                    in_block->_8[0] = alph((i - D2) % E1);
833                 in_block->_8[1] = alph((((i - D2) / E1) % E1) );
                    in_block->_8[2] = alph((((i - D2) / E2) % E1));
835                 len = 3;
                    in_block->_8[len] = 0x80;
837                 memset(&in_block->_8[len + 1], 0, 11 - len);
                    in_block->_32[3] = len << 3;
839             }

841             else // D4
                {
843                 in_block->_8[0] = alph((i - D3) % E1);
                    in_block->_8[1] = alph((((i - D3) / E1) % E1));
845                 in_block->_8[2] = alph((((i - D3) / E2) % E1));
                    in_block->_8[3] = alph((((i - D3) / E3) % E1));
847                 len = 4;
                    in_block->_8[len] = 0x80;
849                 memset(&in_block->_8[len + 1], 0, 11 - len);
                    in_block->_32[3] = len << 3;
851             }

853         }
            else      // D5
855         {
                in_block->_8[0] = alph((i - D4) % E1);
857             in_block->_8[1] = alph((((i - D4) / E1) % E1));
                in_block->_8[2] = alph((((i - D4) / E2) % E1));
859             in_block->_8[3] = alph((((i - D4) / E3) % E1));
                in_block->_8[4] = alph((((i - D4) / E4) % E1));
861             len = 5;
                in_block->_8[len] = 0x80;
863             memset(&in_block->_8[len + 1], 0, 11 - len);
                in_block->_32[3] = len << 3;
865         }

867     }
    }
869 else
    {
871     if (i < D7)
        {
873         if (i < D6) // D6
            {
875             in_block->_8[0] = alph((i - D5) % E1);
                in_block->_8[1] = alph((((i - D5) / E1) % E1));
877             in_block->_8[2] = alph((((i - D5) / E2) % E1));
                in_block->_8[3] = alph((((i - D5) / E3) % E1));
879             in_block->_8[4] = alph((((i - D5) / E4) % E1));
                in_block->_8[5] = alph((((i - D5) / E5) % E1));
881             len = 6;
                in_block->_8[len] = 0x80;
```

```
                memset(&in_block->_8[len + 1], 0, 11 - len);
                in_block->_32[3] = len << 3;
            }

            else // D7
            {
                in_block->_8[0] = alph((i - D6) % E1);
                in_block->_8[1] = alph((((i - D6) / E1) % E1));
                in_block->_8[2] = alph((((i - D6) / E2) % E1));
                in_block->_8[3] = alph((((i - D6) / E3) % E1));
                in_block->_8[4] = alph((((i - D6) / E4) % E1));
                in_block->_8[5] = alph((((i - D6) / E5) % E1));
                in_block->_8[6] = alph((((i - D6) / E6) % E1));
                len = 7;
                in_block->_8[len] = 0x80;
                memset(&in_block->_8[len + 1], 0, 11 - len);
                in_block->_32[3] = len << 3;
            }

        }
        else // D8-10
        {
            if (i < D9)
            {
                if (i < D8) // D8
                {
                    in_block->_8[0] = alph((i - D7) % E1);
                    in_block->_8[1] = alph((((i - D7) / E1) % E1));
                    in_block->_8[2] = alph((((i - D7) / E2) % E1));
                    in_block->_8[3] = alph((((i - D7) / E3) % E1));
                    in_block->_8[4] = alph((((i - D7) / E4) % E1));
                    in_block->_8[5] = alph((((i - D7) / E5) % E1));
                    in_block->_8[6] = alph((((i - D7) / E6) % E1));
                    in_block->_8[7] = alph((((i - D7) / E7) % E1));
                    len = 8;
                    in_block->_8[len] = 0x80;
                    memset(&in_block->_8[len + 1], 0, 11 - len);
                    in_block->_32[3] = len << 3;
                }

                else // D9
                {
                    in_block->_8[0] = alph((i - D8) % E1);
                    in_block->_8[1] = alph((((i - D8) / E1) % E1));
                    in_block->_8[2] = alph((((i - D8) / E2) % E1));
                    in_block->_8[3] = alph((((i - D8) / E3) % E1));
                    in_block->_8[4] = alph((((i - D8) / E4) % E1));
                    in_block->_8[5] = alph((((i - D8) / E5) % E1));
                    in_block->_8[6] = alph((((i - D8) / E6) % E1));
                    in_block->_8[7] = alph((((i - D8) / E7) % E1));
                    in_block->_8[8] = alph((((i - D8) / E8) % E1));
                    len = 9;
                    in_block->_8[len] = 0x80;
                    memset(&in_block->_8[len + 1], 0, 11 - len);
                    in_block->_32[3] = len << 3;
                }

```

```
            }
            else // D10
            {
                in_block->_8[0] = alph((i - D9) % E1);
                in_block->_8[1] = alph((((i - D9) / E1) % E1));
                in_block->_8[2] = alph((((i - D9) / E2) % E1));
                in_block->_8[3] = alph((((i - D9) / E3) % E1));
                in_block->_8[4] = alph((((i - D9) / E4) % E1));
                in_block->_8[5] = alph((((i - D9) / E5) % E1));
                in_block->_8[6] = alph((((i - D9) / E6) % E1));
                in_block->_8[7] = alph((((i - D9) / E7) % E1));
                in_block->_8[8] = alph((((i - D9) / E8) % E1));
                in_block->_8[9] = alph((((i - D9) / E9) % E1));
                len = 10;
                in_block->_8[len] = 0x80;
                memset(&in_block->_8[len + 1], 0, 11 - len);
                in_block->_32[3] = len << 3;
            }

        }
    }
}
#endif

uint64_t F_MD5(union Block *bl, union Hash *ha, uint64_t i)
{
    register uint32_t a, b, c, d;


    // a = HASH_BASE_A;
    // removed for optimization, see FF1 function

    b = HASH_BASE_B;
    c = HASH_BASE_C;
    d = HASH_BASE_D;


    /*
        Optimization. We aren't ever hashing multiple times.
        This means that we can use constants here. Only assign
        the hash value at the very end.
        Done.
    */

    /*
        Optimization. Instead of assign a=val; Just create a
        FF1 function that has the values hard coded into it.
        Done.
    */


    /*
        Optimization. Our passwords will only have 2 things change:
            1. initial 10 characters + padding. bl->32[0-3]
            2. final 64 bits. Really just bl->32[14]
        Hardcode every other value as a 0.
        Done.
```

```
997     */
999     {
            /* Round 1 */
1001        a=bl->_32[3];
            FF (d, a, b, c, bl->_32[ 1], S12, 0xe8c7b756); /* 2 */
1003    #ifndef LEN7
            FF (c, d, a, b, bl->_32[ 2], S13, 0x242070db); /* 3 */
1005    #endif
        #ifdef LEN7
1007        FF (c, d, a, b,           0, S13, 0x242070db); /* 3 */
        #endif
1009
            FF (b, c, d, a,           0, S14, 0xc1bdceee); /* 4 */
1011        FF (a, b, c, d,           0, S11, 0xf57c0faf); /* 5 */
            FF (d, a, b, c,           0, S12, 0x4787c62a); /* 6 */
1013        FF (c, d, a, b,           0, S13, 0xa8304613); /* 7 */
            FF (b, c, d, a,           0, S14, 0xfd469501); /* 8 */
1015        FF (a, b, c, d,           0, S11, 0x698098d8); /* 9 */
            FF (d, a, b, c,           0, S12, 0x8b44f7af); /* 10 */
1017        FF (c, d, a, b,           0, S13, 0xffff5bb1); /* 11 */
            FF (b, c, d, a,           0, S14, 0x895cd7be); /* 12 */
1019        FF (a, b, c, d,           0, S11, 0x6b901122); /* 13 */
            FF (d, a, b, c,           0, S12, 0xfd987193); /* 14 */
1021    #ifndef LEN7
            FF (c, d, a, b, bl->_32[ 3], S13, 0xa679438e); /* 15 */
1023    #endif
        #ifdef LEN7
1025        FF (c, d, a, b,          56, S13, 0xa679438e); /* 15 */
        #endif
1027        FF (b, c, d, a,           0, S14, 0x49b40821); /* 16 */
1029        /* Round 2 */
            GG (a, b, c, d, bl->_32[ 1], S21, 0xf61e2562); /* 17 */
1031        GG (d, a, b, c,           0, S22, 0xc040b340); /* 18 */
            GG (c, d, a, b,           0, S23, 0x265e5a51); /* 19 */
1033        GG (b, c, d, a, bl->_32[ 0], S24, 0xe9b6c7aa); /* 20 */
            GG (a, b, c, d,           0, S21, 0xd62f105d); /* 21 */
1035        GG (d, a, b, c,           0, S22,  0x2441453); /* 22 */
            GG (c, d, a, b,           0, S23, 0xd8a1e681); /* 23 */
1037        GG (b, c, d, a,           0, S24, 0xe7d3fbc8); /* 24 */
            GG (a, b, c, d,           0, S21, 0x21e1cde6); /* 25 */
1039    #ifndef LEN7
            GG (d, a, b, c, bl->_32[ 3], S22, 0xc33707d6); /* 26 */
1041    #endif
        #ifdef LEN7
1043        GG (d, a, b, c,          56, S22, 0xc33707d6); /* 26 */
        #endif
1045        GG (c, d, a, b,           0, S23, 0xf4d50d87); /* 27 */
            GG (b, c, d, a,           0, S24, 0x455a14ed); /* 28 */
1047        GG (a, b, c, d,           0, S21, 0xa9e3e905); /* 29 */
        #ifndef LEN7
1049        GG (d, a, b, c, bl->_32[ 2], S22, 0xfcefa3f8); /* 30 */
        #endif
1051    #ifdef LEN7
            GG (d, a, b, c,           0, S22, 0xfcefa3f8); /* 30 */
1053    #endif
```

```
1055        GG (c, d, a, b,           0, S23, 0x676f02d9); /* 31 */
            GG (b, c, d, a,           0, S24, 0x8d2a4c8a); /* 32 */
1057
            /* Round 3 */
1059        HH (a, b, c, d,           0, S31, 0xfffa3942); /* 33 */
            HH (d, a, b, c,           0, S32, 0x8771f681); /* 34 */
1061        HH (c, d, a, b,           0, S33, 0x6d9d6122); /* 35 */
#ifndef LEN7
1063        HH (b, c, d, a, bl->_32[ 3], S34, 0xfde5380c); /* 36 */
#endif
1065 #ifdef LEN7
            HH (b, c, d, a,          56, S34, 0xfde5380c); /* 36 */
1067 #endif
            HH (a, b, c, d, bl->_32[ 1], S31, 0xa4beea44); /* 37 */
1069        HH (d, a, b, c,           0, S32, 0x4bdecfa9); /* 38 */
            HH (c, d, a, b,           0, S33, 0xf6bb4b60); /* 39 */
1071        HH (b, c, d, a,           0, S34, 0xbebfbc70); /* 40 */
            HH (a, b, c, d,           0, S31, 0x289b7ec6); /* 41 */
1073        HH (d, a, b, c, bl->_32[ 0], S32, 0xeaa127fa); /* 42 */
            HH (c, d, a, b,           0, S33, 0xd4ef3085); /* 43 */
1075        HH (b, c, d, a,           0, S34, 0x04881d05); /* 44 */
            HH (a, b, c, d,           0, S31, 0xd9d4d039); /* 45 */
1077        HH (d, a, b, c,           0, S32, 0xe6db99e5); /* 46 */
            HH (c, d, a, b,           0, S33, 0x1fa27cf8); /* 47 */
1079 #ifndef LEN7
            HH (b, c, d, a, bl->_32[ 2], S34, 0xc4ac5665); /* 48 */
1081 #endif
    #ifdef LEN7
1083        HH (b, c, d, a,           0, S34, 0xc4ac5665); /* 48 */
    #endif
1085

1087        /* Round 4 */
            II (a, b, c, d, bl->_32[ 0], S41, 0xf4292244); /* 49 */
1089        II (d, a, b, c,           0, S42, 0x432aff97); /* 50 */
    #ifndef LEN7
1091        II (c, d, a, b, bl->_32[ 3], S43, 0xab9423a7); /* 51 */
    #endif
1093 #ifdef LEN7
            II (c, d, a, b,          56, S43, 0xab9423a7); /* 51 */
1095 #endif
            II (b, c, d, a,           0, S44, 0xfc93a039); /* 52 */
1097        II (a, b, c, d,           0, S41, 0x655b59c3); /* 53 */
            II (d, a, b, c,           0, S42, 0x8f0ccc92); /* 54 */
1099        II (c, d, a, b,           0, S43, 0xffeff47d); /* 55 */
            II (b, c, d, a, bl->_32[ 1], S44, 0x85845dd1); /* 56 */
1101        II (a, b, c, d,           0, S41, 0x6fa87e4f); /* 57 */
            II (d, a, b, c,           0, S42, 0xfe2ce6e0); /* 58 */
1103        II (c, d, a, b,           0, S43, 0xa3014314); /* 59 */
            II (b, c, d, a,           0, S44, 0x4e0811a1); /* 60 */
1105        II (a, b, c, d,           0, S41, 0xf7537e82); /* 61 */
            II (d, a, b, c,           0, S42, 0xbd3af235); /* 62 */
1107 #ifndef LEN7
            II (c, d, a, b, bl->_32[ 2], S43, 0x2ad7d2bb); /* 63 */
1109 #endif
    #ifdef LEN7
```

```
        II (c, d, a, b,           0, S43, 0x2ad7d2bb); /* 63 */
#endif

        II (b, c, d, a,           0, S44, 0xeb86d391); /* 64 */
    }
    ha->_32[0] = a + HASH_BASE_A;
    ha->_32[1] = b + HASH_BASE_B;
    ha->_32[2] = c + HASH_BASE_C;
    ha->_32[3] = d + HASH_BASE_D;


    /*
        Optimization. Use the return-style sequence
        in order to save 2.6%. 2:30 compared to 2:26.
        uint64_t cmp1, cmp2;
        cmp1 = ha->_64[0] ^ enigma._64[0];
        cmp2 = ha->_64[1] ^ enigma._64[1];

        if ((cmp1 | cmp2) == 0)
        {
            printf("Found the hash!\n");
            printf("  %s\n", (char *)bl->_8);
        }


        Equivalent to
        if( found )
            return i;
        else return 0;
        misses if the password is 'aaaaaaa'
    */

    return i * ((
                    (ha->_32[0] ^ enigma._32[0]) |
                    (ha->_32[1] ^ enigma._32[1]) |
                    (ha->_32[2] ^ enigma._32[2]) |
                    (ha->_32[3] ^ enigma._32[3])
                ) == 0);
    /*
        Optimization
        Test if it matches the default hash. Then we can
        return a boolean value or just output and kill here
    */
    /*
        Optimization.
        Instead of testing, try the following sequence.
        tmp += i* ((ha^enigma)==0);
        return tmp;

        This removes all jumps, and gives an index at the end.
    */
}


uint64_t G_MD5(union Block *bl, union Hash *ha, uint64_t i)
{
    register uint32_t a, b, c, d;
```

```
1169
    // a = HASH_BASE_A;
1171    // removed for optimization, see FF1 function

1173    b = HASH_BASE_B;
    c = HASH_BASE_C;
1175    d = HASH_BASE_D;

1177
    /*
1179        Optimization. We aren't ever hashing multiple times.
        This means that we can use constants here. Only assign
1181        the hash value at the very end.
        Done.
1183    */

1185    /*
        Optimization. Instead of assign a=val; Just create a
1187        FF1 function that has the values hard coded into it.
        Done.
1189    */

1191
    /*
1193        Optimization. Our passwords will only have 2 things change:
            1. initial 10 characters + padding. bl->32[0-3]
1195            2. final 64 bits. Really just bl->32[14]
        Hardcode every other value as a 0.
1197        Done.
    */

1199
    {
1201        /* Round 1 */

1203        FF1(bl->_32[3], b, c, d, bl->_32[ 0], S11, 0xd76aa478); /* 1 */
        a=bl->_32[3];
1205        FF (d, a, b, c, bl->_32[ 1], S12, 0xe8c7b756); /* 2 */
#ifndef LEN7
1207        FF (c, d, a, b, bl->_32[ 2], S13, 0x242070db); /* 3 */
#endif
1209 #ifdef LEN7
        FF (c, d, a, b,           0, S13, 0x242070db); /* 3 */
1211 #endif

1213        FF (b, c, d, a,           0, S14, 0xc1bdceee); /* 4 */
        FF (a, b, c, d,           0, S11, 0xf57c0faf); /* 5 */
1215        FF (d, a, b, c,           0, S12, 0x4787c62a); /* 6 */
        FF (c, d, a, b,           0, S13, 0xa8304613); /* 7 */
1217        FF (b, c, d, a,           0, S14, 0xfd469501); /* 8 */
        FF (a, b, c, d,           0, S11, 0x698098d8); /* 9 */
1219        FF (d, a, b, c,           0, S12, 0x8b44f7af); /* 10 */
        FF (c, d, a, b,           0, S13, 0xffff5bb1); /* 11 */
1221        FF (b, c, d, a,           0, S14, 0x895cd7be); /* 12 */
        FF (a, b, c, d,           0, S11, 0x6b901122); /* 13 */
1223        FF (d, a, b, c,           0, S12, 0xfd987193); /* 14 */
#ifndef LEN7
```

```
1225         FF (c, d, a, b, bl->_32[ 3], S13, 0xa679438e); /* 15 */
#endif
1227 #ifdef LEN7
         FF (c, d, a, b,           56, S13, 0xa679438e); /* 15 */
1229 #endif
         FF (b, c, d, a,            0, S14, 0x49b40821); /* 16 */
1231
         /* Round 2 */
1233         GG (a, b, c, d, bl->_32[ 1], S21, 0xf61e2562); /* 17 */
         GG (d, a, b, c,            0, S22, 0xc040b340); /* 18 */
1235         GG (c, d, a, b,            0, S23, 0x265e5a51); /* 19 */
         GG (b, c, d, a, bl->_32[ 0], S24, 0xe9b6c7aa); /* 20 */
1237         GG (a, b, c, d,            0, S21, 0xd62f105d); /* 21 */
         GG (d, a, b, c,            0, S22,  0x2441453); /* 22 */
1239         GG (c, d, a, b,            0, S23, 0xd8a1e681); /* 23 */
         GG (b, c, d, a,            0, S24, 0xe7d3fbc8); /* 24 */
1241         GG (a, b, c, d,            0, S21, 0x21e1cde6); /* 25 */
#ifndef LEN7
1243         GG (d, a, b, c, bl->_32[ 3], S22, 0xc33707d6); /* 26 */
#endif
1245 #ifdef LEN7
         GG (d, a, b, c,           56, S22, 0xc33707d6); /* 26 */
1247 #endif
         GG (c, d, a, b,            0, S23, 0xf4d50d87); /* 27 */
1249         GG (b, c, d, a,            0, S24, 0x455a14ed); /* 28 */
         GG (a, b, c, d,            0, S21, 0xa9e3e905); /* 29 */
1251 #ifndef LEN7
         GG (d, a, b, c, bl->_32[ 2], S22, 0xfcefa3f8); /* 30 */
1253 #endif
#ifdef LEN7
1255         GG (d, a, b, c,            0, S22, 0xfcefa3f8); /* 30 */
#endif
1257
         GG (c, d, a, b,            0, S23, 0x676f02d9); /* 31 */
1259         GG (b, c, d, a,            0, S24, 0x8d2a4c8a); /* 32 */

1261         /* Round 3 */
         HH (a, b, c, d,            0, S31, 0xfffa3942); /* 33 */
1263         HH (d, a, b, c,            0, S32, 0x8771f681); /* 34 */
         HH (c, d, a, b,            0, S33, 0x6d9d6122); /* 35 */
1265 #ifndef LEN7
         HH (b, c, d, a, bl->_32[ 3], S34, 0xfde5380c); /* 36 */
1267 #endif
#ifdef LEN7
1269         HH (b, c, d, a,           56, S34, 0xfde5380c); /* 36 */
#endif
1271         HH (a, b, c, d, bl->_32[ 1], S31, 0xa4beea44); /* 37 */
         HH (d, a, b, c,            0, S32, 0x4bdecfa9); /* 38 */
1273         HH (c, d, a, b,            0, S33, 0xf6bb4b60); /* 39 */
         HH (b, c, d, a,            0, S34, 0xbebfbc70); /* 40 */
1275         HH (a, b, c, d,            0, S31, 0x289b7ec6); /* 41 */
         HH (d, a, b, c, bl->_32[ 0], S32, 0xeaa127fa); /* 42 */
1277         HH (c, d, a, b,            0, S33, 0xd4ef3085); /* 43 */
         HH (b, c, d, a,            0, S34, 0x04881d05); /* 44 */
1279         HH (a, b, c, d,            0, S31, 0xd9d4d039); /* 45 */
         HH (d, a, b, c,            0, S32, 0xe6db99e5); /* 46 */
1281         HH (c, d, a, b,            0, S33, 0x1fa27cf8); /* 47 */
```

```
#ifndef LEN7
1283        HH (b, c, d, a, bl->_32[ 2], S34, 0xc4ac5665); /* 48 */
#endif
1285 #ifdef LEN7
        HH (b, c, d, a,           0, S34, 0xc4ac5665); /* 48 */
1287 #endif


1289

        /* Round 4 */
1291        II (a, b, c, d, bl->_32[ 0], S41, 0xf4292244); /* 49 */
        II (d, a, b, c,           0, S42, 0x432aff97); /* 50 */
1293 #ifndef LEN7
        II (c, d, a, b, bl->_32[ 3], S43, 0xab9423a7); /* 51 */
1295 #endif
#ifdef LEN7
1297        II (c, d, a, b,          56, S43, 0xab9423a7); /* 51 */
#endif
1299        II (b, c, d, a,           0, S44, 0xfc93a039); /* 52 */
        II (a, b, c, d,           0, S41, 0x655b59c3); /* 53 */
1301        II (d, a, b, c,           0, S42, 0x8f0ccc92); /* 54 */
        II (c, d, a, b,           0, S43, 0xffeff47d); /* 55 */
1303        II (b, c, d, a, bl->_32[ 1], S44, 0x85845dd1); /* 56 */
        II (a, b, c, d,           0, S41, 0x6fa87e4f); /* 57 */
1305        II (d, a, b, c,           0, S42, 0xfe2ce6e0); /* 58 */
        II (c, d, a, b,           0, S43, 0xa3014314); /* 59 */
1307        II (b, c, d, a,           0, S44, 0x4e0811a1); /* 60 */
        II (a, b, c, d,           0, S41, 0xf7537e82); /* 61 */
1309        II (d, a, b, c,           0, S42, 0xbd3af235); /* 62 */
#ifndef LEN7
1311        II (c, d, a, b, bl->_32[ 2], S43, 0x2ad7d2bb); /* 63 */
#endif
1313 #ifdef LEN7
        II (c, d, a, b,           0, S43, 0x2ad7d2bb); /* 63 */
1315 #endif

1317        II (b, c, d, a,           0, S44, 0xeb86d391); /* 64 */
    }
1319    ha->_32[0] = a + HASH_BASE_A;
    ha->_32[1] = b + HASH_BASE_B;
1321    ha->_32[2] = c + HASH_BASE_C;
    ha->_32[3] = d + HASH_BASE_D;
1323


1325    /*
        Optimization. Use the return-style sequence
1327        in order to save 2.6%. 2:30 compared to 2:26.
        uint64_t cmp1, cmp2;
1329        cmp1 = ha->_64[0] ^ enigma._64[0];
        cmp2 = ha->_64[1] ^ enigma._64[1];
1331

        if ((cmp1 | cmp2) == 0)
1333        {
            printf("Found the hash!\n");
1335            printf("  %s\n", (char *)bl->_8);
        }
1337
```

```
1339        Equivalent to
            if( found )
1341            return i;
            else return 0;
1343        misses if the password is 'aaaaaaa'
    */
1345
    return i * ((
1347                    (ha->_32[0] ^ enigma._32[0]) |
                        (ha->_32[1] ^ enigma._32[1]) |
1349                    (ha->_32[2] ^ enigma._32[2]) |
                        (ha->_32[3] ^ enigma._32[3])
1351                ) == 0);
    /*
1353        Optimization
            Test if it matches the default hash. Then we can
1355        return a boolean value or just output and kill here
    */
1357    /*
            Optimization.
1359        Instead of testing, try the following sequence.
            tmp += i* ((ha^enigma)==0);
1361        return tmp;

1363        This removes all jumps, and gives an index at the end.
    */
1365 }
```

3.1.2 Header