Eric Watson

ECE 335 Homework 5

This assignment required us to implement three routines, following code provided to us in the textbook, and then measure the execution times of each function. Doing so allowed us to gain insight into how the software design can significantly impact execution time. The textbook code used several additional functions to operate, in order to eliminate the need for these functions we were required to tailor the routines to accept as parameters: the array address, the address of the destination, and the array length.

In order to perform a full investigation of execution times, I tested the three functions with various array sizes. The table below displays the results of my tests for various array sizes:

Execution Times (In microseconds)

Array Size	**500**	**1000**	**5000**	**10000**	**15000**
Combine3	3	7	23	43	66
Combine5	2	3	15	38	48
Combine7	0	2	11	22	33

Looking at the table above, one can see that the execution times of Combine7 are the fastest for each of the array sizes tested, followed by Combine5, and then Combine3. These different implementations lead to different execution times based on their use of memory access and loop optimization. Combine3 addresses the array directly, which is an improvement in performance to the previous examples in the book, but is a characteristic we have included in our Combine5 and Combine7 for this experiment. Combine5 introduces loop unrolling, a program transformation that reduces the number of iterations for a loop by increasing the number of elements computed on each iteration. Within the Combine5 routine we loop unroll by a factor of two, which reduces some loop overhead, thereby giving us better performance. Combine7 takes it a step further to achieve the best performance of the three routines. This function loop unrolls by a factor of 2 once again, but then re-associates the combining operation within the first loop. This approach increases the number of operations that can be performed in parallel, thus provided us with the fastest execution time.

The terminal output below demonstrates the full functionality of my program. For debugging purposes I am initially testing each routine with an array size of 10. Due to the small array size I can print each randomly generated value without creating a lengthy output. After executing each routine I print out the execution time and the combination value in each corresponding destination array, to ensure that all functions are working properly (computing the same product).

```
Erics-MacBook-Pro-2:HW05_Code Watson$ gcc HW05.c
Erics-MacBook-Pro-2:HW05_Code Watson$ ./a.out
1.092800
0.961940
1.097160
1.028340
1.008140
0.949960
1.092780
1.019660
0.923800
0.977960

Combine 3 Execution Time: 0.000003 seconds
Combination Value: 1.143428

Combine 5 Execution Time: 0.000002 seconds
Combination Value: 1.143428

Combine 7 Execution Time: 0.000000 seconds
Combination Value: 1.143428
```

Testing

The following terminal captures display the program outputs upon testing various array sizes. (These results are organized in the table above).

Array Size: 500

```
Erics-MacBook-Pro-2:HW05_Code Watson$ gcc HW05.c
Erics-MacBook-Pro-2:HW05_Code Watson$ ./a.out

Combine 3 Execution Time: 0.000003 seconds
Combination Value: 3.294029

Combine 5 Execution Time: 0.000003 seconds
Combination Value: 3.294029

Combine 7 Execution Time: 0.000002 seconds
Combination Value: 3.294029
```

Array Size: 1000

```
Erics-MacBook-Pro-2:HW05_Code Watson$ gcc HW05.c
Erics-MacBook-Pro-2:HW05_Code Watson$ ./a.out

Combine 3 Execution Time: 0.000007 seconds
Combination Value: 0.075704

Combine 5 Execution Time: 0.000003 seconds
Combination Value: 0.075704

Combine 7 Execution Time: 0.000002 seconds
Combination Value: 0.075704
```

Array Size: 5000

```
Erics-MacBook-Pro-2:HW05_Code Watson$ gcc HW05.c
Erics-MacBook-Pro-2:HW05_Code Watson$ ./a.out

Combine 3 Execution Time: 0.000023 seconds
Combination Value: 0.002270

Combine 5 Execution Time: 0.000015 seconds
Combination Value: 0.002270

Combine 7 Execution Time: 0.000011 seconds
Combination Value: 0.002270
```

Array Size: 10,000

```
Erics-MacBook-Pro-2:HW05_Code Watson$ gcc HW05.c
Erics-MacBook-Pro-2:HW05_Code Watson$ ./a.out

Combine 3 Execution Time: 0.000043 seconds
Combination Value: 0.000001

Combine 5 Execution Time: 0.000038 seconds
Combination Value: 0.000001

Combine 7 Execution Time: 0.000022 seconds
Combination Value: 0.000001
```

Array Size: 15,000

```
Erics-MacBook-Pro-2:HW05_Code Watson$ gcc HW05.c
Erics-MacBook-Pro-2:HW05_Code Watson$ ./a.out

Combine 3 Execution Time: 0.000066 seconds
Combination Value: 0.000000

Combine 5 Execution Time: 0.000048 seconds
Combination Value: 0.000000

Combine 7 Execution Time: 0.000033 seconds
Combination Value: 0.000000
```

Eric Watson

ECE 335 Homework 5 Source Code (HW05.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#define LENGTH 500
#define K 10000

void print_first_ten(double A[]){
  int i;
  for (i =0; i < 10; i++){
    printf("%f\n", A[i]);
  }
}

void print_dest(double A[]){
    printf("Combination Value: %f\n\n", A[0]);
}

void combine3(double * A, double * dest, int length){
  int i;
  *dest = 1.0;
  for (i=0; i < length; i++){
    *dest = *dest * A[i];
  }
}

void combine5(double * A, double * dest, int length){
  int i;
  int limit = length-1;
  double acc = 1.0;
// Combine 2 elements at a time
  for (i = 0; i < limit; i+=2){
    acc = (acc * A[i]) * A[i+1];
  }
// Finish any remaining elements
  for (;i < length; i++){
    acc = acc * A[i];
  }
  *dest = acc;
}

void combine7(double * A, double * dest, int length){
  int i;
  int limit = length-1;
  double acc = 1.0;
// Combine 2 elements at a time
  for (i=0; i < limit; i+=2){
    acc = acc * (A[i] * A[i+1]);
  }
// Finish any remaining elements
  for (; i < length; i++){
    acc = acc * A[i];
  }
  *dest = acc;
}

int main(){
  int i;
  clock_t begin_three, end_three, begin_five, end_five, begin_seven, end_seven;
  double three_time;
  double five_time;
  double seven_time;
  time_t seconds;
  time(&seconds);
  srand((unsigned int) seconds);
  double A[LENGTH];   //input matrix
  double dest3[1];
  double dest5[1];
  double dest7[1];

  for (i=0; i < LENGTH; i++)    // initialize the array elements
  A[i] = 0.90 + (double)(rand()% K)/50000;

/* DEBUGGING: Testing with small number of values to ensure that
              that the combination functions were working correctly
    print_first_ten(A);
*/
    begin_three = clock();            // start clock
    combine3(A,dest3,LENGTH);
    end_three = clock();              // end clock
    three_time = (double)(end_three-begin_three)/ CLOCKS_PER_SEC;
    printf("\n");
    printf("Combine 3 Execution Time: %f seconds\n", three_time);
    print_dest(dest3);

    begin_five = clock();            // start clock
    combine5(A, dest5, LENGTH);
    end_five = clock();              // end clock
    five_time = (double)(end_five-begin_five)/ CLOCKS_PER_SEC;
    printf("Combine 5 Execution Time: %f seconds\n", five_time);
    print_dest(dest5);

    begin_seven = clock();            // start clock
    combine7(A, dest7, LENGTH);
    end_seven = clock();              // end clock
    seven_time = (double)(end_seven-begin_seven)/ CLOCKS_PER_SEC;
    printf("Combine 7 Execution Time: %f seconds\n", seven_time);
    print_dest(dest7);
}
```