Lab Report

ECPE 170 – Computer Systems and Networks – Spring 2017

Name: STEVE GUERRERO

Lab Topic: PERFORMANCE OPTIMIZATION (LAB #: 06)

What is the total physical RAM installed in the system? (In MB)

16,178 MB

(2) With no applications running (beyond the web browser with this page), how much RAM is used by the

native operating system? (e.g. Windows)

3.8 GB

(3) With no applications running (beyond the web browser with this page), how much RAM is available?

12 GB

(4) Check the virtual machine configuration. How much RAM is currently allocated to Linux in your virtual machine?
Note: Your answer to question 4 must be less than your answer to question 3!   Otherwise, your system will use slow virtual memory (i.e. swapping data to the hard disk) when running this lab.

1982 MB

(5) Try to increase your virtual machine memory allocation, if possible, to the maximum allowed based on your free RAM. Leave ~256MB free for the virtual machine program itself. Now how much RAM is allocated to Linux in your virtual machine?

I allocated 4096 MB, now there is 2511 MB free.

(6) Boot Linux. With no applications running in Linux, how much RAM is available inside the virtual machine? The "System Monitor" program should report that information. This is the space that is actually available for our test application.

The system is using 1.1 GB of 3.8 GB, so that leaves 2.7 GB free.

(7) What is the code doing? (Describe the algorithm in a paragraph, focusing on
the combine1() function.)

Depending on what operation command (SUM or PROD) and data type (FLOAT, INT, CHAR, DOUBLE, etc.) you define in config.h, the function can sum up or multiply the specific data types that the allocated array consists of.

(8) What is the largest number of elements that the vector can hold WITHOUT using swap storage (virtual memory), and how much memory does it take? Be sure to leave enough memory
for Firefox and LibreOffice, since you'll need those when running this lab as well.edit

Tip: The program reports memory usage when run. If you can compile and run it, you can experiment with the size of the vector to confirm your answer. If you see your disk activity light grinding away (either the physical LED, or the on-screen indicator in your virtual machine), you
have chosen poorly... :-(
Tip 2: Run the System Monitor program in Linux in the background, then run your program. You can watch the Memory Usage indicator in the monitor increase when the program is running. The goal is to reach approximately 85% of memory, but not to start using swap memory.

Linux is utilizing approximately 1.8 GB of 3.8 GB of allocated memory so I'd only have 2.0 GB of free memory to use to store the vector. Using a vector size of 460000000 allocated 1754.76 MB using 89.6% of the total system memory. The use of swap memory was not needed in this case. The program took 1.62 seconds to run.

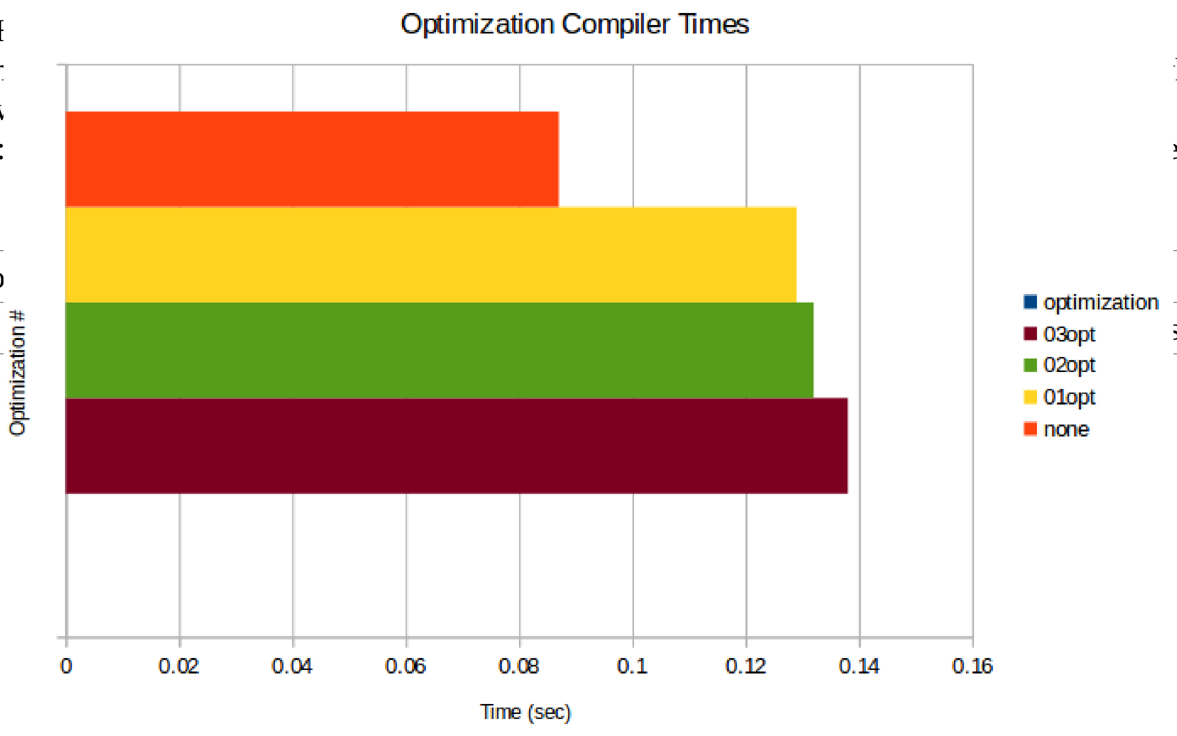
(9) What vector size are you using for all experiments in this lab?

I am using a vector size of 460000000 which utilizes approximately 89.6% of total memory.

(10) How much time does the compiler take to finish with (a) no optimization, (b) with -O1 optimization, (c) with -O2 optimization, and (d) with -O3 optimization? Report the Real time, which is the "wall clock" time. Create both a table and a graph in LibreOffice Calc.

No optimization	01 optimization	02 optimization	03 optimization
Real time = 0.087s	Real time = 0.129s	Real time = 0.132s	Real time = 0.138s

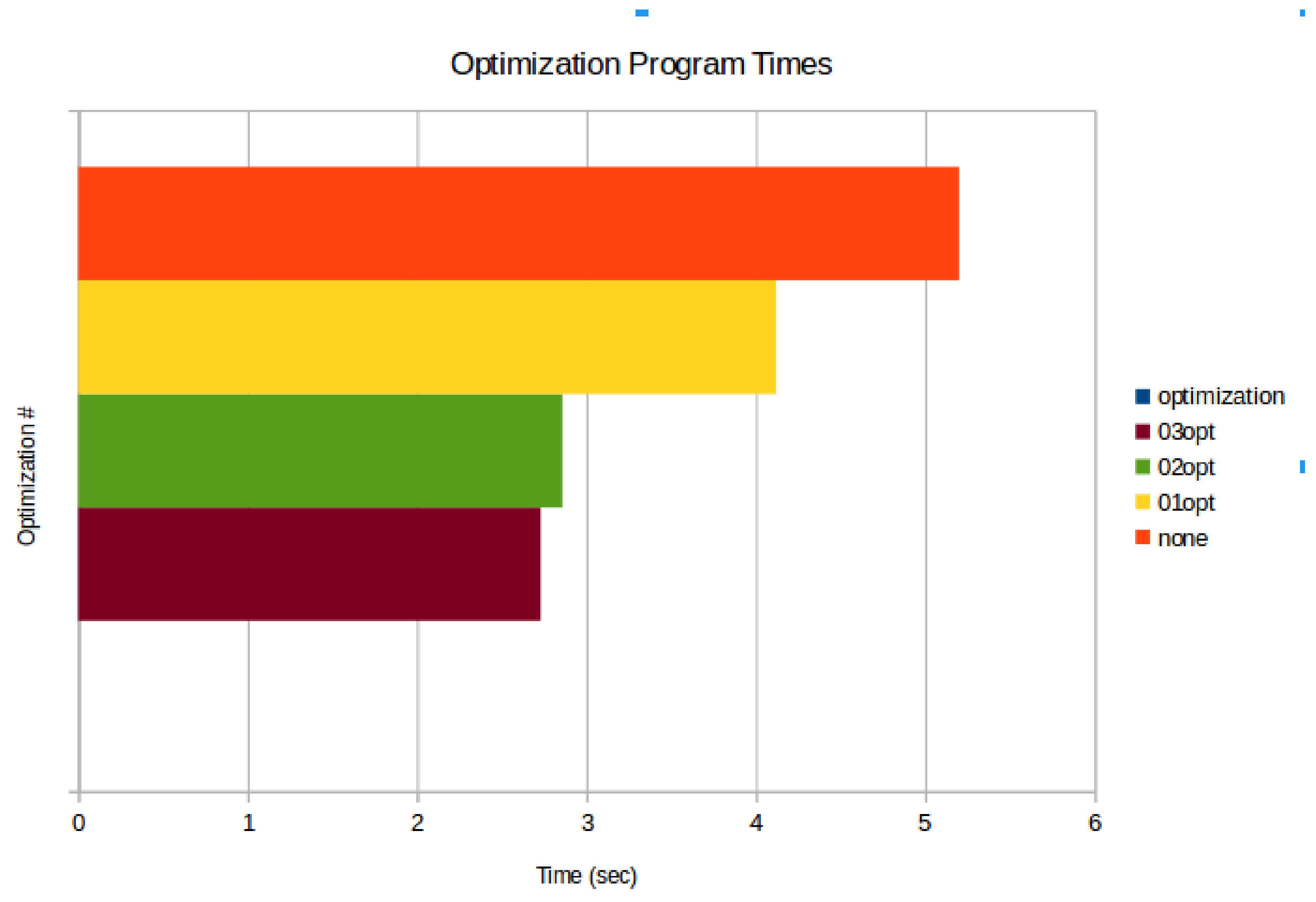
(11) I
optim
the "v
Note:



hich is

e.

No o	
Real	s

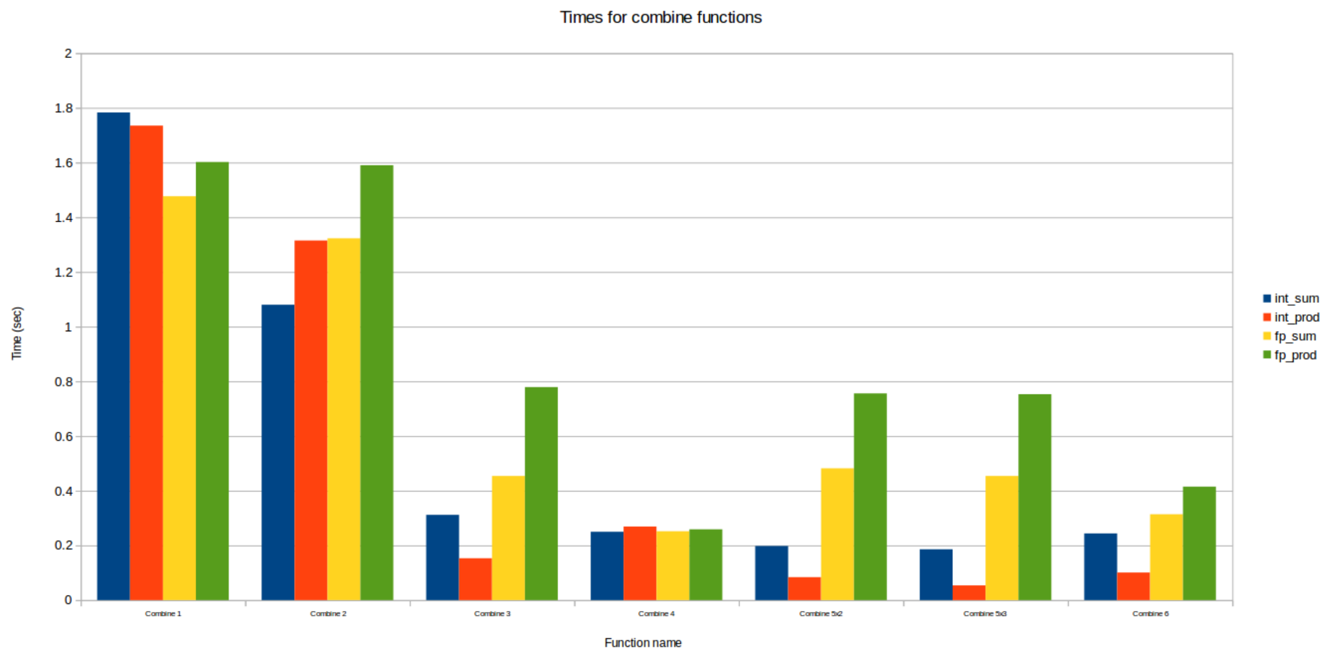


(12) After implementing each function, benchmark it for a variety of data types and mathematical operations. Fill in the table below as you write each function.

Configuration	Vector Size (Elements)	Vector Size (MB)	Time for Integer Add (seconds)	Time for Integer Multiply (sec)	Time for FP (float) ADD (sec)	Time for FP (float) Multiply (sec)
combine1()	460000000	1754.76 MB	1.783	1.735	1.477	1.602
combine2()	460000000	1754.76 MB	1.080	1.315	1.323	1.590
combine3()	460000000	1754.76 MB	0.312	0.153	0.454	0.779
combine4()	460000000	1754.76 MB	0.250	0.269	0.252	0.259

combine5x2()	460000000	1754.76 MB	0.198	0.084	0.482	0.756
combine5x3()	460000000	1754.76 MB	0.186	0.054	0.454	0.753
combine6()	460000000	1754.76 MB	0.244	0.101	0.314	0.415

(13) Using LibreOffice Calc, create a single graph that shows the data in the table created, specifically the four time columns. (You don't need to plot vector size) Note: No credit will be given for a sloppy graph that lack X and Y axis labels, a legend, and a title.



(14) As a reminder, you should be using version control to track your code, and ensure that the final code is checked in along with your report PDF.

APPENDIX

COMBINE.C CODE

```
#include "config.h"
#include "vec.h"
#include "combine.h"

#include <stdio.h>
```

```
// ORIGINAL function.
// This combiner function uses the greater amount
// of abstraction to operate, but has the slowest
// performance.
void combine1(vec_ptr v, data_t *dest)
{
  printf("Running combine1() - No code-level optimizations\n");

  long int i;

  *dest = IDENT;

  for(i=0; i < vec_length(v); i++)
    {
      data_t val;
      get_vec_element(v, i, &val);
      *dest = *dest OP val;
    }
}


// CODE MOTION OPTIMIZATION:

// Move the call to vec_length() out of the loop
// because we (the programmer) know that the vector length will
// not change in the middle of the combine() function.
//  The compiler, though, doesn't know that!
void combine2(vec_ptr v, data_t *dest)
{
  printf("Running combine2()\n");
  printf("Added optimization: Code motion\n");

  long int i;

  *dest = IDENT;

  long int k = vec_length(v);

  for(i=0; i < k; i++)
    {
      data_t val;
      get_vec_element(v, i, &val);
      *dest = *dest OP val;
    }

}


// REDUCING PROCEDURE CALLS OPTIMIZATION:

// This optimization eliminates the function call to
// get_vec_element() and accesses the data directly,
// trading off higher performance versus some loss
// of program modularity.
void combine3(vec_ptr v, data_t *dest)
{
  printf("Running combine3()\n");
  printf("Added optimization: Reducing procedure calls\n");
 data_t val;
 data_t *get_vec_start(vec_ptr v);

  long int i;

  *dest = IDENT;

  long int k = vec_length(v);

  for(i=0; i < k; i++)
    {

    *dest = *dest OP val;

    }
```

```
}


// ELIMINATING UNNEEDED MEMORY ACCESSES OPTIMIZATION:

// This optimization eliminates the trip to memory
// to store the result of each operation (and retrieve it
// the next time). Instead, it is saved in a local variable
// (i.e. a register in the processor)
// and only written to memory at the very end.
void combine4(vec_ptr v, data_t *dest)
{
  printf("Running combine4()\n");
  printf("Added optimization: Eliminating unneeded memory accesses\n");

 data_t val;
 data_t accumulate;
 data_t *get_vec_start(vec_ptr v);
 val = v->data[0];
  long int i;

  *dest = IDENT;

  long int k = vec_length(v);

  for(i=0; i < k; i++)
    {
     accumulate = v->data[i];
    *dest = accumulate OP val;

    }

}


// LOOP UNROLLING x2
// (i.e. process TWO vector elements per loop iteration)
void combine5x2(vec_ptr v, data_t *dest)
{
  printf("Running combine5x2()\n");
  printf("Added optimization: Loop unrolling x2\n");


 data_t accumulate = 0;
 data_t *get_vec_start(vec_ptr v);

  long int i;

  *dest = IDENT;

  long int k = vec_length(v);

  for(i=0; i < k; i=i+2)
    {
     accumulate = (accumulate OP v->data[i]) OP v->data[i+1];
    *dest = accumulate;
    }
}

// LOOP UNROLLING x3
// (i.e. process THREE vector elements per loop iteration)
void combine5x3(vec_ptr v, data_t *dest)
{
  printf("Running combine5x3()\n");
  printf("Added optimization: Loop unrolling x3\n");

  data_t accumulate = 0;
  data_t *get_vec_start(vec_ptr v);

  long int i;

  *dest = IDENT;
  long int k = vec_length(v);

  for(i=0; i < k; i=i+3)
    {
     accumulate = (accumulate OP v->data[i]) OP (v->data[i+1]) OP (v->data[i+2]);
    *dest = accumulate;
    }

}


// LOOP UNROLLING x2 + 2-way parallelism
void combine6(vec_ptr v, data_t *dest)
{
  printf("Running combine6()\n");
  printf("Added optimization: Loop unrolling x2, Parallelism x2\n");

  data_t accumulate_1 = 0;
  data_t accumulate_2 = 0;
  data_t *get_vec_start(vec_ptr v);

  long int i;
  *dest = IDENT;

  long int k = vec_length(v);

  for(i=0; i < k; i=i+2)
    {
     accumulate_1 = (accumulate_1 OP v->data[i]);
     accumulate_2 = (accumulate_2 OP v->data[i+1]);

    }
*dest = accumulate_1 OP  accumulate_2;
}
```