Lab Report

ECPE 170 – Computer Systems and Networks – Spring 2022

Name: Thomas Lau

Lab Topic: Performance Optimization (Lab #: 5)

Question #1:

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

Answer:

8000 MB

Question #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)

Answer:

8196MB

Question #3:

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

Answer:

7804 MB

Question #4:

Check the virtual machine configuration. How much RAM is currently allocated to Linux in your virtual machine?

Answer:

6000 MB

Question #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?

Answer:

7000 MB

Question #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.

Answer:

6771.1 MB

Question #7:

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

Answer:

The code here for combine1() sets the dest pointer to IDENT and then proceeds to enter for loop that runs for the duration of the user inputed vector size. Within this for loop, the combiner function creates a value type data_t, gets the vector element at the address of said value, and then adds or multiplies that onto the value at dest pointer. The purpose of the code is to get the product or sum of all the elements in the vector and we can simply make a change in the config.h file to PROD or SUM to make all combine functions to do that operation.

Question #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.

Answer:

Around 11000000 element

Question #9:

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

Answer:

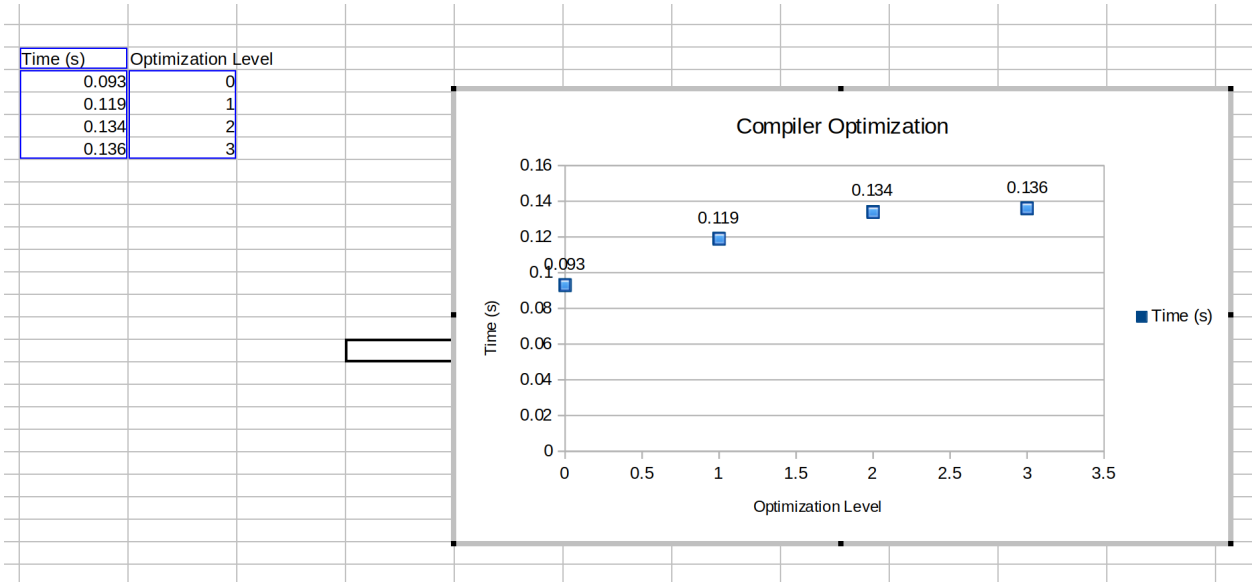
11000000

Question #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.

Answer:

Time (s)	**Optimization Level**
0.093	0
0.119	1
0.134	2
0.136	3



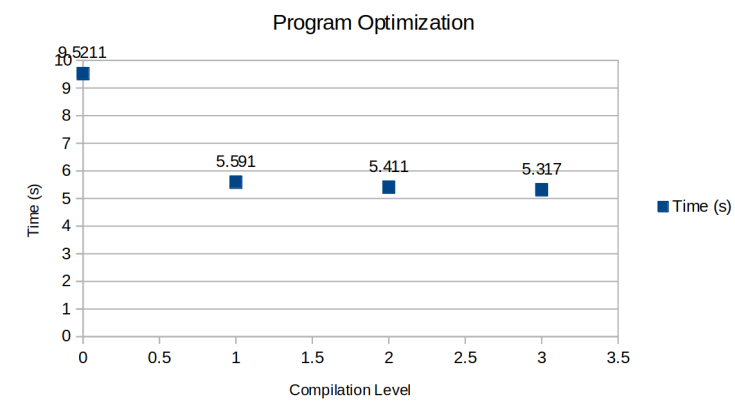
Question #11:

How much time does the program 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.

Answer:

Time (s)	Optimization Level
9.5211	0
5.591	1
5.411	2
5.317	3

Time (s)	Compilation Level
9.5211	0
5.591	1
5.411	2
5.317	3



Question #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.

Answer:

Configuration	Vector Size (elements)	Vector Size (MB)	Time for Integer Add (s)	Time for Integer Multiply (s)	Time for FP (float) Add	Time for FP (float) Multiply
combine1()	1100000000	4196.17	3.638	3.458	3.177	2.821
combine2()	1100000000	4196.17	1.845	1.890	2.615	2.601
combine3()	1100000000	4196.17	1.647	2.208	2.515	2.513
combine4()	1100000000	4196.17	0.738	0.956	1.189	1.193
combine5x2()	1100000000	4196.17	0.507	0.951	1.165	1.178
combine5	11000000	4196.17	0.494	0.939	1.161	1.171

x3()	00					
combine6()	110000000 00	4196.17	0.467	0.611	0.730	0.764

Question #13:

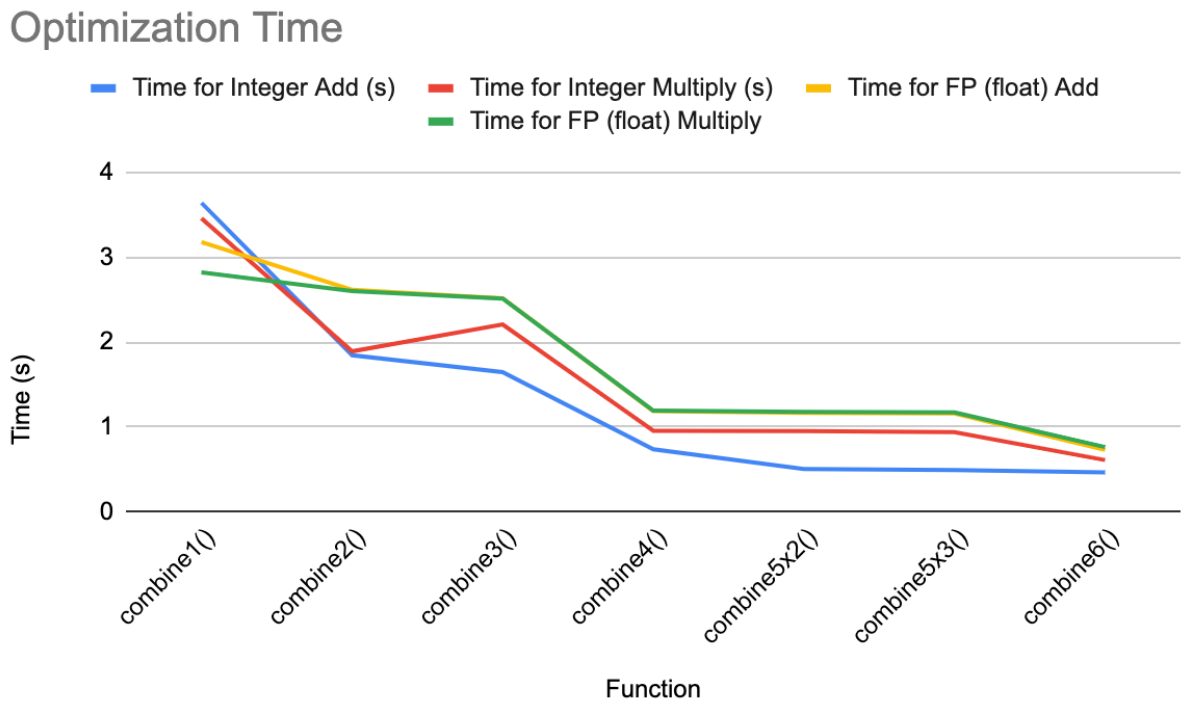
Using LibreOffice Calc, make two graphs:

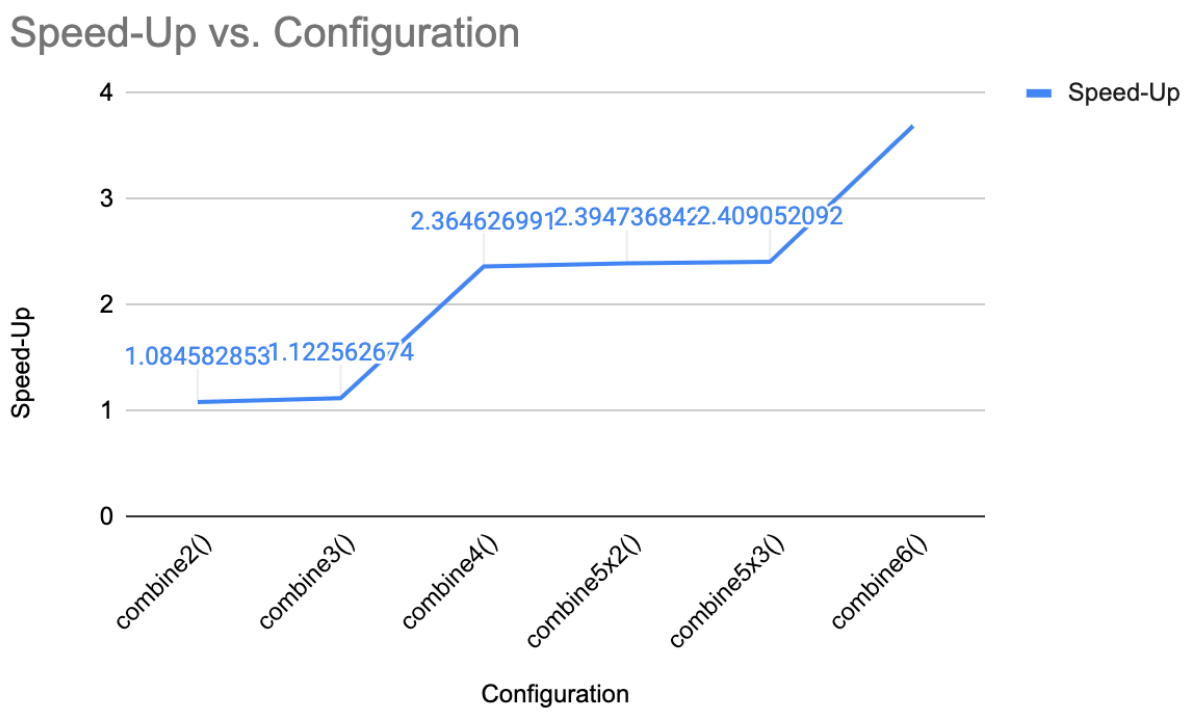
Graph 1: Create a single graph that shows the data in the table created, specifically the four time columns. You don't need to plot vector size.

Graph 2: For FP (float) multiply only, plot a line graph that shows the speed-up of combine2(), combine3(), combine4(), combine5x2(), combine5x3(), and combine6() over combine1() for the vector size tested in Question 12. Plot speed-up on the y axis and function names on the x-axis. Note that the speed-up of program A over program B is defined as (TB/TA) where TB is the execution time for program B and TA is the execution time for program A.

Note: No credit will be given for sloppy graphs that lack X and Y axis labels, a legend (for graph 1), and a title.

Answer:


Optimization Time
Time for Integer Add (s)
Time for Integer Multiply (s)
Time for FP (float) Add
Time for FP (float) Multiply
Time (s)
4
3
2
1
0
combine1()
combine2()
combine3()
combine4()
combine5x2()
combine5x3()
combine6()
Function


Speed-Up vs. Configuration
Speed-Up
Speed-Up
4
3
2
1
0
1.084582853
1.122562674
2.364626991
2.394736842
2.409052092
combine2()
combine3()
combine4()
combine5x2()
combine5x3()
combine6()
Configuration

Question #14:

As a reminder, you should track your code, and ensure that the final code is checked in along with your report PDF. You need to add your code to the report.

Answer:

Combine.c:

combine.c - lab05 - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

LAB05

- combine_program
- combine.c
- combine.h
- combine.o
- config.h
- ECPE 170 Lab 5.pdf
- main.c
- main.o
- Makefile
- README.md
- vec.c
- vec.h
- vec.o

OUTLINE

TIMELINE

combine.c | vec.h | vec.c | config.h | main.c | Makefile

combine.c > combine5x3(vec_ptr, data_t *)

```c
#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");

  // XXX - STUDENT CODE GOES HERE - XXX
  long int i;
  long int length = vec_length(v);
  *dest = IDENT;

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

main* | 0 0 | Ln 150, Col 76 | Spaces: 2 | UTF-8 | LF | C | Linux

combine.c - lab05 - Visual Studio Code

File Edit Selection View Go Run Terminal Help

combine.c > combine5x3(vec_ptr, data_t *)

```c
//  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");

  // XXX - STUDENT CODE GOES HERE - XXX
  long int i;
  long int length = vec_length(v);
  *dest = IDENT;

  for(i=0; i < length; 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");

  // XXX - STUDENT CODE GOES HERE - XXX
  long int i;
  long int length = vec_length(v);
  *dest = IDENT;
  data_t * VData = get_vec_start(v);
  for(i=0; i < length; i++)
  {
      *dest = *dest OP VData[i];
  }
}


// ELIMINATING UNNEEDED MEMORY ACCESSES OPTIMIZATION:

// This optimization eliminates the trip to memory
// to store the result of each operation (and retrieve it
```

combine.c - lab05 - Visual Studio Code

File Edit Selection View Go Run Terminal Help

combine.c > combine5x3(vec_ptr, data_t *)

```c
// 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");

  // XXX - STUDENT CODE GOES HERE - XXX
  long int i;
  long int length = vec_length(v); //variable to hold vector length
  *dest = IDENT;
  data_t  accumulate = IDENT;
  data_t * VData = get_vec_start(v);
  for(i=0; i < length; i++)
  {
      accumulate =  accumulate OP VData[i];
  }
  *dest = accumulate;
}


// 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");

  // XXX - STUDENT CODE GOES HERE - XXX
  long int i;
  long int length = vec_length(v); //variable to hold vector length
  *dest = IDENT;
  data_t  accumulate = IDENT;
  data_t * VData = get_vec_start(v);
  if(length % 2 == 1){
    accumulate = accumulate OP VData[length - 1];
    length -= 1;
  }
  for(i=0; i < length; i+=2)
  {
      accumulate =  (accumulate OP VData[i]) OP VData[i+1];
  }
  *dest = accumulate;
}

// LOOP UNROLLING x3
```

combine.c - lab05 - Visual Studio Code

```c
// 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");

  // XXX - STUDENT CODE GOES HERE - XXX
  long int i;
  long int length = vec_length(v); //variable to hold vector length
  *dest = IDENT;
  data_t  accumulate = IDENT;
  data_t * VData = get_vec_start(v);
  if(length % 3 == 1){
    accumulate =  accumulate OP VData[length - 1];
    length -= 1;
  }
  else if(length % 3 == 2){
    accumulate = (accumulate OP VData[length - 1]) OP VData[length - 2];
    length -= 2;
  }
  for(i=0; i < length; i+=3)
  {
    accumulate =  (accumulate OP VData[i]) OP VData[i+1] OP VData[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");

  // XXX - STUDENT CODE GOES HERE - XXX
  long int i;
  long int length = vec_length(v); //variable to hold vector length
  *dest = IDENT;
  data_t  accumulate0 = IDENT;
  data_t accumulate1 = IDENT;
  data_t * VData = get_vec_start(v);
  for( i = 0; i < length; i+=2){
    accumulate0 =  accumulate0 OP VData[i];
    accumulate1 = accumulate1 OP VData[i+1];
  }
  *dest = accumulate0 OP accumulate1;
}
```

Makefile:

Makefile - lab05 - Visual Studio Code

```makefile
# The variable CFLAGS specifies compiler options
#   -c :    Only compile (don't link)
#   -Wall:  Enable all warnings about lazy / dangerous C programming
#   -std=c99: Using newer C99 version of C programming language
CFLAGS=-c -Wall -std=c99 -O1

# All of the .h header files to use as dependencies
HEADERS=config.h combine.h vec.h

# All of the object files to produce as intermediary work
OBJECTS=combine.o main.o vec.o

# The final program to build
EXECUTABLE=combine_program

# ----------------------------------------

all: $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
	$(CC) $(OBJECTS) -o $(EXECUTABLE)

%.o: %.c $(HEADERS)
	$(CC) $(CFLAGS) -o $@ $<

clean:
	rm -rf *.o $(EXECUTABLE)
```

Vec.c:

```c
//Return address of beginning vector
data_t *get_vec_start(vec_ptr v){
  return v->data;
}
```