ICS 632
Assignment #1
Hualiang Li
hualiang@hawaii.edu

Note:
All tests were run on 20 cores and 1 node. All C code is compiled with -Ofast flag.
To run under basic naïve algorithm, run command:

    make basic
    make run

To run under tiled version, run command:

    make tiled
    make run

To collect tiled version data for each tiled size, run command:

    python compile_N_run.py   (generate raw.dat)
    python beautify.py   (generate beauty.dat)
    python final_step.py   (generate perf.dat, l1_miss.dat, llc_miss.dat)

All generated data are stored in a folder called "data".
To run compile_N_run.py in compute node, you have to load icc compiler first.

Question 1:

Below is my exercise.c file. I used C preprocessor and the -D compiler flag to define values for N and b at compile-time, as well as to deter- mine at compile time whether the code runs the basic algorithm or the tiled algorithm. I also made a makefile for a easier compiling time. See Appendix A for my makefile.

```c
#include <stdio.h>
//#include <sys/time.h>

#ifndef N
    #define N 17600
#endif

#ifndef b
    #define b 50
#endif

double A[N][N];
double B[N][N];

int main() {
    //struct timeval start, end;
    //gettimeofday(&start, NULL);
    int i, j;
#ifdef BASIC
    for (i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            A[i][j] += B[j][i];
        }
    }
    //printf("Runing basic algorithm\n");
#endif
#ifdef TILED
    int k, l;
    for(i=0; i<N; i+=b){
        for(j=0; j<N; j+=b){
            for(k=i; k<i+b && k<N; k++){
                for(l=j; l<j+b && l<N; l++){
                    A[k][l] += B[l][k];
                }
            }
        }
    }
    //printf("Runing tiled algorithm\n");
#endif
    //gettimeofday(&end, NULL);
    //printf("Seconds elapsed: %f\n", (end.tv_sec*1000000.0 + end.tv_usec - start.tv_sec*1000000.0 -
        start.tv_usec) / 1000000.0);
}
```

Figure 1: C source file for this assignment

I tried multiple times with basic algorithm, and found if I define N to be 17600, it runs about 2 seconds under 20-core 1-node environment.

Question 2:

Method:

I used python to collect the performance data. See Appendix B, C, D for my python code of automation data collecting. First, I used Compile_N_Run.py to compile and run the C code for 10 trials for each tile size between 1 and 300. This python code will redirect the performance data to a file called raw.dat. Then I used beautify.py to make the data more readable. These data is stored in beauty.dat. At last step, I used final_step.py to calculate the average elapsed time for each tile size.

Result:

Here is the plot of average elapsed time vs. tile size:

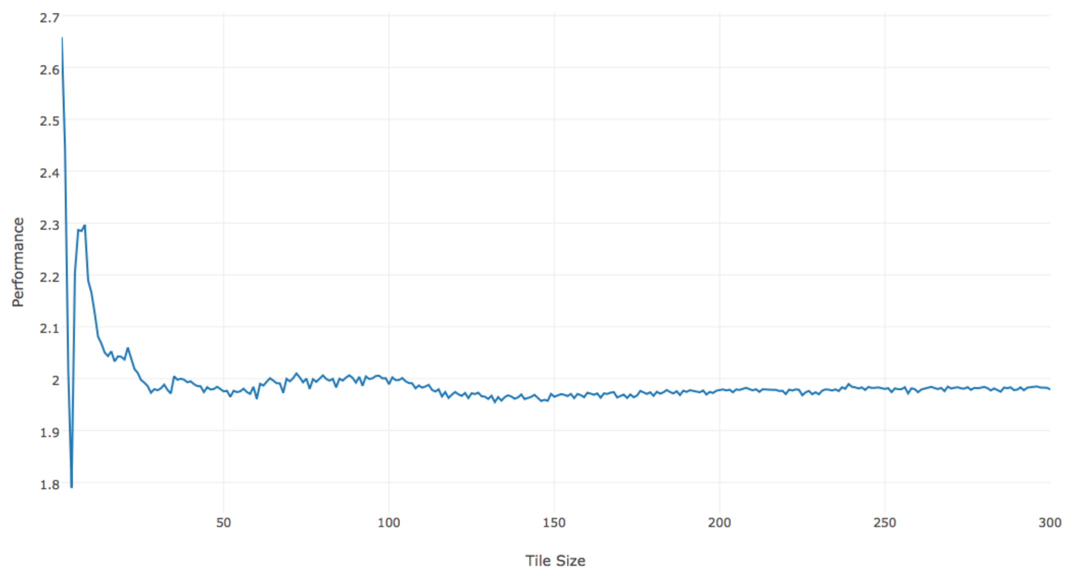

Figure 2: Elapsed Time VS. Tile Size

As clearly stated in the graph and data, for this program, tile size 4 achieves best elapsed time. With tile size 1, the elapsed time is the max. This is because with tile size 1, it is basically same with the basic algorithm. Moreover, the program has to compute extra variables, which increases the execution time. As tile size increasing, the elapsed time decreases generally. Before the tile size decreasing to 35, the elapsed time decreases greatly. With tile size between 35 and 100, it keeps around same elapsed time. With between 100 and 120, it decreases lightly. After the tile size bigger than 120, the elapsed time remains in a stable range.

With tile size of 4, the elapsed time reaches its minimum of 1.79 seconds. The speedup with regard to the basic version is 2/1.79 = 1.12.

Question 3:

Below is the graph of average L1 cache misses and LLC cache misses vs. tile size.

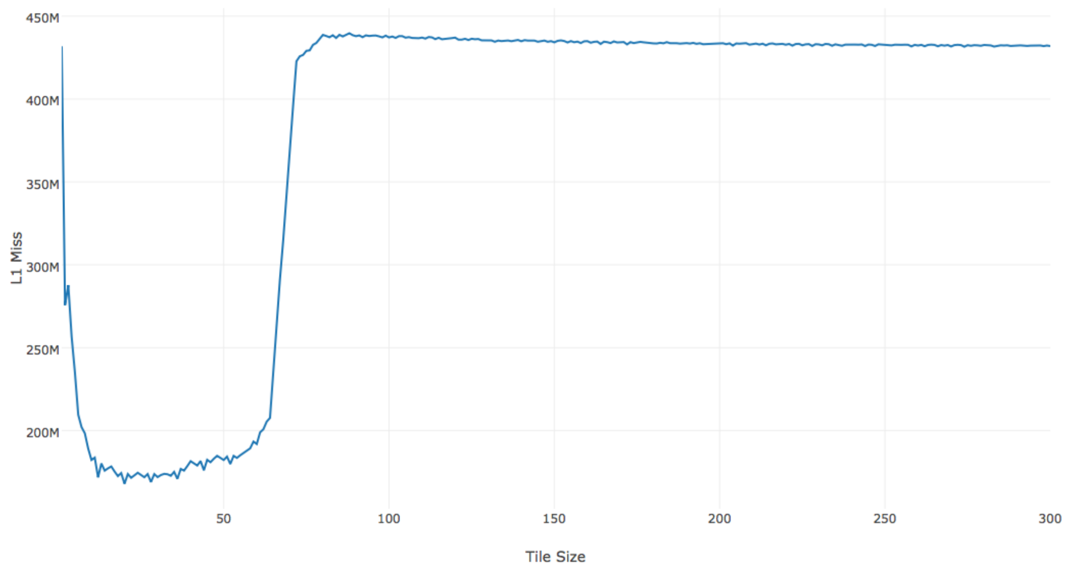

Figure 3: L1 Miss VS. Tile Size

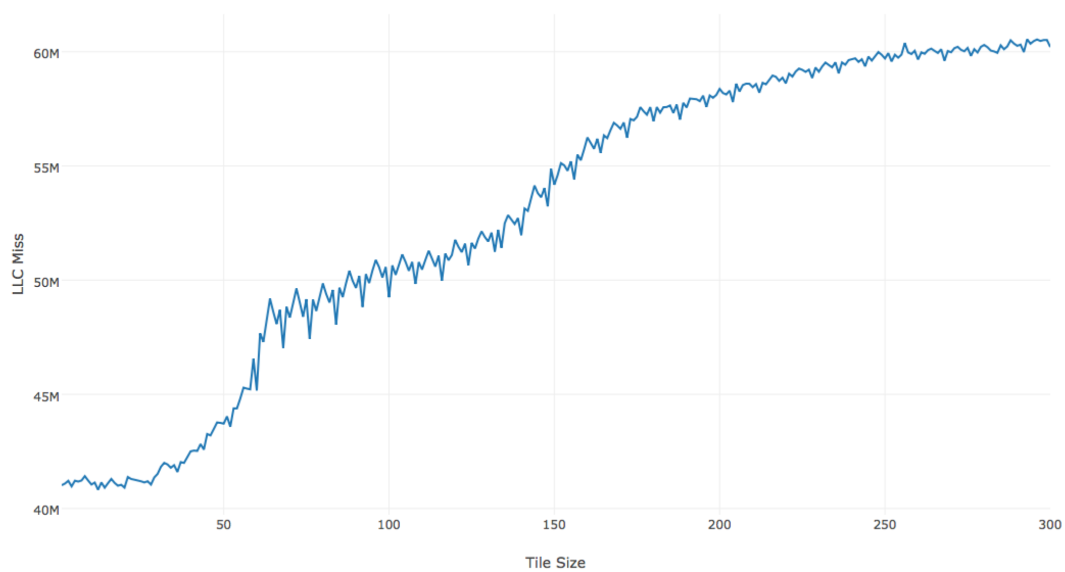

Figure 4: LLC Miss VS. Tile Size

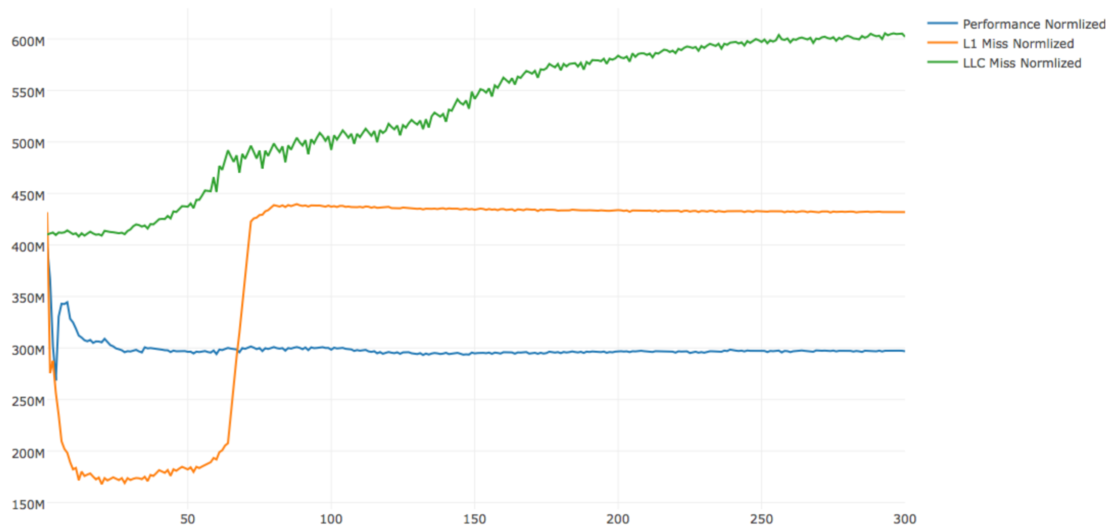

Figure 5: Elapsed Time, L1 Miss, LLC Miss VS. Tile size

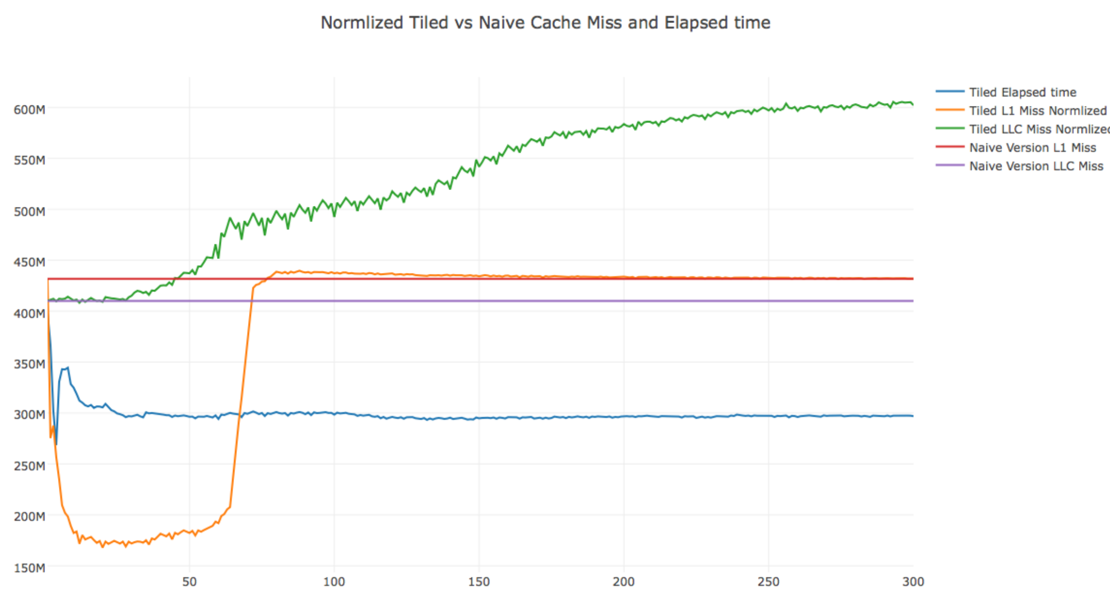

Figure 6: Tiled version VS. Naïve version

For the basic version the L1 cache miss is 431 million, LLC cache miss is 41 million, while with tiled tile size of 4 version, the L1 caches miss is 257 million, LLC cache miss is 41million. We can see that L1 dcache load misses is much smaller than the naïve version. While the LLC cache misses for both version starts almost same, but the tiled version's LLC load misses increases as tile size grows.

From the graph, we can see that, generally, the elapsed time decreases as L1 load misses decreases, which makes sense. However, there is some situation where this is not true. For example, our best tile size of 4, the L1 load miss is not in its lowest, which means the lowest L1 cache miss does not guarantee the lowest elapsed time. Based on this result, it is usually difficult to pick a decent tile size for best performance.

Appendix A

makfile:

```
CC=icc
CFLAGS=-mcmodel=medium -Ofast -c
PFLAGS=-D N=17600 -D b=4

all: exercise1

basic: CFLAGS += -D BASIC
basic: exercise1

tiled: CFLAGS += -D TILED
tiled: exercise1

exercise1: exercise1.o
	$(CC) exercise1.o -o exercise1

exercise1.o: exercise1.c
	$(CC) $(CFLAGS) $(PFLAGS) exercise1.c

run:
	./exercise1

clean:
	rm *.o
real_clean:
	rm exercise1
```

Appendix B

compile_N_run.py:

```
from subprocess import call

def run_c_program():
    #call('rm output', shell=True)
    for i in range(1,301):
        with open("./data/raw.dat", "a") as myfile:
            myfile.write("tile size=")
            myfile.write(str(i))
            myfile.write("\n")
        for j in range(10):
            compile_command = 'icc exercise1.c -o exercise1 -mcmodel=medium -Ofast -D T
            compile_command += str(i)
            call(compile_command, shell=True)
            run_command =""
            run_command +="perf stat"
            run_command +=" -e L1-dcache-load-misses"
            run_command +=" -e LLC-load-misses"
            run_command +=" ./exercise1"
            run_command +=" 2>&1"
            run_command +=" | grep -E 'time|L1-dcache|LLC'"
            run_command +=" >> ./data/raw.dat"
            call(run_command, shell=True)
            #run = 'sbatch exercise1.slurm'

run_c_program()
```

Appendix C

beautify.py:

```
import re
'''
This code beautify the raw data generate from  'perf stat | grep > file'
Such that it is easy to read and compute
'''
def warehouse():
    out = open('./data/beauty.dat', 'w')
    with open('./data/raw.dat', 'r') as inputdata:
        line = ''
        item = ""
        count = 0
        for line in inputdata:
            if line ==''
                break
            if line.startswith('tile'):
                out.write(line)
                continue
            count += 1
            line = re.sub('[,@]', '', line)
            line = line.strip()
            line = line.split()
            for word in line:
                if is_number(word):
                    item += str(word)
                    if not count %3 == 0:
                        item += '\t'
            if count % 3 == 0:
                out.write(item)
                out.write("\n")
                item = ""

    out.close()

def is_number(s):
    try:
        float(s)
        return True
    except ValueError:
        try:
            int(s)
            return True
        except ValueError:
            return False


warehouse()
```

Appendix D

final_step.py:

```
def cal_ave():
    performance = open('./data/perf.dat', 'w')
    l1_miss = open('./data/l1_miss.dat', 'w')
    llc_miss = open('./data/llc_miss.dat', 'w')
    with open('./data/beauty.dat', 'r') as infile:
        line = ""
        count = 0
        perf = 0
        l1 = 0
        llc = 0
        for line in infile:
            if line == '':
                break
            if line.startswith('tile'):
                continue
            count += 1
            line = line.strip().split()
            l1 += int(line[0])
            llc += int(line[1])
            perf += float(line[2])
            if count % 10 == 0:
                performance.write(str(perf/10))
                l1_miss.write(str(l1/10))
                llc_miss.write(str(llc/10))
                performance.write('\n')
                l1_miss.write('\n')
                llc_miss.write('\n')
                perf = 0
                llc = 0
                l1 = 0
    performance.close()
    l1_miss.close()
    llc_miss.close()

cal_ave()
```