EECE 7352: Computer Architecture

Assignment 1

Jiayun Xin

NUID: 001563582

College of Engineering

Northeastern University Boston, Massachusetts

Spring, 2022

Q1.

A.

1) Changing the size of environment variable can cause the program changing because of the change of start address of the stack.
2) One system measured can be better than another system though it is not because of measurement bias.
3) Changing the link order of a program can affect performance in many ways.
4) Code compiler “gcc” can cause measurement bias.
5) Room temperature can cause measurement bias because temperature affects CPU clock speed.

B.

The two major findings I choose are the effect of linking order and “gcc” version. The benchmark I used is linpack downloaded from “https://github.com/nfinit/ansibench”/. The benchmark was executed on Discovery machine. The specific cpu information of Discovery is list below.

```
[xin.ji@c0208 linpack]$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                48
On-line CPU(s) list:   0-47
Thread(s) per core:    2
Core(s) per socket:    12
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 63
Model name:            Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz
Stepping:              2
CPU MHz:               2999.902
CPU max MHz:           3500.0000
CPU min MHz:           1200.0000
BogoMIPS:              5199.96
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              30720K
NUMA node0 CPU(s):     0-11,24-35
NUMA node1 CPU(s):     12-23,36-47
```

Effect of linking order on measurement bias:

```
[xin.ji@c0208 linpack]$ make
gcc -Wall -ansi -O2  -lm -DSP ./src/linpack.c -o ./bin/linpacksp
gcc -Wall -ansi -O2  -lm -DDP ./src/linpack.c -o ./bin/linpackdp
[xin.ji@c0208 linpack]$ ./bin/linpackdp
Enter array size (q to quit) [100]:  100
LINPACK benchmark, Double precision.
Machine precision:  15 digits.
Array size 100 X 100.
Memory required:  79K.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
    4096    0.64  75.00%  12.50%  12.50%  1292190.476
    8192    1.31  72.52%   8.40%  19.08%  1365333.333
   16384    2.60  72.31%   5.00%  22.69%  1440053.068
   32768    5.25  75.81%   6.48%  17.71%  1340049.383
   65536   10.48  74.62%   5.25%  20.13%  1383276.782
```

```
[xin.ji@c0208 linpack]$ make
gcc -Wall -ansi -O3  -lm -DSP ./src/linpack.c -o ./bin/linpacksp
gcc -Wall -ansi -O3  -lm -DDP ./src/linpack.c -o ./bin/linpackdp
[xin.ji@c0208 linpack]$ ./bin/linpacksp
Enter array size (q to quit) [100]:  100
LINPACK benchmark, Single precision.
Machine precision:  6 digits.
Array size 100 X 100.
Memory required:  40K.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
    8192    0.79  49.37%  10.13%  40.51%  3079262.250
   16384    1.60  71.25%   5.00%  23.75%  2372546.750
   32768    3.17  70.03%   7.89%  22.08%  2343730.500
   65536    6.35  64.88%   7.09%  28.03%  2533482.750
  131072   12.72  65.72%   6.92%  27.36%  2506064.500
```

```
[xin.ji@c0208 linpack]$ make
gcc -ansi -O2 -Wall  -lm -DSP ./src/linpack.c -o ./bin/linpacksp
gcc -ansi -O2 -Wall  -lm -DDP ./src/linpack.c -o ./bin/linpackdp
[xin.ji@c0208 linpack]$ ./bin/linpacksp
Enter array size (q to quit) [100]:  100
LINPACK benchmark, Single precision.
Machine precision:  6 digits.
Array size 100 X 100.
Memory required:  40K.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
    4096    0.60  71.67%  11.67%  16.67%  1447253.375
    8192    1.19  63.87%   4.20%  31.93%  1786731.625
   16384    2.42  71.90%   6.61%  21.49%  1523424.500
   32768    4.88  66.39%  11.07%  22.54%  1531487.125
   65536    9.69  67.29%   7.95%  24.77%  1588205.875
  131072   19.42  70.19%   6.64%  23.17%  1552012.750
```

```
[xin.ji@c0208 linpack]$ make
gcc -ansi -O3 -Wall  -lm -DSP ./src/linpack.c -o ./bin/linpacksp
gcc -ansi -O3 -Wall  -lm -DDP ./src/linpack.c -o ./bin/linpackdp
[xin.ji@c0208 linpack]$ ./bin/linpacksp
Enter array size (q to quit) [100]:  100
LINPACK benchmark, Single precision.
Machine precision:  6 digits.
Array size 100 X 100.
Memory required:  40K.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
    8192    0.79  55.70%   7.59%  36.71%  2894506.500
   16384    1.59  67.30%   5.03%  27.67%  2516961.500
   32768    3.19  64.26%   4.70%  31.03%  2631370.000
   65536    6.40  65.31%   7.19%  27.50%  2495263.750
  131072   12.77  63.90%   5.87%  30.23%  2598881.750
```

```
[xin.ji@c0208 linpack]$ make
gcc -O2 -Wall -ansi  -lm -DSP ./src/linpack.c -o ./bin/linpacksp
gcc -O2 -Wall -ansi  -lm -DDP ./src/linpack.c -o ./bin/linpackdp
[xin.ji@c0208 linpack]$ ./bin/linpacksp
Enter array size (q to quit) [100]:  100
LINPACK benchmark, Single precision.
Machine precision:  6 digits.
Array size 100 X 100.
Memory required:  40K.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
    4096   0.60  63.33%   8.33%  28.33%  1682852.250
    8192   1.24  64.52%   4.84%  30.65%  1682853.000
   16384   2.42  71.90%   6.61%  21.49%  1523424.500
   32768   4.82  68.88%   7.47%  23.65%  1573101.750
   65536   9.71  70.96%   7.11%  21.94%  1527444.250
  131072  19.51  70.73%   6.56%  22.71%  1535548.125
[xin.ji@c0208 linpack]$ make
gcc -O3 -Wall -ansi  -lm -DSP ./src/linpack.c -o ./bin/linpacksp
gcc -O3 -Wall -ansi  -lm -DDP ./src/linpack.c -o ./bin/linpackdp
[xin.ji@c0208 linpack]$ ./bin/linpacksp
Enter array size (q to quit) [100]:  100
LINPACK benchmark, Single precision.
Machine precision:  6 digits.
Array size 100 X 100.
Memory required:  40K.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
    8192   0.79  64.56%  10.13%  25.32%  2452971.000
   16384   1.59  57.23%   7.55%  35.22%  2810200.250
   32768   3.18  66.35%   4.40%  29.25%  2572893.000
   65536   6.36  68.71%   4.56%  26.73%  2484552.000
  131072  12.73  62.29%   5.58%  32.13%  2680093.750
```

Linking order	-Wall -ansi -O2/O3	-ansi -O2/O3 -Wall	-O2/O3 -Wall -ansi
O2 – time(s)	10.48	19.42	19.51
O3 – time(s)	12.72	12.77	12.73

As we can see from the table, linking order does cause measurement bias.

Effect of “gcc” version on measurement bias:
There are some “gcc” versions that I can choose, such as 10.1.0 11.1.0 4.9.1 5.5.0 6. 4.0 7.2.0 7.3.0-skylake 8.1.0 8.1.0-AMD 9.2.0-skylake 9.2.0-zen2. I choose three of them to test the effect of “gcc” version on measurement bias.

“gcc” version:7.3.0

```
[xin.ji@c0227 linpack]$ make
gcc -Wall -ansi -O2  -lm -DSP ./src/linpack.c -o ./bin/linpacksp
gcc -Wall -ansi -O2  -lm -DDP ./src/linpack.c -o ./bin/linpackdp
[xin.ji@c0227 linpack]$ ./bin/linpacksp
Enter array size (q to quit) [100]:  100
LINPACK benchmark, Single precision.
Machine precision:  6 digits.
Array size 100 X 100.
Memory required:  40K.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
    4096   0.64  70.31%  10.94%  18.75%  1391589.875
    8192   1.29  66.67%  10.85%  22.48%  1447253.875
   16384   2.55  74.51%   4.71%  20.78%  1432924.625
   32768   5.20  74.81%   5.19%  20.00%  1391590.000
   65536  10.32  73.55%   6.49%  19.96%  1401697.375
[xin.ji@c0227 linpack]$ make
gcc -Wall -ansi -O3  -lm -DSP ./src/linpack.c -o ./bin/linpacksp
gcc -Wall -ansi -O3  -lm -DDP ./src/linpack.c -o ./bin/linpackdp
[xin.ji@c0227 linpack]$ ./bin/linpacksp
Enter array size (q to quit) [100]:  100
LINPACK benchmark, Single precision.
Machine precision:  6 digits.
Array size 100 X 100.
Memory required:  40K.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
    8192   0.87  72.41%   4.60%  22.99%  2160079.500
   16384   1.70  67.06%   6.47%  26.47%  2315605.500
   32768   3.42  63.45%   8.19%  28.36%  2362864.250
   65536   6.87  67.54%   6.84%  25.62%  2265761.750
  131072  13.74  68.92%   6.62%  24.45%  2230835.500
```

“gcc” version: 9.2.0

```
[xin.ji@c0208 linpack]$ make
gcc -Wall -ansi -O2  -lm -DSP ./src/linpack.c -o ./bin/linpacksp
gcc -Wall -ansi -O2  -lm -DDP ./src/linpack.c -o ./bin/linpackdp
[xin.ji@c0208 linpack]$ ./bin/linpackdp
Enter array size (q to quit) [100]:  100
LINPACK benchmark, Double precision.
Machine precision:  15 digits.
Array size 100 X 100.
Memory required:  79K.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
    4096   0.64  75.00%  12.50%  12.50%  1292190.476
    8192   1.31  72.52%   8.40%  19.08%  1365333.333
   16384   2.60  72.31%   5.00%  22.69%  1440053.068
   32768   5.25  75.81%   6.48%  17.71%  1340049.383
   65536  10.48  74.62%   5.25%  20.13%  1383276.782
```

```
[xin.ji@c0208 linpack]$ make
gcc -Wall -ansi -O3  -lm -DSP ./src/linpack.c -o ./bin/linpacksp
gcc -Wall -ansi -O3  -lm -DDP ./src/linpack.c -o ./bin/linpackdp
[xin.ji@c0208 linpack]$ ./bin/linpacksp
Enter array size (q to quit) [100]:  100
LINPACK benchmark, Single precision.
Machine precision:  6 digits.
Array size 100 X 100.
Memory required:  40K.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
    8192    0.79  49.37%  10.13%  40.51%  3079262.250
   16384    1.60  71.25%   5.00%  23.75%  2372546.750
   32768    3.17  70.03%   7.89%  22.08%  2343730.500
   65536    6.35  64.88%   7.09%  28.03%  2533482.750
  131072   12.72  65.72%   6.92%  27.36%  2506064.500
```

"gcc" version: 10.1.0

```
[xin.ji@c0227 linpack]$ make
gcc -Wall -ansi -O2  -lm -DSP ./src/linpack.c -o ./bin/linpacksp
gcc -Wall -ansi -O2  -lm -DDP ./src/linpack.c -o ./bin/linpackdp
[xin.ji@c0227 linpack]$ ./bin/linpacksp
Enter array size (q to quit) [100]:  100
LINPACK benchmark, Single precision.
Machine precision:  6 digits.
Array size 100 X 100.
Memory required:  40K.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
    4096    0.65  75.38%   3.08%  21.54%  1418875.875
    8192    1.28  72.66%   5.47%  21.88%  1447253.500
   16384    2.66  72.18%   5.26%  22.56%  1405100.625
   32768    5.17  75.44%   7.74%  16.83%  1346282.625
   65536   10.46  71.70%   6.41%  21.89%  1417139.000
[xin.ji@c0227 linpack]$ make
gcc -Wall -ansi -O3  -lm -DSP ./src/linpack.c -o ./bin/linpacksp
gcc -Wall -ansi -O3  -lm -DDP ./src/linpack.c -o ./bin/linpackdp
^[[A[xin.ji@c0227 linpack]$ ./bin/linpacksp
Enter array size (q to quit) [100]:  100
LINPACK benchmark, Single precision.
Machine precision:  6 digits.
Array size 100 X 100.
Memory required:  40K.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
    8192    0.74  55.41%  12.16%  32.43%  2894506.500
   16384    1.50  60.67%   7.33%  32.00%  2837752.000
   32768    3.02  63.91%   3.31%  32.78%  2851732.500
   65536    6.01  55.91%   7.15%  36.94%  3054890.750
  131072   11.98  62.85%   5.68%  31.47%  2820469.750
```

"gcc" version	7.3.0	9.2.0	10.1.0
O2 – time(s)	10.32	10.48	10.46
O3 – time(s)	13.74	12.72	11.98

As we can see from the table, "gcc" version does cause measurement bias.

C.

Benchmark: Linpack

Linking order	-Wall -ansi -O2/O3	-ansi -O2/O3 -Wall	-O2/O3 -Wall -ansi
O2 – time(s)	10.48	19.42	19.51
O3 – time(s)	12.72	12.77	12.73

Benchmark: Whetstone

Linking order	-Wall -ansi -O2/O3	-ansi -O2/O3 -Wall	-O2/O3 -Wall -ansi
O2 – time(s)	37	37	37
O3 – time(s)	37	36	36

Benchmark: Nbench

Linking order	-Wall -ansi -O2/O3	-ansi -O2/O3 -Wall	-O2/O3 -Wall -ansi
O2 – numeric sort(iterations/sec)	532.8	531.84	535.32
O3 – numeric sort(iterations/sec)	532.68	533.76	533.76

All three benchmarks are sourced from “https://github.com/nfinit/ansibench”/.
The solution I used is executing a diverse set of benchmarks and change their linking order to see the running time. The reason is that there is a possibility the benchmark itself is biased and I test a diverse set of workloads. After testing three different benchmarks, the results of whetstone and nbench do not have relatively large difference by changing linking order. If I put the O2/O3 results of whetstone and nbench on a histogram, it is a tight distribution. Using a diverse set of benchmarks decreases measurement bias successfully.

Q2.

Why You Should Care About Quantile Regression. Why quantile regression is more appropriate than ANOVA? Can you provide experimental evidence to support this?

Quantile regression is an effective statistical method to evaluate computer performance, though it is difficult to analyze computer performance. The paper "Why You Should Care About Quantile Regression" applied ANOVA and quantile regression respectively to analyze a reproducible experiment.

Although ANOVA can be a way to analyze a repetitive experiment, its analysis cannot include all the data because of the requirements imposed on the data. Compared with ANOVA, quantile regression can be a better choice because its conclusion would conclude more details than ANOVA and the data do not need to meet some specific statistical requirements.

I collected and designed an experiment data about a class of student heights to statistic the quantile and ANOVA value when they are 18 and the quantile percentage are 3%, 10%, 25%, 50%, 75%, 90%, 97%. The result is shown below. Compared with the calculated variance, quantile table shows more information about student height. The distribution of student height can be seen from the quantile table while ANOVA only display the degree of difference from the average height. Quantile regression would have similar effect analysis with quantile table. Therefore, quantile regression is more appropriate than ANOVA.

Quantile(%)	3	10	25	50	75	90	97
Height(cm)	155	158	161	170	178	184	190

ANOVA: 63

Q3.
Why is IPC considered harmful for multiprocessor workloads? What are the potential solution? What do the authors mean by "work-related" metrics?

For the reason why IPC is harmful for multiprocessor workloads, it is that IPC can result in inaccurate performance and incorrect conclusions because the instruction path of multithreaded workloads can vary slightly due to the effects, such as mutex lock and architectural enhances.

The potential solution includes using user-code IPC, ignoring idle time, ignoring spin locks and trace.

Work related metrics are kinds of method using metrics proportional to time per program to predict performance, which provides a safer and accurate choice than conventional IPC.

Q4
Original average CPI = (40% * 2 + 20% * 6 + 40% * 8cycles) = 5.2

Optimization 1 (reduces the number of cycles taken to execute an ADD instruction to a single cycle):
Average CPI = 40%ADD * 1cycles + 20%MULT * 6cycles + 40%OTHER * 8cycles)
= 4.8
Speedup1 = T_{OLD} / T_{NEW} = Original CPI / new CPI = 5.2 / 4.8 = 1.08

Optimization 2 (reduces the number of cycles taken to execute an MULT instruction to 4 cycles):
Average CPI = 40%ADD * 2cycles + 20%MULT * 4cycles + 40%OTHER * 8cycles = 4.8

Speedup2 = T_{OLD} / T_{NEW} = Original CPI / new CPI = 5.2 / 4.8 = 1.08

Since we have same speedups by both optimizations, so any approach will give the same performance improvement.

Q5

a.

The relative execution time without the enhancement:

40% + 60% * 8 = 520%

The speedup we have obtained from fast mode:

$Executiontime_{unenhanced}$/ $Executiontime_{enhanced}$ = 520%/100% = 5.2

b.

the percentage of the original execution converted to fast mode:

($Speedup_{overall}$ * $Speedup_{accelerated}$ − $Speedup_{accelerated}$) / ($Speedup_{overall}$ * $Speedup_{accelerated}$ − $Speedup_{overall}$)

= 5.2 * 8 – 8 / 5.2 * 8 – 5.2

= 0.9231

92.31% original execution was converted to fast mode