EECE 7205: Introduction of Computer Engineering

Assignment 1

Jiayun Xin

NUID: 001563582

College of Engineering

Northeastern University Boston, Massachusetts

Fall, 2021

1.

Code source:
Cpu 频率：cat /proc/cpuinfo | grep MHz
https://www.cnblogs.com/ggjucheng/archive/2013/01/14/2859613.html
htop – cpu

https://github.com/nfinit/ansibench
https://github.com/jamesETsmith/SpMV_benchmark

Salloc -N1 –-exclusive -p express

a.

Node: c0241
Linux-based system: Discovery systems
Architecture: x86_64
Model: Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz
Frequency of the CPU: 2.60GHz
Number of cores: 48
Threads per core: 2
Cores per socket: 12
Memory size: 131141092 kB (in total)
Operating system version: 3.10.0-1160.25.1.el7.x86_64

Source code: https://github.com/nfinit/ansibench
Benchmark: LINPACK - calculate FLOPS

```
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
----------------------------------------------------
    2048    0.60  70.00%  10.00%  20.00%  753777.875
    4096    1.18  79.66%   5.08%  15.25%  723626.688
    8192    2.38  79.41%   6.72%  13.87%  705977.375
   16384    4.75  76.42%   5.05%  18.53%  747933.875
   32768    9.52  82.98%   3.89%  13.13%  700001.812
   65536   19.00  75.16%   4.95%  19.89%  760711.875
```

```
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
----------------------------------------------------
    2048    0.59  71.18%   1.70%  27.12%  841440.688
    4096    1.19  89.08%   4.20%   6.72%  651913.312
    8192    2.38  78.99%   4.20%  16.81%  730934.812
   16384    4.75  77.05%   5.47%  17.47%  738394.188
   32768    9.52  76.79%   3.89%  19.33%  753775.500
   65536   19.01  79.43%   4.68%  15.89%  724082.938
```

```
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
----------------------------------------------------
    2048    0.61  85.25%   3.28%  11.48%  670023.562
    4096    1.18  79.66%   7.63%  12.71%  702556.250
    8192    2.38  78.15%   5.46%  16.39%  727266.562
   16384    4.76  75.84%   5.04%  19.12%  751823.188
   32768    9.50  80.74%   4.21%  15.05%  717350.562
   65536   19.02  77.92%   5.21%  16.88%  732323.875
```

```
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
----------------------------------------------------
    2048    0.60  91.67%   1.67%   6.67%  646089.250
    4096    1.18  78.81%   5.08%  16.10%  730932.000
    8192    2.39  85.36%   6.70%   7.95%  657838.812
   16384    4.75  75.79%   6.53%  17.68%  740282.375
   32768    9.50  77.37%   7.16%  15.47%  720924.000
   65536   19.02  78.92%   5.42%  15.67%  721823.125
```

```
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
----------------------------------------------------
    2048    0.59  81.36%   3.39%  15.26%  723648.750
    4096    1.19  78.99%   1.68%  19.33%  753772.500
    8192    2.38  83.19%   5.46%  11.35%  685906.812
   16384    4.75  77.26%   6.53%  16.21%  727261.000
   32768    9.50  76.95%   5.68%  17.37%  737457.688
   65536   19.04  79.04%   4.25%  16.70%  730013.562
```

All the execution time over the 5 runs of the program are the results of evaluating single-precision performance and about 19 second, but the execution time of the first program is a little faster than the other four programs. And the first program has the highest KFLOPS. The difference of execution time and KFLOPS can attribute to memory sharing, cache occupation and core occupation, which causes the difference of percentage about each process like DGEFA, DGESL and OVERHEAD.

Source code: https://github.com/nfinit/ansibench
Benchmark: STREAM - measuring memory performance

```
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:            6083.9     0.027747     0.026299     0.030557
Scale:           5983.5     0.028196     0.026740     0.031045
Add:             8308.6     0.029919     0.028886     0.033146
Triad:           8037.0     0.030937     0.029862     0.034443
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:            6079.3     0.026855     0.026319     0.028399
Scale:           5986.5     0.027251     0.026727     0.029320
Add:             8285.6     0.029678     0.028966     0.032043
Triad:           8053.1     0.030729     0.029802     0.033942
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:            5858.5     0.027401     0.027311     0.027614
Scale:           5746.5     0.027977     0.027843     0.028265
Add:             8095.8     0.029769     0.029645     0.029946
Triad:           7841.7     0.030738     0.030606     0.031002
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:            5882.4     0.027242     0.027200     0.027363
Scale:           5740.5     0.027999     0.027872     0.028176
Add:             8136.7     0.029607     0.029496     0.029966
Triad:           7937.2     0.030307     0.030237     0.030376
-------------------------------------------------------------
```

```
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           5866.2     0.027436     0.027275     0.027784
Scale:          5739.9     0.027961     0.027875     0.028146
Add:            8129.0     0.029627     0.029524     0.029955
Triad:          7853.2     0.030721     0.030561     0.031214
-------------------------------------------------------------
```

The Stream benchmark is used to measure the memory performance (bandwidth) with the respect of four kernels, which are copy, scale, add and triad. The triad function is a combination of the others, so I use this to observe the program execution time. Through the screenshot, all the execution time of average time of the triad function over the 5 runs of the program are the about 0.03, but the average time of the fourth program is a little faster than the other four programs. The difference of execution time can attribute to the occupation of cache 1, 2 and main memory and the simultaneous use of the same core.

Source code: https://github.com/nfinit/ansibench

Benchmark: Whetstone – measures a computer's floating-point performance

```
[xin.ji@c0241 whetstone]$ ./bin/whetdc
      0       0       0   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
12000000 14000000 12000000 -1.7680e-313  2.3567e-314 -1.7679e-313  2.3568e-314
14000000 12000000 12000000 -9.8808e-320  9.8808e-320  9.8808e-320  2.9643e-319
345000000       1       1   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
210000000       1       2   6.0000e+00   6.0000e+00  9.8808e-320  2.9643e-319
32000000       1       2  4.9407e-320  4.9407e-320  4.9407e-320  4.9407e-320
899000000       1       2   1.0000e+00   1.0000e+00   9.9994e-01   9.9994e-01
616000000       1       2   3.0000e+00   2.0000e+00   3.0000e+00  2.9643e-319
      0       2       3   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
93000000       2       3   1.0000e+00   1.0000e+00   1.0000e+00   1.0000e+00

Loops: 1000000, Iterations: 1, Duration: 39 sec.
C Converted Double Precision Whetstones: 2564.1 MWIPS
[xin.ji@c0241 whetstone]$ ./bin/whetdc
      0       0       0   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
12000000 14000000 12000000 -1.7680e-313  2.3567e-314 -1.7679e-313  2.3568e-314
14000000 12000000 12000000 -9.8808e-320  9.8808e-320  9.8808e-320  2.9643e-319
345000000       1       1   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
210000000       1       2   6.0000e+00   6.0000e+00  9.8808e-320  2.9643e-319
32000000       1       2  4.9407e-320  4.9407e-320  4.9407e-320  4.9407e-320
899000000       1       2   1.0000e+00   1.0000e+00   9.9994e-01   9.9994e-01
616000000       1       2   3.0000e+00   2.0000e+00   3.0000e+00  2.9643e-319
      0       2       3   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
93000000       2       3   1.0000e+00   1.0000e+00   1.0000e+00   1.0000e+00

Loops: 1000000, Iterations: 1, Duration: 38 sec.
C Converted Double Precision Whetstones: 2631.6 MWIPS
[xin.ji@c0241 whetstone]$ ./bin/whetdc
      0       0       0   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
12000000 14000000 12000000 -1.7680e-313  2.3567e-314 -1.7679e-313  2.3568e-314
14000000 12000000 12000000 -9.8808e-320  9.8808e-320  9.8808e-320  2.9643e-319
345000000       1       1   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
210000000       1       2   6.0000e+00   6.0000e+00  9.8808e-320  2.9643e-319
32000000       1       2  4.9407e-320  4.9407e-320  4.9407e-320  4.9407e-320
899000000       1       2   1.0000e+00   1.0000e+00   9.9994e-01   9.9994e-01
616000000       1       2   3.0000e+00   2.0000e+00   3.0000e+00  2.9643e-319
      0       2       3   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
93000000       2       3   1.0000e+00   1.0000e+00   1.0000e+00   1.0000e+00

Loops: 1000000, Iterations: 1, Duration: 39 sec.
C Converted Double Precision Whetstones: 2564.1 MWIPS
```

```
[xin.ji@c0241 whetstone]$ ./bin/whetdc
      0       0       0   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
12000000 14000000 12000000 -1.7680e-313  2.3567e-314 -1.7679e-313  2.3568e-314
14000000 12000000 12000000 -9.8808e-320  9.8808e-320  9.8808e-320  2.9643e-319
345000000       1       1   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
210000000       1       2   6.0000e+00   6.0000e+00  9.8808e-320  2.9643e-319
32000000       1       2  4.9407e-320  4.9407e-320  4.9407e-320  4.9407e-320
899000000       1       2   1.0000e+00   1.0000e+00   9.9994e-01   9.9994e-01
616000000       1       2   3.0000e+00   2.0000e+00   3.0000e+00  2.9643e-319
      0       2       3   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
93000000       2       3   1.0000e+00   1.0000e+00   1.0000e+00   1.0000e+00

Loops: 1000000, Iterations: 1, Duration: 38 sec.
C Converted Double Precision Whetstones: 2631.6 MWIPS
[xin.ji@c0241 whetstone]$ ./bin/whetdc
      0       0       0   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
12000000 14000000 12000000 -1.7680e-313  2.3567e-314 -1.7679e-313  2.3568e-314
14000000 12000000 12000000 -9.8808e-320  9.8808e-320  9.8808e-320  2.9643e-319
345000000       1       1   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
210000000       1       2   6.0000e+00   6.0000e+00  9.8808e-320  2.9643e-319
32000000       1       2  4.9407e-320  4.9407e-320  4.9407e-320  4.9407e-320
899000000       1       2   1.0000e+00   1.0000e+00   9.9994e-01   9.9994e-01
616000000       1       2   3.0000e+00   2.0000e+00   3.0000e+00  2.9643e-319
      0       2       3   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
93000000       2       3   1.0000e+00   1.0000e+00   1.0000e+00   1.0000e+00

Loops: 1000000, Iterations: 1, Duration: 38 sec.
C Converted Double Precision Whetstones: 2631.6 MWIPS
```

The Whetstone benchmark measures a computer or server's floating-point performance. The overall performance is calculated by Whetstone Instructions Per Second (WIPS). Through the screenshot, all the duration over the 5 runs of the program are about 38sec, but the second, fourth and fifth program is a little faster than the other four programs. And the Whetstone speed ratings are 2631.6 MWIPS. The difference of execution time and KFLOPS can attribute to memory sharing, cache occupation and core occupation.

b.

Benchmark: LINPACK

O0((do no optimization, the default if no optimization level is specified)),

```
[xin.ji@c0241 linpack]$ make
gcc -Wall -ansi -O0  -lm -DSP ./src/linpack.c -o ./bin/linpacksp
gcc -Wall -ansi -O0  -lm -DDP ./src/linpack.c -o ./bin/linpackdp
[xin.ji@c0241 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
----------------------------------------------------
    2048   0.60  80.00%   5.00%  15.00%  709437.875
    4096   1.18  77.12%   7.63%  15.25%  723626.688
    8192   2.37  78.48%   5.06%  16.46%  730936.188
   16384   4.74  79.54%   5.91%  14.56%  714693.062
   32768   9.46  79.81%   4.65%  15.54%  724532.750
   65536  18.95  78.94%   5.28%  15.78%  725439.625
```

O1((optimize minimally)),

```
[xin.ji@c0241 linpack]$ make
gcc -Wall -ansi -O1  -lm -DSP ./src/linpack.c -o ./bin/linpacksp
gcc -Wall -ansi -O1  -lm -DDP ./src/linpack.c -o ./bin/linpackdp
[xin.ji@c0241 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    0.72  66.67%   9.72%  23.61%  2631370.250
   16384    1.42  73.94%   4.93%  21.13%  2584381.000
   32768    2.86  74.48%   6.64%  18.88%  2495264.000
   65536    5.73  68.94%   6.46%  24.61%  2680100.250
  131072   11.44  69.23%   6.56%  24.21%  2670821.250
```

O2((optimize more)),

```
[xin.ji@c0241 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@c0241 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    0.65  61.54%   3.08%  35.38%  3445839.750
   16384    1.30  68.46%   7.69%  23.85%  2923743.750
   32768    2.61  59.39%   7.28%  33.33%  3327019.750
   65536    5.21  67.95%   4.41%  27.64%  3071093.500
  131072   10.44  66.57%   3.93%  29.50%  3146206.500
```

O3((optimize even more)),

```
[xin.ji@c0241 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@c0241 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
----------------------------------------------------
   16384    0.92  66.30%   5.43%  28.26%  4385616.500
   32768    1.82  54.40%   4.40%  41.21%  5410294.500
   65536    3.66  59.29%   4.10%  36.61%  4990527.500
  131072    7.30  56.85%   5.34%  37.81%  5100440.500
  262144   14.62  58.82%   5.27%  35.91%  4942578.500
```

Ofast((optimize very aggressively to the point of breaking standard compliance))

```
[xin.ji@c0241 linpack]$ make
gcc -Wall -ansi -Ofast  -lm -DSP ./src/linpack.c -o ./bin/linpacksp
gcc -Wall -ansi -Ofast  -lm -DDP ./src/linpack.c -o ./bin/linpackdp
[xin.ji@c0241 linpack]$ ./bin/linpack
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
----------------------------------------------------
    2048   0.58  77.59%   6.90%  15.52%  738394.558
    4096   1.18  78.81%   3.39%  17.80%  746006.873
    8192   2.34  81.62%   2.56%  15.81%  734646.362
   16384   4.68  79.70%   5.77%  14.53%  723626.667
   32768   9.35  80.11%   4.81%  15.08%  729094.878

   65536  18.72  79.43%   5.88%  14.69%  724986.015
```

Benchmark: Whetstone

O0((do no optimization, the default if no optimization level is specified)),

```
[xin.ji@c0186 whetstone]$ make
gcc -ansi -O0  -lm -DPRINTOUT ./src/whetstone.c -o ./bin/whetdc
[xin.ji@c0186 whetstone]$ ./bin/whetdc
      0       0       0   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
12000000 14000000 12000000 -1.7680e-313  2.3567e-314 -1.7679e-313  2.3568e-314
14000000 12000000 12000000 -9.8808e-320  9.8808e-320  9.8808e-320  2.9643e-319
345000000       1       1   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
210000000       1       2   6.0000e+00   6.0000e+00  9.8808e-320  2.9643e-319
32000000       1       2  4.9407e-320  4.9407e-320  4.9407e-320  4.9407e-320
899000000       1       2   1.0000e+00   1.0000e+00   9.9994e-01   9.9994e-01
616000000       1       2   3.0000e+00   2.0000e+00   3.0000e+00  2.9643e-319
      0       2       3   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
93000000       2       3   1.0000e+00   1.0000e+00   1.0000e+00   1.0000e+00

Loops: 1000000, Iterations: 1, Duration: 39 sec.
C Converted Double Precision Whetstones: 2564.1 MWIPS
```

O1((optimize minimally)),

```
      0       0       0   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
12000000 14000000 12000000 -1.7680e-313  2.3567e-314 -1.7679e-313  2.3568e-314
14000000 12000000 12000000 -9.8808e-320  9.8808e-320  9.8808e-320  2.9643e-319
345000000       1       1   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
210000000       1       2   6.0000e+00   6.0000e+00  9.8808e-320  2.9643e-319
32000000       1       2  4.9407e-320  4.9407e-320  4.9407e-320  4.9407e-320
899000000       1       2   1.0000e+00   1.0000e+00   9.9994e-01   9.9994e-01
616000000       1       2   3.0000e+00   2.0000e+00   3.0000e+00  2.9643e-319
      0       2       3   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
93000000       2       3   1.0000e+00   1.0000e+00   1.0000e+00   1.0000e+00

Loops: 1000000, Iterations: 1, Duration: 32 sec.
C Converted Double Precision Whetstones: 3125.0 MWIPS
```

O2((optimize more)),

```
[xin.ji@c0186 whetstone]$ make
gcc -ansi -O2  -lm -DPRINTOUT ./src/whetstone.c -o ./bin/whetdc
[xin.ji@c0186 whetstone]$ ./bin/whetdc
      0       0       0   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
12000000 14000000 12000000 -1.7680e-313  2.3567e-314 -1.7679e-313  2.3568e-314
14000000 12000000 12000000 -9.8808e-320  9.8808e-320  9.8808e-320  2.9643e-319
345000000       1       1   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
210000000       1       2   6.0000e+00   6.0000e+00  9.8808e-320  2.9643e-319
32000000       1       2  4.9407e-320  4.9407e-320  4.9407e-320  4.9407e-320
899000000       1       2   1.0000e+00   1.0000e+00   9.9994e-01   9.9994e-01
616000000       1       2   3.0000e+00   2.0000e+00   3.0000e+00  2.9643e-319
      0       2       3   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
93000000       2       3   1.0000e+00   1.0000e+00   1.0000e+00   1.0000e+00

Loops: 1000000, Iterations: 1, Duration: 28 sec.
C Converted Double Precision Whetstones: 3571.4 MWIPS
```

O3((optimize even more)),

```
[xin.ji@c0186 whetstone]$ make
gcc -ansi -O3  -lm -DPRINTOUT ./src/whetstone.c -o ./bin/whetdc
[xin.ji@c0186 whetstone]$ ./bin/whetdc
      0       0       0   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
12000000 14000000 12000000 -1.7680e-313  2.3567e-314 -1.7679e-313  2.3568e-314
14000000 12000000 12000000 -9.8808e-320  9.8808e-320  9.8808e-320  2.9643e-319
345000000       1       1   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
210000000       1       2   6.0000e+00   6.0000e+00  9.8808e-320  2.9643e-319
32000000       1       2  4.9407e-320  4.9407e-320  4.9407e-320  4.9407e-320
899000000       1       2   1.0000e+00   1.0000e+00   9.9994e-01   9.9994e-01
616000000       1       2   3.0000e+00   2.0000e+00   3.0000e+00  2.9643e-319
      0       2       3   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
93000000       2       3   1.0000e+00   1.0000e+00   1.0000e+00   1.0000e+00

Loops: 1000000, Iterations: 1, Duration: 27 sec.
C Converted Double Precision Whetstones: 3703.7 MWIPS
```

Ofast((optimize very aggressively to the point of breaking standard compliance))

```
[xin.ji@c0186 whetstone]$ make
gcc -ansi -Ofast  -lm -DPRINTOUT ./src/whetstone.c -o ./bin/whetdc
[xin.ji@c0186 whetstone]$ ./bin/whetdc
      0       0       0   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
12000000 14000000 12000000   0.0000e+00   0.0000e+00   0.0000e+00   0.0000e+00
14000000 12000000 12000000   0.0000e+00   0.0000e+00   0.0000e+00   0.0000e+00
345000000       1       1   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
210000000       1       2   6.0000e+00   6.0000e+00   0.0000e+00   0.0000e+00
32000000       1       2   0.0000e+00   0.0000e+00   0.0000e+00   0.0000e+00
899000000       1       2   1.0000e+00   1.0000e+00   9.9994e-01   9.9994e-01
616000000       1       2   3.0000e+00   2.0000e+00   3.0000e+00   0.0000e+00
      0       2       3   1.0000e+00  -1.0000e+00  -1.0000e+00  -1.0000e+00
93000000       2       3   1.0000e+00   1.0000e+00   1.0000e+00   1.0000e+00

Loops: 1000000, Iterations: 1, Duration: 3 sec.
C Converted Double Precision Whetstones: 33333.3 MWIPS
```

The above screenshots show the results of turning on optimization flags of gcc to reduce the cost of compilation and produce an expected result. The optimization strategy I use is changing the -O variable, which can make code compilation take more time and take up much more memory as the increasing of optimization level. The levels I applied of the optimization flag are O0, O1, O2, O3 and Ofast.

O0 is the default setting and turns off optimization totaly. O1 reduces the execution time by turning on a list of optimization flags, such as -fdce, -fdse and -fif-conversion. It is the basic optimization level and produce faster execution results for LINPACK and Whetstone. O2 flag optimizes even more than O1. Except those flags O1 turning on, O2 also turns on another lists of optimization flags based on O1, such as -falign-functions and -falign-jumps. Without taking too much compilation time and compromising on memory, it increases benchmark performance more than O1 and the execution time results are shown above. Similarly, O3 turns on addition optimization flags depending on O2. It produces a better execution time but not recommend because it may slow down a system due to large binaries and memory usage. Ofast not only turns on all O3 optimizations, but also enables some optimizations that are not valid for all standard-compliant programs, like -ffast-math, -fallow-store-data-races and the Fortran-specific -fstack-arrays. It is not a recommended optimization flag either because it breaks strict standards compliance. The Whetstone is executed much better but linpack, which can attribute to the breaking of standards compliance.

c.

Pthreads are kinds of C language programming types by using pthread.h header to take advantage of those additional threads provided by server or PC. Pthread can efficiently increase testing speed and decrease the cost of data exchange.

For benchmark LINPACK, I would use Pthreads to create three threads to execute DGEFA, EGESL and OVERHEAD separately, and summary the results to obtain additional speedup.

For benchmark Stream, it run each kernel "NTIMES" to evaluate and report memory performance and the source code set "NTIMES" as ten. I would take advantage of Pthreads to create 10 threads and each thread execute the program for each kernel for one time at the same time. To conclude, I sum each result, terminate threads and report results in total.

For benchmark Whetstone, it evaluates program running time by execute instruction for "n" times loop. I would take advantage of Pthreads to create like 10 threads and each thread execute the program for "n/10" time. In conclusion, I sum result of each thread, terminate threads and write down totally duration and MWIPS.

2.

a.

```
[Running] cd "/Users/jiayunxin/Desktop/NEU/EECE 5640/Assignment/" && g++ 5640_a1_1.cpp -o 5640_a1_1 && "/Users/jiayunxin/
Desktop/NEU/EECE 5640/Assignment/"5640_a1_1
Run program with 1 thread(s)
Time taken for execution: 0.001243 seconds
Array is in sorted order

[Done] exited with code=0 in 0.168 seconds

[Running] cd "/Users/jiayunxin/Desktop/NEU/EECE 5640/Assignment/" && g++ 5640_a1_1.cpp -o 5640_a1_1 && "/Users/jiayunxin/
Desktop/NEU/EECE 5640/Assignment/"5640_a1_1
Run program with 2 thread(s)
Time taken for execution: 0.001056 seconds
Array is in sorted order

[Done] exited with code=0 in 0.238 seconds

[Running] cd "/Users/jiayunxin/Desktop/NEU/EECE 5640/Assignment/" && g++ 5640_a1_1.cpp -o 5640_a1_1 && "/Users/jiayunxin/
Desktop/NEU/EECE 5640/Assignment/"5640_a1_1
Run program with 4 thread(s)
Time taken for execution: 0.000589 seconds
Array is in sorted order

[Done] exited with code=0 in 0.236 seconds

[Running] cd "/Users/jiayunxin/Desktop/NEU/EECE 5640/Assignment/" && g++ 5640_a1_1.cpp -o 5640_a1_1 && "/Users/jiayunxin/
Desktop/NEU/EECE 5640/Assignment/"5640_a1_1
Run program with 8 thread(s)
Time taken for execution: 0.000417 seconds
Array is in sorted order

[Done] exited with code=0 in 0.159 seconds
```

The screenshot shows that the merge sort program was run with 1, 2, 4 and 8 threads separately on my mac OS system and relevant code is cited from website https://malithjayaweera.com/2019/02/parallel-merge-sort/. Pthreads were used by divided the whole random integers into n threads and evaluate results. Ideally, results display that the execution will take less time as program run with more threads.

b. Challenges:

1) Pick up a sorting method. Not all sorting methods are suitable for using multiple threads.
2) Find a suitable place in merge sort program to use Pthreads. In the process of merge sorting, there are some places can be changed to use Pthreads, but not all places are reasonable.
3) How to assign work to each thread. Pthreads are new things for me, understanding the use and code of Pthreads need take a little more time.

c. Strong scaling means that the upper limit of speedup is determined by the serial fraction of the code. Weak scaling means that there is no upper limit for the scaled speedup and the execution time is calculated based on the amount of work done for a scaled problem size. Through the performance report, the execution time is markedly improved as threads increase, but the improved results are not double. It indicates that the weak scaling properties of my sorting implementation are stronger than the strong scaling. The serial fraction of my sorting method affect result and not determine the upper limit of speedup.

3.

```
[xin.ji@c0186 a1]$ 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:               1200.183
CPU max MHz:           3500.0000
CPU min MHz:           1200.0000
BogoMIPS:              5200.12
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
```

a. The CPU Model: Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz

b.

L1d cache size: 32K
L1i cache size: 32K
L2 cache size: 256K
L3 cache size: 30720K
L1d cache assoc: 8
L1i cache assoc: 8

L2 cache assoc: 8
L3 cache assoc: 20

c. High speed data transfer: over 10 Gbps Ethernet (GbE)
Bandwidth: 10 GbE or a high-performance HDR200 InfiniBand (IB) interconnect running at 200 Gbps

d.

```
[xin.ji@c0186 a1]$ uname -a
Linux c0186 3.10.0-1160.25.1.el7.x86_64 #1 SMP Wed Apr 28 21:49:45 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
```

Linux c0186 3.10.0-1160.25.1.el7.x86_64 #1 SMP Wed Apr 28 21:49:45 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux

4.

Summary of trends:

A supercomputer is composed of processors, memory, I/O system, and an interconnect. Four main factors determine the Top500 list, which are number of cores, Rmax, Rpeak and cost of power. Rmax is the maximum value that the system can perform of the Linpack benchmark while Rpeak represents theoretical maximum number that system can run the Linpack benchmark. Rmax is the dominating factor to rank the Top500 supercomputer. In the top 10 of the TOP500, there are over 20 thousand TFlop/s Rmax and nearly over half million cores number. IBM, AMD and Intel and the three main company providing processors and GPU of NVIDIA was widely used by those top10 supercomputer. Apart from achieving high computing performance, power consumption is an indispensable factor that supercomputer designers must consider. Based on the contemporary data, top10 supercomputer consume exceed 5 megawatts. In the future supercomputer design, high-performance, high-density packaging and low power consumption and heat generation would be a better choice and a new goal to pursue.

My architecture: Supercomputer Jiayun, IBM POWER9 22C 2.8GHZ, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband

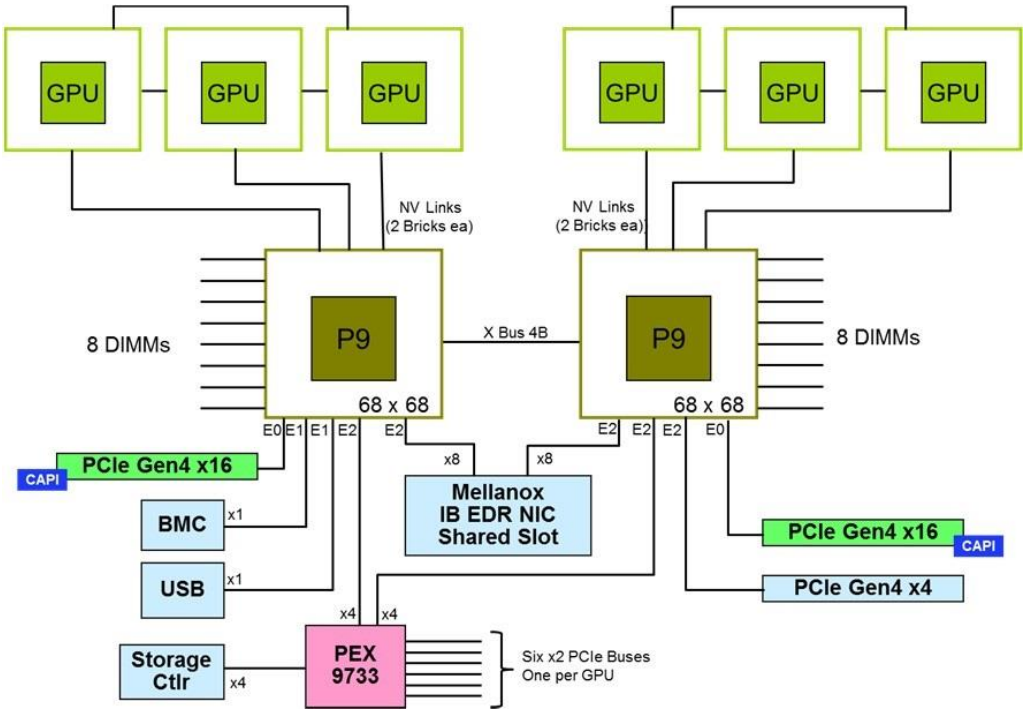


Figure 1

Picture source: https://www.nextplatform.com/2017/12/05/power9-to-the-people/

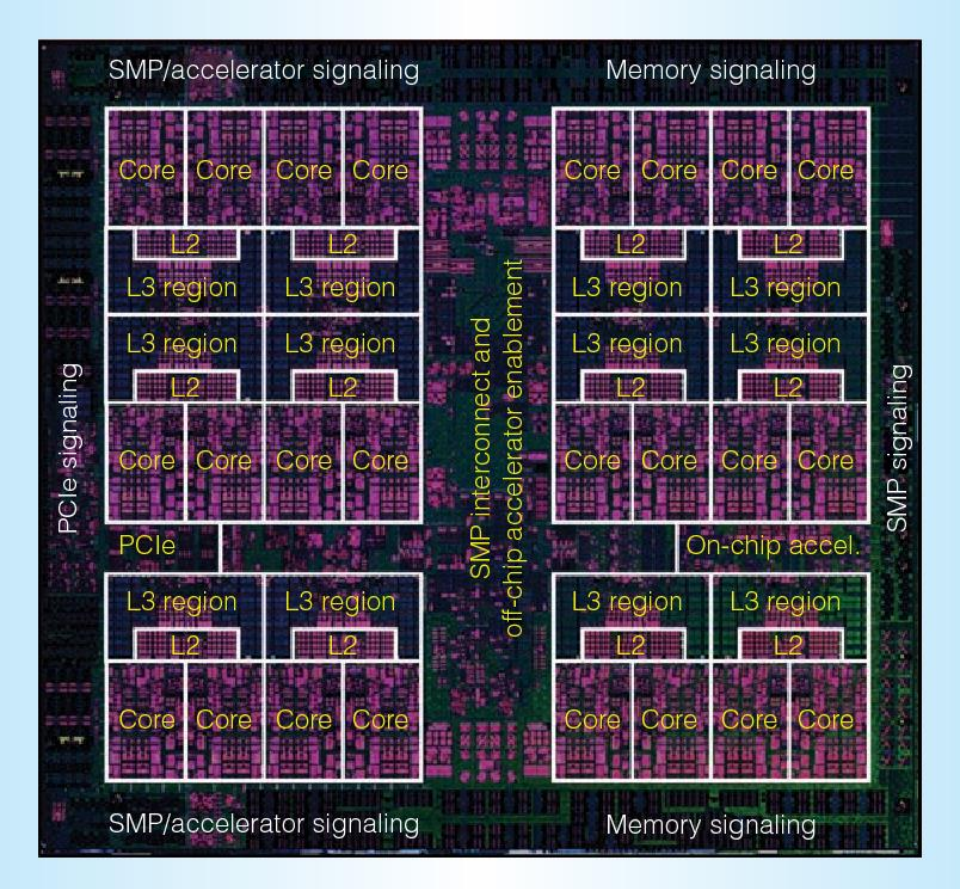


Figure 2

Picture source: https://en.wikichip.org/wiki/ibm/microarchitectures/power9#Memory_Hierarchy

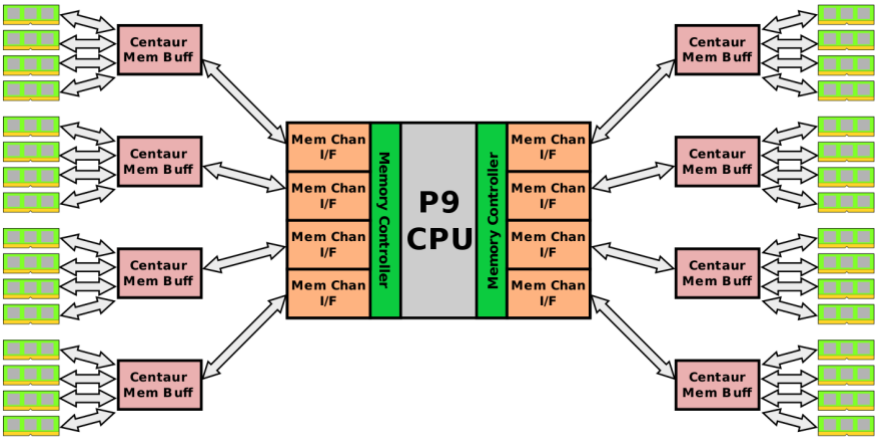
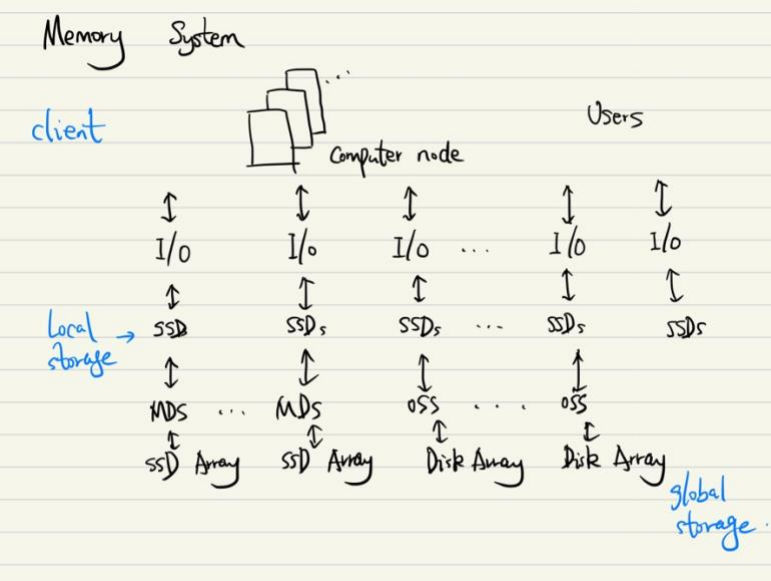


Figure 3

Picture source: https://en.wikichip.org/wiki/ibm/microarchitectures/power9#Memory_Hierarchy



CPU Clock Speed (GHz): 2.8

Nodes: Login Nodes: 5
GPU nodes: 9000
Total nodes: 10000

CPUs: CPU Architecture: IBM Power 9
Cores/Node: 24
Total Cores: 250,000
Min. feature size: 14nm

GPUs: GPU architecture: NVIDIA V100 (Volta)
Total GPUs: 18,000
GPUs per compute node: 4

Memory: Memory total (GB): 1,500,000
CPU memory/Node (GB): 256
GPU Memory/Node (GB): 64
L1 cache: 32+32 KiB per core
L2 cache: 512 KiB per core
L3 cache: 120 MiB per chip
L4 cache: via Centaur

The first picture shows a system board block diagram and the second one displays the CPU architecture I would like to use. The supercomputer I design will have 10,000 nodes, each hold 2 Power 9 CPUs with 24 cores. Overall, about 250,000 cores are used to increase Rmax value. Minimum feature size I choose 14nm to avoid high heat generation caused by smaller size chip and more intensive arrangement. The GPU I choose NVIDIA V100. All nodes are linked together by using Mellanox dual-rail EDR InfiniBand network due to its high bandwidth and low latency.

5.

Differences:

a. Ordering method: The supercomputers in Top500 are ranked by Ramx (TFlop/s) while the Green 500 ranks computer based on power efficiency (GFlops/watts). The power efficiency is calculated by making Rmax dividing power consumption.
b. Number of cores: top 10 systems in Green500 have much less number of cores than the top 10 systems in Top500.
c. Power consumption: top 10 systems in Green500 consumes much less power than the top 10 systems in Top500.
d. CPU: 8 of 10 systems in Green500 list apply AMD EPYC processor while those systems in Top500 use like IBM Power9, Intel Xeon and AMD EPYC.