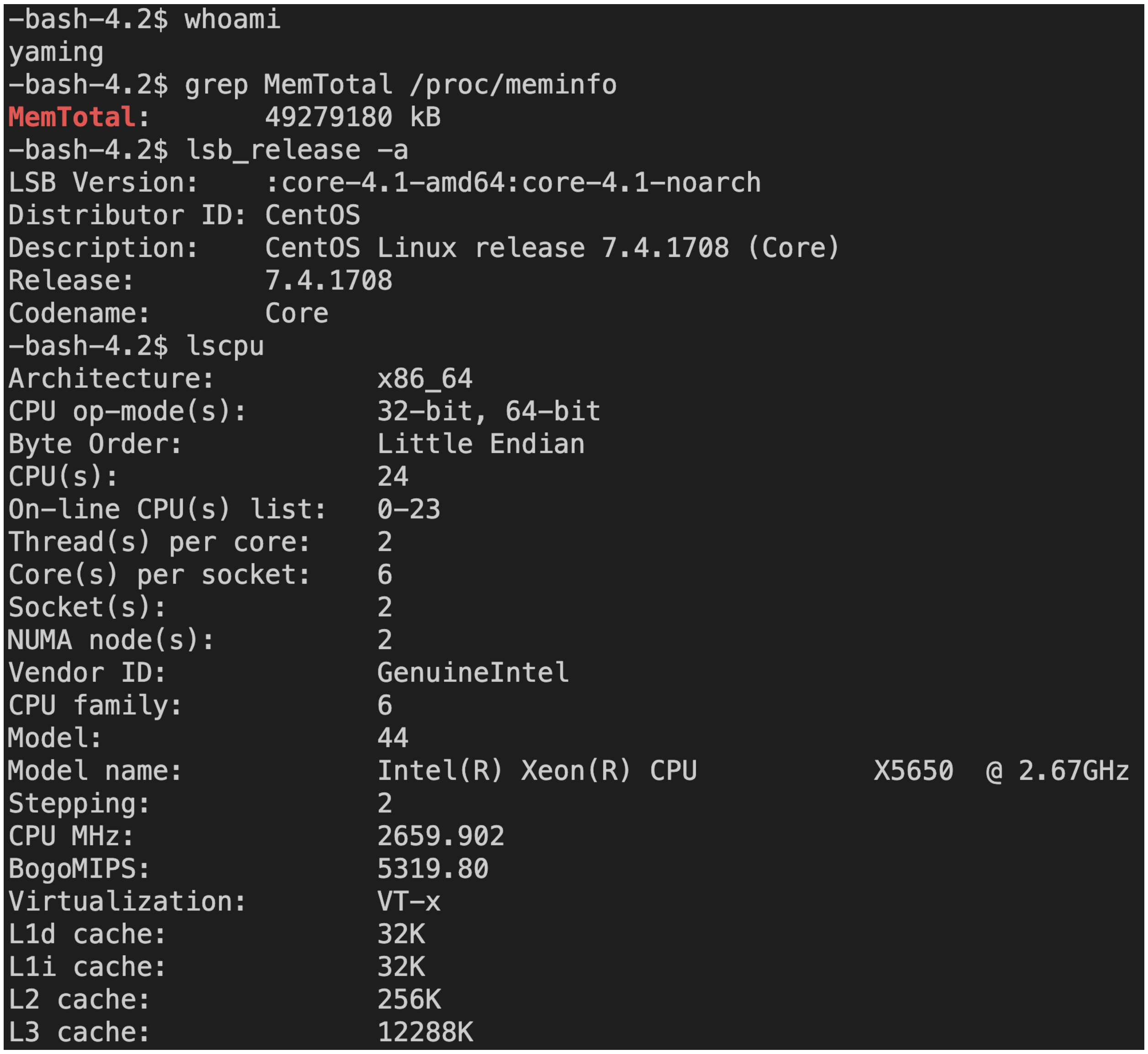
Q1.

Before we start, all the benchmarks are done on COE systems, which have the following information.

Model name	Intel(R) Xeon(R) CPU X5650 @ 2.67GHz
Core(s) per socket	6
CPU(s)	24
MemTotal	49279180 kB
OS version	CentOS Linux release 7.4.1708 (Core)



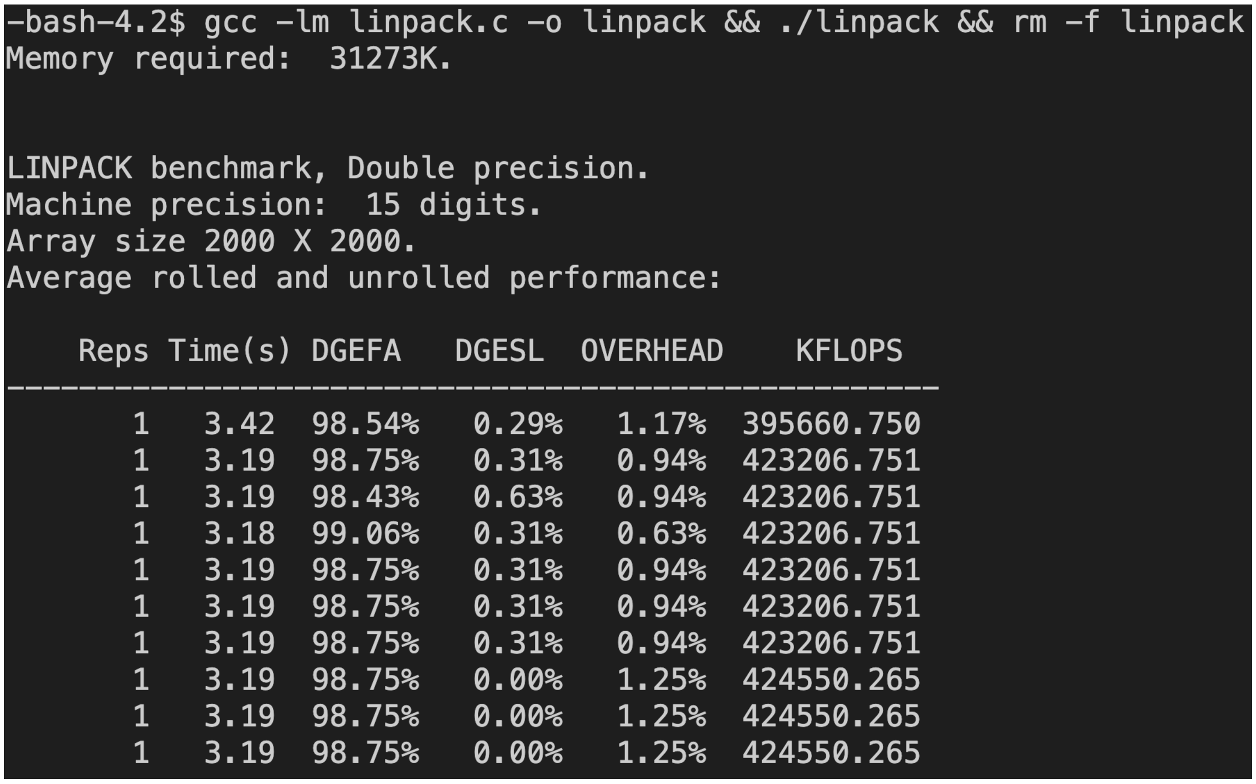
```
-bash-4.2$ whoami
yaming
-bash-4.2$ grep MemTotal /proc/meminfo
MemTotal:       49279180 kB
-bash-4.2$ lsb_release -a
LSB Version:    :core-4.1-amd64:core-4.1-noarch
Distributor ID: CentOS
Description:    CentOS Linux release 7.4.1708 (Core)
Release:        7.4.1708
Codename:       Core
-bash-4.2$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                24
On-line CPU(s) list:   0-23
Thread(s) per core:    2
Core(s) per socket:    6
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 44
Model name:            Intel(R) Xeon(R) CPU           X5650  @ 2.67GHz
Stepping:              2
CPU MHz:               2659.902
BogoMIPS:              5319.80
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              12288K
```

1. **Array size 2000 * 2000 Linpack Benchmark**:

Turn	Running time (s)
1	3.39
2	3.39
3	3.38
4	3.38

5	3.37
6	3.38
7	3.42
8	3.39
9	3.38
10	3.38

Execution time, averaged over 10 runs of the program: 3.386s
If I separately run the benchmark, I do not find any run that is distinctively slower or faster than others. For example, my runtime above is from 3.37s to 3.42s. However, if I write a for loop in the benchmark and loop for 10 times in the benchmark, the first runtime is always slower than others'. I guess it is because in the first loop there is no cache in the OS.

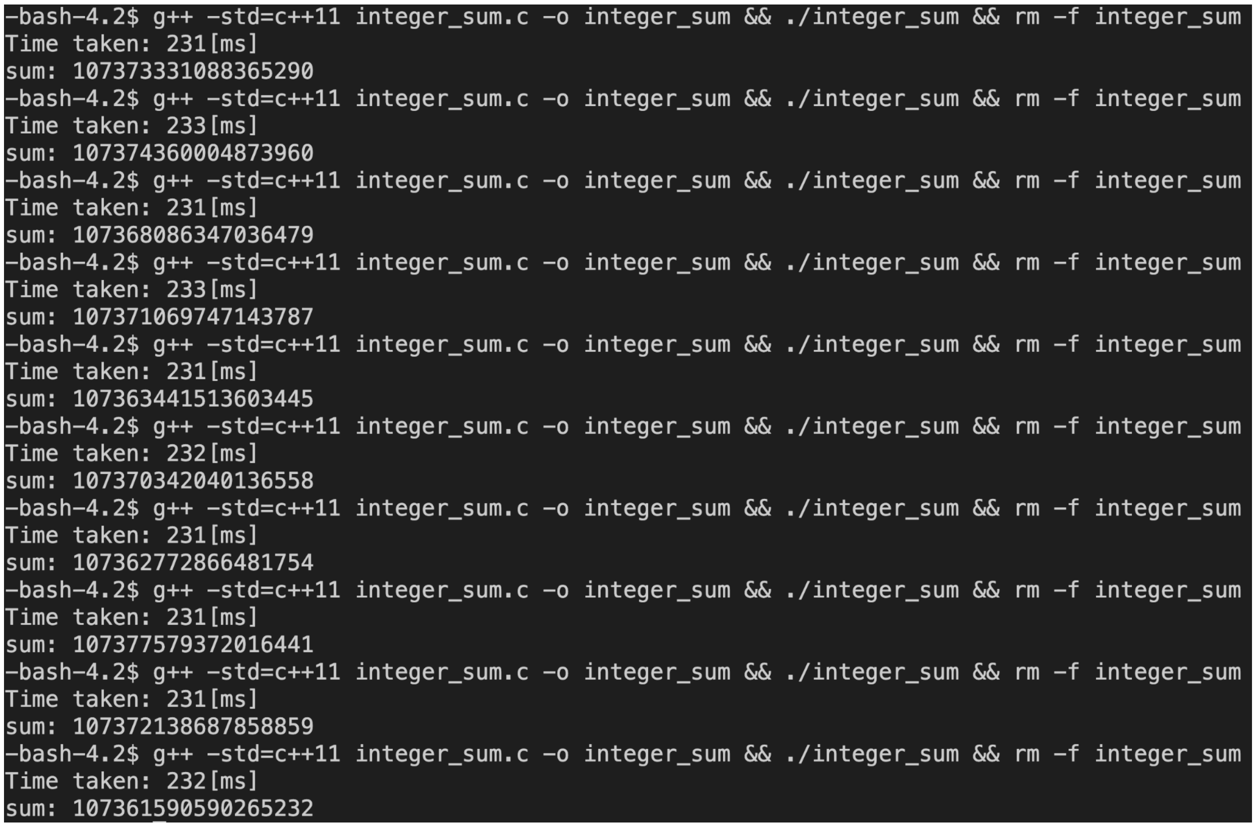


```
-bash-4.2$ gcc -lm linpack.c -o linpack && ./linpack && rm -f linpack
Memory required:  31273K.


LINPACK benchmark, Double precision.
Machine precision:  15 digits.
Array size 2000 X 2000.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
       1   3.42  98.54%   0.29%   1.17%  395660.750
       1   3.19  98.75%   0.31%   0.94%  423206.751
       1   3.19  98.43%   0.63%   0.94%  423206.751
       1   3.18  99.06%   0.31%   0.63%  423206.751
       1   3.19  98.75%   0.31%   0.94%  423206.751
       1   3.19  98.75%   0.31%   0.94%  423206.751
       1   3.19  98.75%   0.31%   0.94%  423206.751
       1   3.19  98.75%   0.00%   1.25%  424550.265
       1   3.19  98.75%   0.00%   1.25%  424550.265
       1   3.19  98.75%   0.00%   1.25%  424550.265
```

Array size 10,000,000 integer sum up:

Turn	Running Time (ms)
1	231
2	233
3	231
4	233
5	231
6	232
7	231
8	231
9	231
10	232

Execution time, averaged over 10 runs of the program: 231.6ms
This is a simple integer sum up benchmark written by myself, the most running time is the cost in generating random integers from 0 to INT_MAX. The running time is just the sum up of the integers in the array, so the testing time is relatively stable.

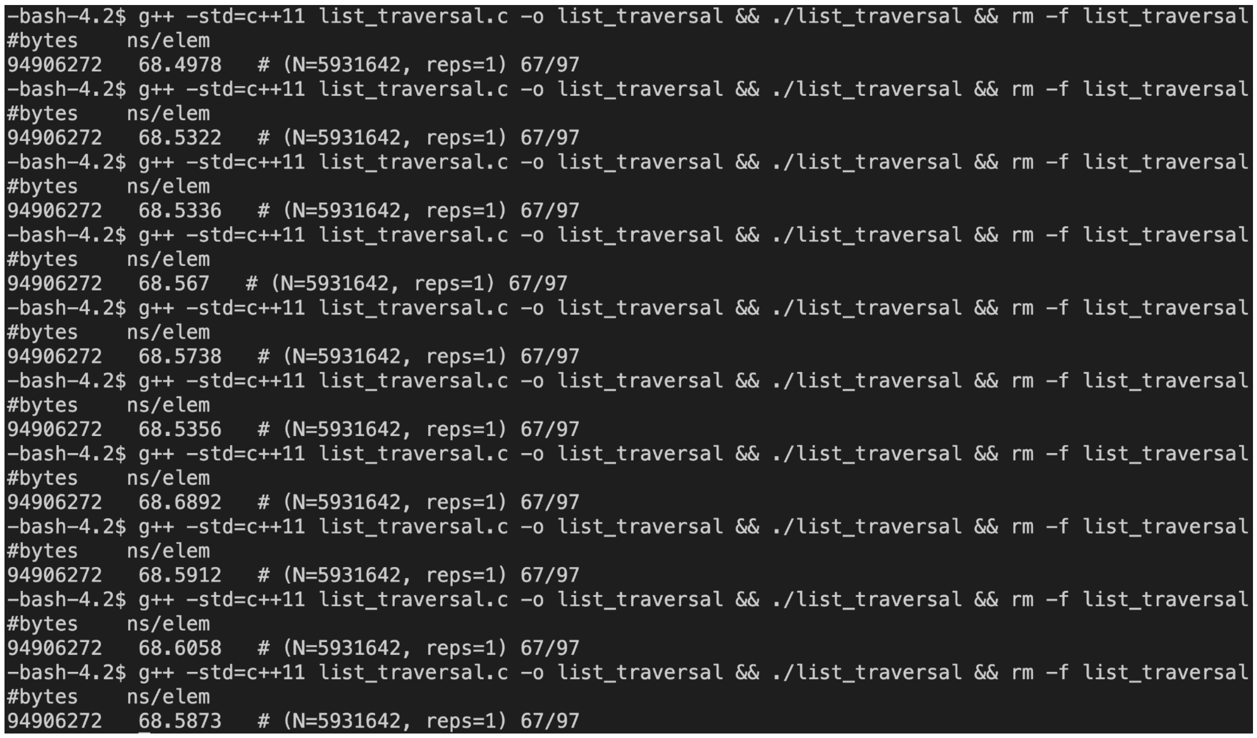


```
-bash-4.2$ g++ -std=c++11 integer_sum.c -o integer_sum && ./integer_sum && rm -f integer_sum
Time taken: 231[ms]
sum: 107373331088365290
-bash-4.2$ g++ -std=c++11 integer_sum.c -o integer_sum && ./integer_sum && rm -f integer_sum
Time taken: 233[ms]
sum: 107374360004873960
-bash-4.2$ g++ -std=c++11 integer_sum.c -o integer_sum && ./integer_sum && rm -f integer_sum
Time taken: 231[ms]
sum: 107368086347036479
-bash-4.2$ g++ -std=c++11 integer_sum.c -o integer_sum && ./integer_sum && rm -f integer_sum
Time taken: 233[ms]
sum: 107371069747143787
-bash-4.2$ g++ -std=c++11 integer_sum.c -o integer_sum && ./integer_sum && rm -f integer_sum
Time taken: 231[ms]
sum: 107363441513603445
-bash-4.2$ g++ -std=c++11 integer_sum.c -o integer_sum && ./integer_sum && rm -f integer_sum
Time taken: 232[ms]
sum: 107370342040136558
-bash-4.2$ g++ -std=c++11 integer_sum.c -o integer_sum && ./integer_sum && rm -f integer_sum
Time taken: 231[ms]
sum: 107362772866481754
-bash-4.2$ g++ -std=c++11 integer_sum.c -o integer_sum && ./integer_sum && rm -f integer_sum
Time taken: 231[ms]
sum: 107377579372016441
-bash-4.2$ g++ -std=c++11 integer_sum.c -o integer_sum && ./integer_sum && rm -f integer_sum
Time taken: 231[ms]
sum: 107372138687858859
-bash-4.2$ g++ -std=c++11 integer_sum.c -o integer_sum && ./integer_sum && rm -f integer_sum
Time taken: 232[ms]
sum: 107361590590265232
```

Average access time of item in linked list:

Turn	Avg access time (ns)
1	68.4978
2	68.5322
3	68.5336
4	68.567
5	68.5738
6	68.5356
7	68.6892
8	68.5912
9	68.6058

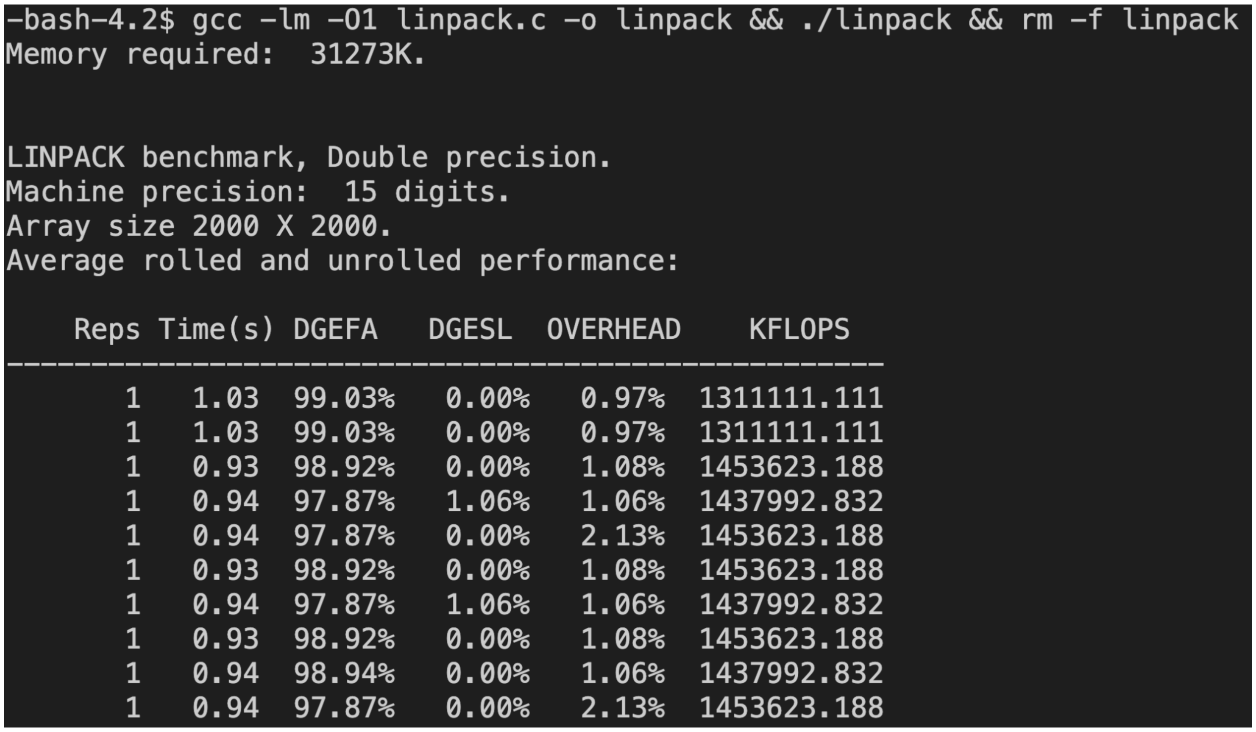
10	68.5873



```
-bash-4.2$ g++ -std=c++11 list_traversal.c -o list_traversal && ./list_traversal && rm -f list_traversal
#bytes    ns/elem
94906272   68.4978   # (N=5931642, reps=1) 67/97
-bash-4.2$ g++ -std=c++11 list_traversal.c -o list_traversal && ./list_traversal && rm -f list_traversal
#bytes    ns/elem
94906272   68.5322   # (N=5931642, reps=1) 67/97
-bash-4.2$ g++ -std=c++11 list_traversal.c -o list_traversal && ./list_traversal && rm -f list_traversal
#bytes    ns/elem
94906272   68.5336   # (N=5931642, reps=1) 67/97
-bash-4.2$ g++ -std=c++11 list_traversal.c -o list_traversal && ./list_traversal && rm -f list_traversal
#bytes    ns/elem
94906272   68.567   # (N=5931642, reps=1) 67/97
-bash-4.2$ g++ -std=c++11 list_traversal.c -o list_traversal && ./list_traversal && rm -f list_traversal
#bytes    ns/elem
94906272   68.5738   # (N=5931642, reps=1) 67/97
-bash-4.2$ g++ -std=c++11 list_traversal.c -o list_traversal && ./list_traversal && rm -f list_traversal
#bytes    ns/elem
94906272   68.5356   # (N=5931642, reps=1) 67/97
-bash-4.2$ g++ -std=c++11 list_traversal.c -o list_traversal && ./list_traversal && rm -f list_traversal
#bytes    ns/elem
94906272   68.6892   # (N=5931642, reps=1) 67/97
-bash-4.2$ g++ -std=c++11 list_traversal.c -o list_traversal && ./list_traversal && rm -f list_traversal
#bytes    ns/elem
94906272   68.5912   # (N=5931642, reps=1) 67/97
-bash-4.2$ g++ -std=c++11 list_traversal.c -o list_traversal && ./list_traversal && rm -f list_traversal
#bytes    ns/elem
94906272   68.6058   # (N=5931642, reps=1) 67/97
-bash-4.2$ g++ -std=c++11 list_traversal.c -o list_traversal && ./list_traversal && rm -f list_traversal
#bytes    ns/elem
94906272   68.5873   # (N=5931642, reps=1) 67/97
```

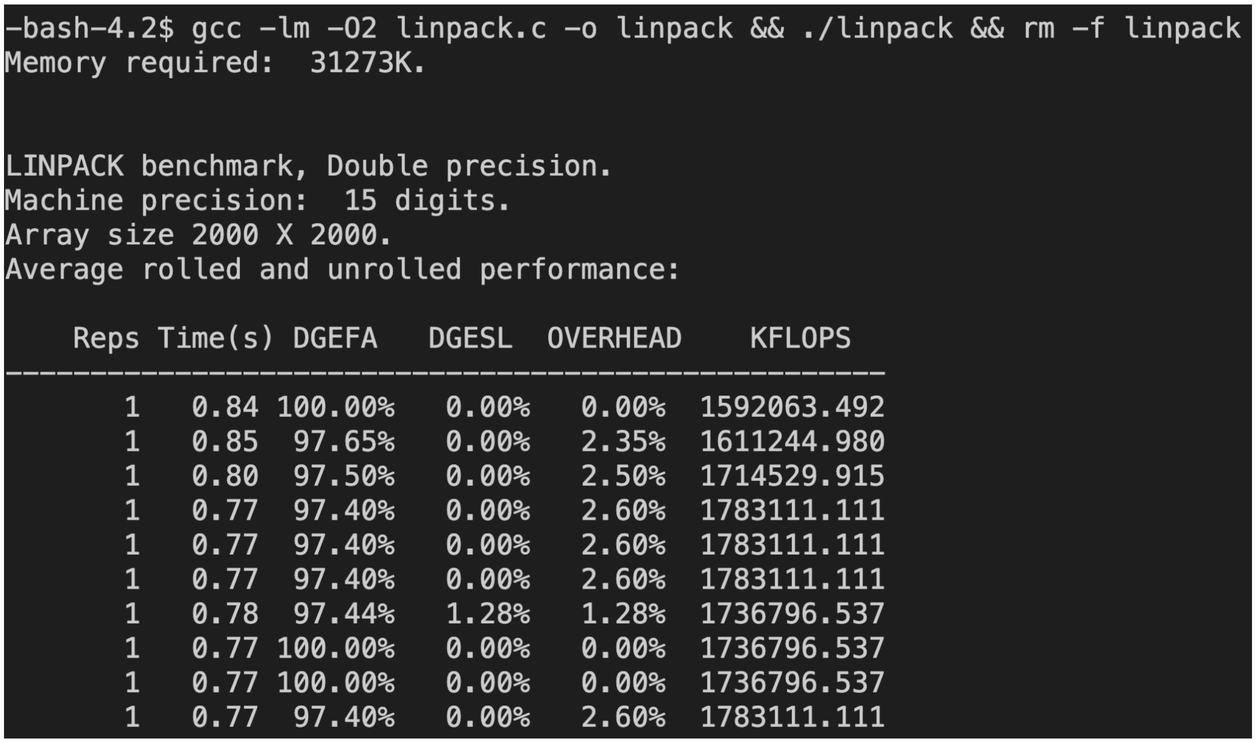
Execution time, averaged over 10 runs of the program: 68.57135ns
This is a linked list traversal benchmark of memory intensive. We generate an amount of items in the linked list and use one thread to traverse all the items in the list. With the increment of items, we found that the access time per element is significantly increased. The reason may be that the rest of the items are stored on the disk so the access time becomes slower.

2. I add optimization on gcc by adding -O option. After adding the -O1 option, the runtime of Linpack is enhanced by nearly 80%. As is shown in the figure below.



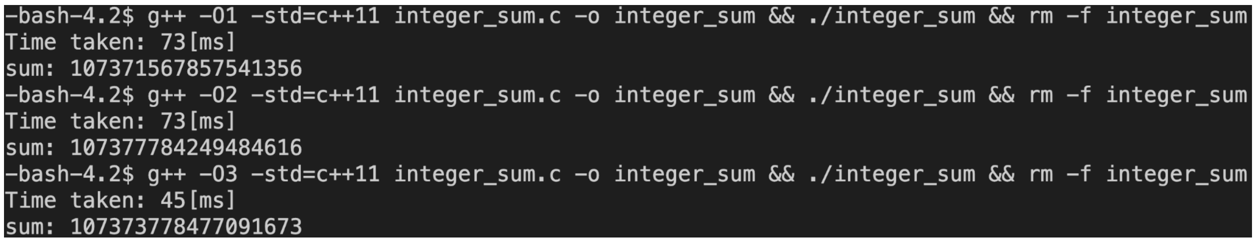
```
-bash-4.2$ gcc -lm -O1 linpack.c -o linpack && ./linpack && rm -f linpack
Memory required:  31273K.


LINPACK benchmark, Double precision.
Machine precision:  15 digits.
Array size 2000 X 2000.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
       1   1.03  99.03%   0.00%   0.97%  1311111.111
       1   1.03  99.03%   0.00%   0.97%  1311111.111
       1   0.93  98.92%   0.00%   1.08%  1453623.188
       1   0.94  97.87%   1.06%   1.06%  1437992.832
       1   0.94  97.87%   0.00%   2.13%  1453623.188
       1   0.93  98.92%   0.00%   1.08%  1453623.188
       1   0.94  97.87%   1.06%   1.06%  1437992.832
       1   0.93  98.92%   0.00%   1.08%  1453623.188
       1   0.94  98.94%   0.00%   1.06%  1437992.832
       1   0.94  97.87%   0.00%   2.13%  1453623.188
```

Then I add the -O2 option, which is the class 2 of optimization. The running time is also enhanced. But when I set gcc to -O3, the improvement is not so significant. So I think the optimization flag of class 2 is enough for the parallel of Linpack.



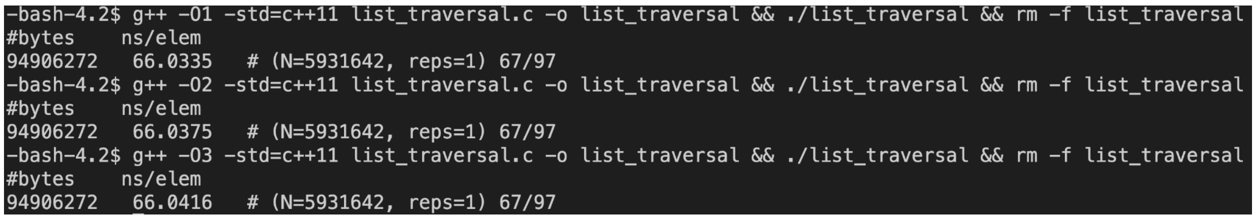
```
-bash-4.2$ gcc -lm -O2 linpack.c -o linpack && ./linpack && rm -f linpack
Memory required:  31273K.


LINPACK benchmark, Double precision.
Machine precision:  15 digits.
Array size 2000 X 2000.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
       1   0.84 100.00%   0.00%   0.00%  1592063.492
       1   0.85  97.65%   0.00%   2.35%  1611244.980
       1   0.80  97.50%   0.00%   2.50%  1714529.915
       1   0.77  97.40%   0.00%   2.60%  1783111.111
       1   0.77  97.40%   0.00%   2.60%  1783111.111
       1   0.77  97.40%   0.00%   2.60%  1783111.111
       1   0.78  97.44%   1.28%   1.28%  1736796.537
       1   0.77 100.00%   0.00%   0.00%  1736796.537
       1   0.77 100.00%   0.00%   0.00%  1736796.537
       1   0.77  97.40%   0.00%   2.60%  1783111.111
```

As for the Integer sum up, obviously it could be optimized by divide and conquer by multiple threads. I also added the -O option to optimize the binary code. Here is the result. Tackling this kind of simple sum up, using option -O1 is enough for the improvement of our code. Using -O2, the running time does not change. However, after we add -O3 to gcc, there is still a huge gap in the running time. The total running time of 10,000,000 integers summed up is only 45 ms.



```
-bash-4.2$ g++ -O1 -std=c++11 integer_sum.c -o integer_sum && ./integer_sum && rm -f integer_sum
Time taken: 73[ms]
sum: 107371567857541356
-bash-4.2$ g++ -O2 -std=c++11 integer_sum.c -o integer_sum && ./integer_sum && rm -f integer_sum
Time taken: 73[ms]
sum: 107377784249484616
-bash-4.2$ g++ -O3 -std=c++11 integer_sum.c -o integer_sum && ./integer_sum && rm -f integer_sum
Time taken: 45[ms]
sum: 107373778477091673
```

As for the average access time of items in the linked list, we also added -O option to gcc and here is the result. It seems that the -O1 option is great enough for us to optimize because the average access time does not decrease when we set -O2 or -O3.



```
-bash-4.2$ g++ -O1 -std=c++11 list_traversal.c -o list_traversal && ./list_traversal && rm -f list_traversal
#bytes    ns/elem
94906272   66.0335   # (N=5931642, reps=1) 67/97
-bash-4.2$ g++ -O2 -std=c++11 list_traversal.c -o list_traversal && ./list_traversal && rm -f list_traversal
#bytes    ns/elem
94906272   66.0375   # (N=5931642, reps=1) 67/97
-bash-4.2$ g++ -O3 -std=c++11 list_traversal.c -o list_traversal && ./list_traversal && rm -f list_traversal
#bytes    ns/elem
94906272   66.0416   # (N=5931642, reps=1) 67/97
```

3. As for Linpack, I'd like to add threads to help with the calculation for the matrix data. Since the calculation of the matrix result is independent of each other, this is a great opportunity to run parallel threads to calculate. As for integer sum up, it is obvious that we could use up to # of elements in array / 2 threads to sum up all the adjacent elements and then divide the number of threads by 2 to calculate the following sum up. As for the average access time of items in the linked list. We could use two quick-slow pointers first to find the middle of the linked list. After that, we could use 2 threads to traverse the linked list from start and middle separately to obtain additional speedup.

References

1. Array size 2000 * 2000 Linpack Benchmark: https://netlib.org/benchmark/linpackc.new
2. Average access time of item in linked list: https://github.com/emilk/ram_bench
3. Generate uniformly distributed pseudo random integers: https://en.cppreference.com/w/cpp/numeric/random/uniform_int_distribution
4. Measure cpp elapsed time: https://stackoverflow.com/questions/2808398/easily-measure-elapsed-time