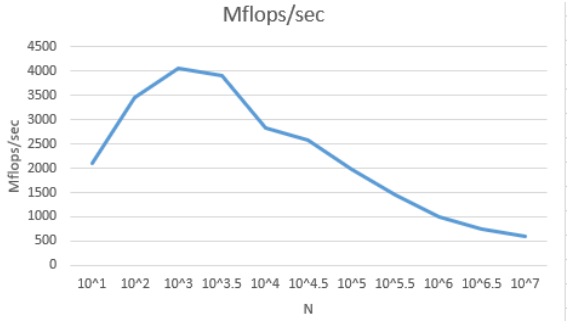
HW1 report

Author: Kangdong Yuan

Introduction: In task1, the goal is writing the benchmark in c code and running it on a cluster. The purpose of this benchmark is to measure the performance of data transfers between memory and arithmetic units of a processor. The N is the size of arrays. The MFlops/sec variable is computed to be the MFlops divided by sec rate for the whole loop nest. The researcher needs to try the different scale of N to get the performance graphs for the vector triad. And in task2, the condition comparison has been added to the program. Moreover, the array C has been initialized by three cases: (a) positive values only (b) negative values only (c) random values between −1 and 1. Then, the researcher needs to run the benchmark with N at L1 cache, L2 cache, and memory size. And the test environment is CPU: Intel Xeon E5-2680 v2 @ 2.80GHz 16 cores, L1 cache: 32KB, L2 cache 256KB, L3 cache:20MB; DRAM:256GB; system: Red-hat.



Observation: For the data in L1 cache size (10^1-10^3), the MFlops are very high and get the max for N=10^3. But after the data fit L2 cache size (10^3.5 -10^4), the MFlops go down drastically. For the data in L3 cache size (10^4.5 -10^6.5), the MFlops decrease steadily to a very low level. Finally, for data in memory level (10^7), the MFlops is lowest at this graph.

Explanation: We can see that the MFlops/sec related to the scale of the N. N is the size of the arrays and it is the number of computations in each inner loop. Each iteration in loop performs 3 loads and 1 store, which are all related to data transfer performance between memory and arithmetic units.

```
11    do i=1,N
12      A(i) = B(i) + C(i) * D(i)    ! 3 loads, 1 store
13    enddo
```

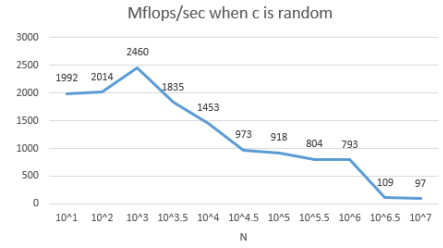
So, the data access and transfer speed are deterministic factors for MFlops/sec. There are several levels of memory: L1 cache, L2 cache, L3 cache, memory. The recently used data will be stored in memory for future use. The latency of data access and transfer increase from L1 cache to memory. If N is small, the size of recently used data can be stored in nearest caches, and the data request can be satisfied with low latency. For example, the MFlops/sec above the 2000 when the size of array is no greater than 10^3.

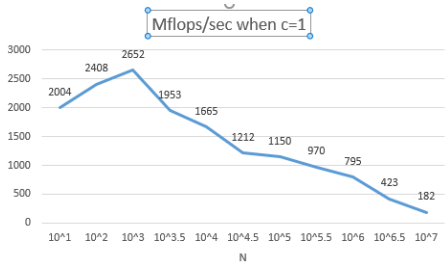
However, the capacity of caches is limited. When N increases, the cache cannot store recent used data, data must be fetched from outer cache even from DRAM. The latency and speed of data access and transfer from DRAM are very slow compared to cache. Moreover, CPU's arithmetic performance will be affected because of the slow speed of data flow. For

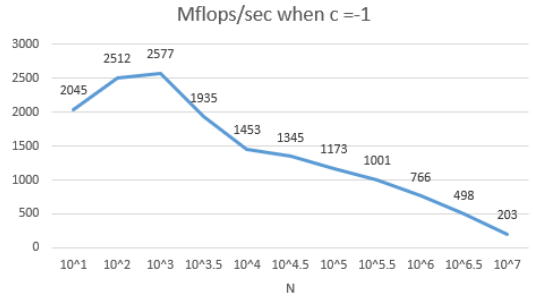
example, when N equals 10^3.5, the L1 cache cannot store all data. The arithmetic units have to access data from L2 cache, so MFlops/sec drop dramatically. The MFlops/sec is also very low, when N=10^7, because the cache cannot store the data. The data transferred to the DRAM, it causes huge latency and low speed for data low between memory and arithmetic units.

Taks2

Observation







Observation: we can infer from the table that the MFlops/sec is lower when C is random initialized, starting from N at L1 cache size. However, when N at memory size (N=10^7), the difference between different initializations increases. But we can see that the MFlops between c = 1 and c=-1 are not large, or we can say they are at the same level. But the performance has the same trend as task1, because the MFlops decrease greatly between the gap of L1, L2, L3 cache, and memory level.

Explanation: The phenomena that performance is bad when array C was random initialized with range between -1 and 1 can be explained by branch prediction. When the CPU computes the conditional branch, it will do the branch prediction to predict the most possible value before accessing the actual value. The predictions of branches are based on statistical methods that related to the data in the past. If the prediction is true, the instruction will be fed into the pipeline. But, if the prediction is wrong, the pipeline has to be rolled back to the position of the branch, which means huge cost on time. The penalty of misprediction will impair performance of the CPU. In the benchmark of task2, if C is random initialized with positive and negative values, it is hard to predict the next value by statistical method. The misprediction will happen very frequently with random initialization, so the performance will be hindered by misprediction. It is consistent with the observation that an array with random initialization has much lower MFlops/sec.