Version 0  
**Vector vector multiplication**  
 Vector Vector multiplication is the simplest case to study how the data size increase can affect FLOPS/s and memory bandwidth. For calculating average time, I count 100 times and average it.

When data size increase, since in the situation 2 in the graph, caches successful hold the data and the transportation times delay cause by RAM is minimized. In situation 3, caches insufficiently hold the data since data size bigger than caches size, then the FLOPS decrease.

The above case is similar to bandwidth reading. When the data size is in situation 2, the array is capable fit into the caches and is saturate at that value is bound by cache bandwidth. After that bw\_r drop in situation 3 since the array is out of the caches size.

So how about the bandwidth of storing data? When the array is huge that bw\_w decays. First of all, if data already resides in cache that will occur write hit. But when the size increase, cache is not capable to handle and occurs a write miss then emit a write allocate, then it slow down the speed and writing to memory is inefficient.   
**Matrix vector multiplication**  
Basically, matrix vector multiplication is similar to vector vector multiplication.   
For calculating average time, I count 100 times and average it.

|  |  |  |  |
| --- | --- | --- | --- |
| Nr1 | Nc1 | Nr2 | Nc2 |
| 8 | 256 | 256 | 1 |
| size | flops | Mem\_r | Mem\_w |
| 18.0625 | 4096 | 4096 | 8 |

The red circle indicates a particular situation. The first guess is that the flops/s, bw\_r and bw\_w had a peak is because the magic number X-byte boundary. Then we can find other magic number combination that they are also at the peak. Then the guess maybe is true.

|  |
| --- |
|  |
|  |

For this case, when we compare bw\_r and bw\_w, the red circle resides a good proposition that have a good bw\_w and also a good bw\_w. Meaning that, when the size is not large and the size is magic number, that the size well fit in caches, then the bandwidth of read and write both have a good performance.  
**Matrix matrix multiplication** For calculating average time, I count 50 times and average it.

Since vector vector multiplication is 2 arrays memory allignment continuous (ex. 1\*n or n\*1, one row instead) and matrix vector multiplication is one array continuous (n\*1, one column instead), matrix matrix multiplication is complicated that is hard to reach their performances due the data size is large and of course, time expensive. Some biases in this part is that the mem\_size is not growing by double, it is just randomly allocate the size. Perhaps some effect cause by the double size growing can’t be found here.

From my observations, the trend still is similar to previous graph, but there’re some points we can discuss. The red circles indicates the caches line size that can’t fit the array size, that is the worse situation that the array size just exceed the caches size a little bit that have a worse performance. After that, the array size continues increase, and then array can fit into caches better than the red circles one. Eventually have this kind of graph.

That’s no any special from mem\_size vs bw\_w, that because the size is too large, then the bandwidth decays, on the other hand the curve is smoothier than previous two, maybe is cause by the mem\_w increase faster than the previous two and increase bw\_w.   
For the case of bw\_r vs bw\_w, the peak of bw\_r is maintain at a high value longer than the previous vector-vector and matrix-vector multiplication. Similar to mem\_size vs bw\_w.

**Compare Benchmarks  
Best Benchmarks:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | nr1 | nc1 | nr2 | nc2 | mem\_size(KB) | norm | secs | flops | mem\_r |
| vv | 1 | 32768 | 32768 | 1 | 512.008 | 1.30E+21 | 0.000158 | 65536 | 65536 |
| mv | 18 | 8192 | 8192 | 1 | 1216.14 | 1.39E+21 | 0.00071 | 294912 | 294912 |
| mm | 60 | 2500 | 2500 | 60 | 2371.88 | 6.01E+21 | 0.042162 | 1.80E+07 | 1.80E+07 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | mem\_w | flops/s | flops/s (MFLOPS) | bw\_r (MB/sec) | bw\_w (MB/sec) |
|  | Vv |  |  |  | 1 | 4.15E+08 | 4.15E+02 | 3169.32 | 0.04836 |
|  | Mv |  |  |  | 18 | 4.15E+08 | 4.15E+02 | 3167.73 | 0.193343 |
|  | mm |  |  |  | 3600 | 4.27E+08 | 4.27E+02 | 3257.18 | 0.651437 |

1. Theoretical bandwidth of cluster: 25.6GB/sec  
   Theoretical Peak of FLOPS: 26.4GFLOPS
2. VV performance  
   bw\_r : 3.0950 GB/sec -> 12.09% usage of bandwidth  
   flops/s: 4.15E+02 MFLOPS -> 1.57% usage
3. MV performance  
   bw\_r : 3.0935 GB/sec -> 12.084% usage of bandwidth  
   flops/s: 4.15E+02 MFLOPS -> 1.57% usage
4. MM performance  
   bw\_r : 3.1808 GB/sec -> 12.43% usage of bandwidth  
   flops/s: 4.27E+02 MFLOPS -> 1.62% usage

As the above graph, we can separate these 3 levels (VV, MV, MM) to 3 steps (Start, Peak, Saturate).  
In start, 3 levels have different characteristic. VV FLOPS arise when the array size increase to fit in caches size. The first peak of MV is cause by the magic number, after that the size is just exceed caches size a little bit that make whole performance drop. MM smoothly increase because of array sizes (? Contradict with MV result, why is the size increase but the curve tend to stable?).   
In peak, these 3 levels almost the same to reach the same performance. The distinction between 3 levels is how long it will stay in this step. No doubt that VV is longest, but MM surprisingly stay longer than MV.  
In Saturatation, VV and MV becomes steady but MM unstable.

The above graph saying the relation about bandwidth read and writes. Once the two arrays sizes increase (m\*n, n\*p matrix), the more communication we need to read and write (m\*p matrix) from or to memory. Then the curve will smooth if require of writes increase.

**Optimization**

**VV multiplication:**version 0: 0%  
version 1: loop unrolling -> haven’t improve anything (maybe done by compiler)

**MV multiplication:**Version 0: 0%  
Version 1 with loop unrolling, haven’t improve anything (maybe done by compiler)

**MM multiplication:**version 0: 0%  
version 1 with block algorithms, trying block size from 1-16, the best block size between 11-13   
-> nr1=256 nc1=100000 nr2=100000 nc2=256, improve 0.79% time  
version 2 with MKL blas, compare with version 0 -> 66 7000 7000 66, improve 23.02% time  
 compare with version 1 -> 66 7000 7000 66, improve 22.13% time  
=> Even using MKL blas, the performance didn’t reach what teacher said 80% or above performance.