Lab 1 Part 1 Report: Analytical Modeling

Hanqing Zhu(hz6696)
Siyuan Ma(sm73423)
Yigong Qin (yq2424)

1 QUESTION 1

In this section, we would elaborate the formulae for the locality of the original, cache-aware and cache-obvious implementations for one level of locality, i.e., L1 cache.

First, we donate the matrix multiplication as follows.

$$C = AB \tag{1}$$

where each size of A, B, C is $N \times N$ respectively for simplification.

Then the definition of locality can be seen as follows:

$$locality = \frac{\#words\ serviced\ by\ the\ local\ storage}{total\ \#accessed\ words} \tag{2}$$

Before we discuss the locality details for different model, we list following assumptions.

- ONLY one-level memory hierarchy is held, which is L1 cache.
- The L1 cache has a size of Z, which is much smaller than the size of matrices, i.e., it is impossible to put all entries of three matrices into local storage.
- Only consider load/write operations on elements of matrices, i.e., we assume all temporary variables are stored in register.
- The L1 cache is as an ideal cache (i.e., LRU replacement policy, fully associative, one word per cache line).

1.1 Triply-nested ijk loop

First, we discuss the original matrix multiplication algorithm. As we only consider load/write element of matrices A, B, C, the overall access efforts $TotalAccess_{ijk}$ is $4N^3$.

Then the total number of cache hit is discussed in the following cases:

- **Case I** $Z \leq 3$
  When cache size Z is very small such that it cannot hold $C[i][j]$ plus two values $A[i][k], B[k][j]$, no data reuse could be gained and $locality$ is 0.

$$Locality_{ijk} = 0 \tag{3}$$

- **Case II** $3 < Z < N + 2$
  When cache size Z can contain $C[i][j]$ plus two values to compute the inner loop, entries of A and B are still un-reused but fetched value of C is reused within inner loop. During each loop of k, c_{ij} is reused $2N - 1$ times. Thus total reuse is $N(2N - 1)$ times. If assuming $Z << N$, the locality keeps approximately constant as 1/2.

$$\begin{aligned} Locality_{ijk} &= \frac{N^2(2N-1)}{4N^3} \\ &= \frac{2N^3 - N^2}{4N^3} \\ &= \frac{1}{2} - \frac{1}{4N} \end{aligned} \tag{4}$$

- **Case III** $N + 2 \geq Z < N^2 + N + 1$
  When cache size Z can hold one row of A such that the row of A can be reused in the inner loop, we gain extra reuse on one row of A. Each row of A is accessed N times, among which $N - 1$ times of accesses could reuse the entire row in cache. That forms totally $N \cdot N(n-1)$ times of extra reuses. Total locality is

$$\begin{aligned} Locality_{ijk} &= \frac{N^2(2N-1) + N \cdot N(N-1)}{4N^3} \\ &= \frac{3N^3 - 2N^2}{4N^3} \\ &= \frac{3}{4} - \frac{1}{2N} \end{aligned} \tag{5}$$

- **Case IV** $N^2 + N + 1 \leq Z$
  When cache can hold one row of A, all of B and one element of C, we gain extra reuses on B. Each element of B is reused $N - 1$ times, so total extra reuses are $N^2(N-1)$ times. Total locality is

$$\begin{aligned} Locality_{ijk} &= \frac{N^2(2N-1) + N^2(N-1) + N^2(N-1)}{4N^3} \\ &= \frac{4N^3 - 3N^2}{4N^3} \\ &= 1 - \frac{3}{4N} \end{aligned} \tag{6}$$

1.2 Cache-aware algorithm

We donate the block size is b and we use the blocked version of matrix multiplication. For simplicity, we also assume that $N >> B$ so that reuse of A and B between blocks are eliminated.

As we divide the original matrix multiplication as blocked version to better suit the cache size, we have optimization for cache access within block multiplication but the total number of data access is $\left(\frac{N}{b}\right)^2 \left(\frac{N}{b}\right)(4b^3) = 4N^3$.

Since in cache-aware algorithm, we can always set block size $b < \sqrt{\frac{Z}{3}}$ such that L1 cache can at least hold three blocks.

- **Case I** $Z >= 3$
  The minimum block size is 1. When cache size $Z >= 3$ it can contain all three blocks for A, B, C. Thus each block multiplication has $4b^3 - 3b^2$ times of reuses, which is multiplied by $\left(\frac{N}{b}\right)^3$ times of block multiplications. Besides, each block of C could be reused $\frac{N}{b} - 1$ times among multiplication of 1 row of blocks in A and 1 column of blocks in B. The number

of blocks in C is $\left(\frac{N}{b}\right)^2$. Thus, we have total locality as

$$\begin{aligned} Locality_{block} &= \frac{\left(\frac{N}{b}\right)^3 (4b^3 - 3b^2) + (\frac{N}{b} - 1)b^2 \left(\frac{N}{b}\right)^2}{4N^3} \\ &= 1 - \frac{1}{2b} - \frac{1}{4N} \\ &\approx 1 - \frac{1}{2b} \end{aligned} \tag{7}$$

- **Case II** $Z < 3$
  No locality could be obtained in such circumstance. We have

$$Locality_{block} = 0 \tag{8}$$

Of course, after dividing the matrices into blocks, we can still treat the block as an entity and further discuss the locality such as when the L1 cache can load one row of blocks of A, i.e., $\left(\frac{N}{b}b^2\right) = Nb$. This procedure is the same with Question 1 and we can always set the b larger to increase locality.

1.3 Cache-oblivious algorithm

Cache-oblivious algorithm is a divide-and-conquer algorithm contains no information of cache size. For simplicity, we will assume A, B, C matrix are all square matrix of size N. Also, to fit all A, B and C into cache, the constant cache size should satisfy $Z \geq \alpha N^2$, we will assume $\alpha = 3$

- **Case I** $Z \geq 3N^2$
  When cache size Z is large enough to fit all the matrix, the cache misses are $3N^2$. The recursive calls of the algorithm will always be a hit after matrices are stored in the cache. So the locality is the same as we got before in **Case IV** of ijk loop.

$$\begin{aligned} Locality_{cache\ oblivious} &= \frac{4N^3 - 3N^2}{4N^3} \\ &= 1 - \frac{3}{4N} \end{aligned} \tag{9}$$

- **Case II** $Z < 3N^2$
  When the A, B, C matrix cannot fit into cache simultaneously, the cache-oblivious algorithm will be recursively called to half the longest dimension of matrices. The cache misses due to recursions are same as the cache-aware algorithm where we know the block size of matrix. We set the block size to be $b = \sqrt{\frac{Z}{3}}$, the cache misses of recurrence is $3\frac{N^3}{b}$.

$$\begin{aligned} Locality_{cache\ oblivious} &= \frac{4N^3 - 3N^2 - \frac{3N^3}{b}}{4N^3} \\ &= 1 - \frac{3}{4N} - \frac{3}{4b} \end{aligned} \tag{10}$$

1.4 Visualization of locality formulate

1.4.1 Triply-nested ljk loop. Figure 1 shows the relationship between L1 locality and L1 cache size under 4 different matrix sizes. We can observe stages levels of Locality within each matrix size. This is corresponding to the previously discussed 4 cases. Note that different matrix sizes causes different edge of cache size, and within the same stage, there are slight difference between matrix sizes. This is because both edge conditions and locality within each stage is a function of matrix size. Figure 2 illustrates the change of locality regarding to the increasing matrix size, which makes more sense to measuring locality on a fixed cache size. Larger cache is guaranteed to have better locality, and within each cache size, the peak locality happens at an edge case. When matrix is small, it falls into case 4. When matrix is large, it falls into case 2 where locality is close to 0.5. There is no locality at all with cache size 2.

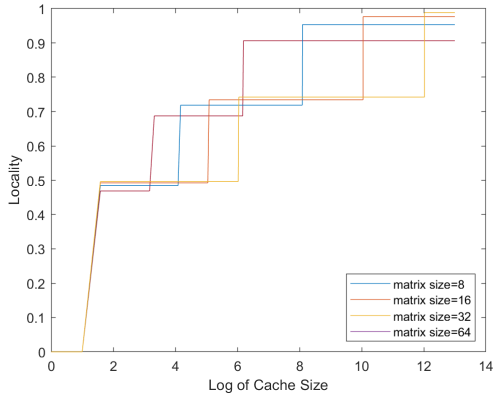

Figure 1: Locality vs Cache Size in Triply-nested ijk Algorithm

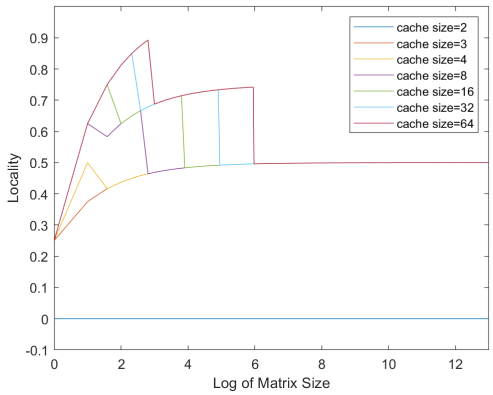

Figure 2: Locality vs Matrix Size in Triply-nested ijk Algorithm

1.4.2 Cache-aware algorithm. If cahce size is larger than 3, We can always make block size small enough so that all 3 blocks simultaneously fit in the cache. We assume the matrix is far larger than our cache so that there is no locality among different blocks. In such circumstance locality is independent of matrix size, and is only a function of block size, which is further a function of cache size. Figure 3 illustrates such function. Locality is 0 if cache is smaller than 3. In extreme conditions where cache is so small that it only contains block size=1, the cache-aware algorithm degrades to the triply-ljk, and locality degrades to 1/2, which is the same as ijk **Case II**. When cache size keeps increasing, we can observe that there are mini steps where locality stays the same until cache grows larger enough to contain 1- element larger blocks.

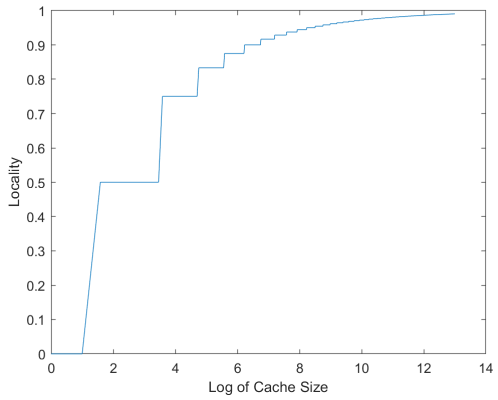

Figure 3: Locality vs Cache size in Cache-aware Algorithm

1.4.3 Cache-oblivious algorithm. As shown in figure 4, cache-oblivious algorithm achieved slightly lower locality comparing to cache-aware algorithm, due to the cost of recursion.

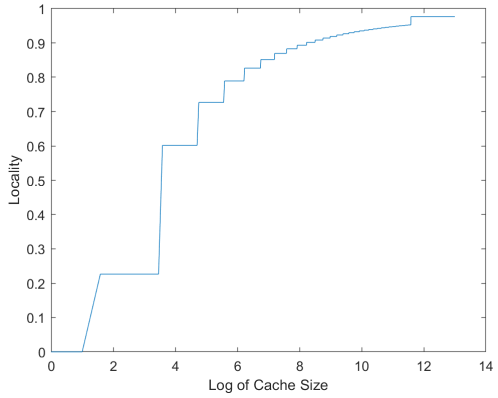

Figure 4: Locality vs Cache size in Cache-oblivious Algorithm

2 PROBLEM 2

When considering a memory hierarchy, i.e., register, L1 cache, L2 cache, we still assume that all variable accesses(matrix A, B, C loads and matrix C writes) are made from memory.

For a two-level of cache system, we have following observations: L2 cache has a bigger size than L1 but a slower latency. A miss at L1 cache results in requesting data from L2 cache. If an L2 hit happened, we would fetch data from L2. Or a miss in the L2 would lead to requesting data from the main memory and directly feeding the data into L1 cache. In other words, a miss at L1 cache would lead to requesting data from L2 where would a miss/hit. A hit at L1 cache means that we can also find this data at L2 cache. We can always treat L1 and L2 cache as an entity since L2 always contains L1.

Hence, the calculations of hit rate for the combined system of L1 and L2 cache and the hit rate for L1 cache can be calculated using the formulae in Question 1. Moreover, we have the following claims:

- We assume the cache replacement policy is inclusive.
- Total number of access is the same as in Question 1.
- The formulate of number of L1 hits w.r.t L1 cache size is the same as in Question 1.
- We can calculate a combined number of hits with cache size is the size of L2 as L2 contains L1.
- Number of accesses serviced by L2 = Number of combined accesses serviced by local storage - Number of accesses serviced by L1
- Number of L2 accesses = Total number of accesses - Number of L1 accesses serviced by L1

2.1 Triply-nested ijk loop

We have discussed four cases in Question1 with different L1 cache sizes. The addition of cache L2 won't affect the L1 locality. Here we calculate the locality for the combination of L1 and L2 cache.

- **Case I** $3 \leq Z_2 < N+2,\ Z_1 < 3$

$$Locality_{ijk} = \frac{2N^3 - N^2}{4N^3} = \frac{1}{2} - \frac{1}{4N} \tag{11}$$

- **Case II** $N+2 \leq Z_2 < N^2+N+1,\ Z_1 < N+2$

$$Locality_{ijk} = \frac{3N^3 - 2N^2}{4N^3} = \frac{3}{4} - \frac{1}{2N} \tag{12}$$

- **Case III** $N^2+N+1 \leq Z_2,\ Z_1 < N^2+N+1$

$$Locality_{ijk} = \frac{4N^3 - 3N^2}{4N^3} = 1 - \frac{3}{4N} \tag{13}$$

2.2 Cache-aware

For cache-aware and cache-oblivious, the cache misses related to L2 cache size. The block size in L2 should satisfy $b_2 = \sqrt{\frac{Z_2}{3}}$, where Z_2 should be several times larger than Z_1.

$$\begin{aligned} Locality_{block} &= \frac{\left(\frac{N}{b_2}\right)^3 (4b_2^3 - 3b_2^2) + (\frac{N}{b_2} - 1)b_2^2 \left(\frac{N}{b_2}\right)^2}{4N^3} \\ &= 1 - \frac{1}{2b_2} - \frac{1}{4N} \end{aligned} \tag{14}$$

2.3 Cache-oblivious

$$Locality_{cache\ oblivious} = \frac{4N^3 - 3N^2 - \frac{3N^3}{b_2}}{4N^3} \tag{15}$$

2.4 Some examples to demonstrate

As shown in figure 5, the only difference is that all cache sizes start from $Z = 3$, because there is no reason to make L2 cache size less than L1 cache size. By comparison, we can observe that cache-aware algorihtm is the most superior, but requires adjusting the size of blocks. Cache-oblivious does not care about cache size, but we need to pay a minor loss of locality to achieve recursions. Triply-nested ijk algorithm is too discrete on locality, which means a larger cache may not improve locality.

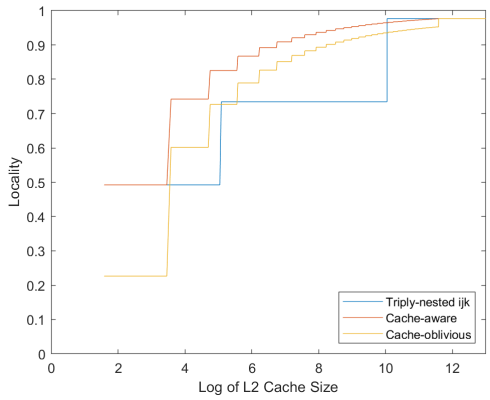

Figure 5: Locality vs L2 Cache size

3 PROBLEM 3

Here we consider extending our model to a heterogeneous-memory system which contains either non-volatile memory(NVM) or on-package memory. Because cache-oblivious algorithm has competitive performance (or locality) as cache-aware algorithm, we will combine them to one case in the following discussions.

The maximum matrix multiplication performance is given by:

$$Max\ CPU\ Performance = min\ (f,\ AI * BW) \tag{16}$$

where AI is arithmetic intensity and BW is bandwidth. For hierarchical systems, the CPU performance is limited by the minimum performance at different levels. Performance at each level is given by arithmetic intensity times bandwidth at the higher level.

$$AI * BW = min\ (AI_i * BW_{i+1}) \tag{17}$$

To obtain the highest performance for CPU, $AI_i * BW_{i+1}$ at different memory hierarchy should match the performance of CPU.

3.1 Analysis of Non-volatile memory and On-package memory

A heterogeneous-memory system with non-volatile memory and on-package memory provides extra options to enhance the performance considering the limitations of BW and cache size.

For NVM which has higher latency and lower bandwidth than DRAM, we will put it behind DRAM in hierarchy. It will be used if we don't have enough capacity of DRAM and the capacity of NVM is large enough that it can store matrices of any size in this matrix multiplication problem.

High-bandwidth on-package memory is added between on-chip cache and off-chip DRAM to relax the bandwidth restriction of the DRAM, which can boost our performance.

3.2 Triply-nested ijk Algorithm

From the analysis in Question1, we know that locality (or AI) depend on the cache size (capacity). If we don't have enough capacity for L2, the performance will be limited by the bandwidth of DRAM. Since on-package memory has higher bandwidth than DRAM, we can put it between L2 and DRAM. Also the capacity of on-package memory should be an order of magnitude higher than L2 since it is cheaper (eg, $Z_2 = O(N), Z_{on-package} = O(N^2)$). We design the following architecture for ijk algorithm:

- **CPU**

$$AI_{CPU} = \frac{2N^3}{4N^3} = \frac{1}{2} \tag{18}$$

- **L1 cache,** $Z_1 = 3$

$$AI_{L1} = \frac{2N^3}{2N^3 + 2N^2} \approx 1 \tag{19}$$

- **L2 cache,** $Z_2 = N + 2$

$$AI_{L2} = \frac{2N^3}{N^3 + 2N^2} \approx 2 \tag{20}$$

- **On-package memory,** $Z_3 = N^2 + N + 1$

$$AI_{on-package} = \frac{2N^3}{3N^2} = \frac{2N}{3} \tag{21}$$

Assume the relative bandwidth of L1 is 1, the performance of CPU will be $AI_{CPU} * BW_{L1} = 0.5$ To match this performance at every memory level, the bandwidth at every level is the following:

$$BW_{L1} = 1 \tag{22}$$

$$BW_{L2} = \frac{1}{2} \tag{23}$$

$$BW_{on-package} = \frac{1}{4} \tag{24}$$

$$BW_{DRAM} = \frac{3}{4N} \tag{25}$$

In this architecture, on-package memory should have half the bandwidth of L2 cache so it won't affect the CPU performance. We can see that the required bandwidth for DRAM is only $\frac{3}{4N}$, which is very low for a large matrix. So the inclusion of on-package memory can relax the restriction of low bandwidth of DRAM without having a very big L2 cache. Also, because the bandwidth we need is really low, we can replace the DRAM with NVM (which has bandwidth lower than DRAM), or use a combination of DRAM and NVM to enlarge our capacity for large problems.

3.3 Cache-aware/cache-oblivious Algorithm

As the analysis in previous section, we design the following architecture for Cache-aware/cache-oblivious algorithm:

- **CPU**

$$AI_{CPU} = \frac{2N^3}{4N^3} = \frac{1}{2} \tag{26}$$

- **L1 cache,** $Z_1 = 3$ and the block size in L1 $b_1 = 1$

$$AI_{L1} = \frac{2N^3}{\left(\frac{N}{b_1}\right)^3 2b_1^2} = b_1 = 1 \tag{27}$$

- **L2 cache,** $Z_2 = 3b_2^2$ and the block size in L2 is b_2

$$AI_{L2} = \frac{2N^3}{\left(\frac{N}{b_2}\right)^3 2b_2^3} = b_2 \tag{28}$$

- **On-package memory,** $Z_3 = 3b_3^2$ and the block size in on-package memory is b_3

$$AI_{on-package} = \frac{2N^3}{\left(\frac{N}{b_3}\right)^3 2b_3^3} = b_3 \tag{29}$$

- **DRAM,** $Z_4 = 3b_4^2$ and the block size in DRAM is b_4

$$AI_{DRAM} = \frac{2N^3}{\left(\frac{N}{b_4}\right)^3 2b_4^2} = b_4 \tag{30}$$

- **NVM,** $Z_5 = 3b_5^2$ and the block size in NVM is b_5

$$AI_{NVM} = \frac{2N^3}{\left(\frac{N}{b_5}\right)^3 2b_5^2} = b_5 \tag{31}$$

We analyze the performance by fixing the max CPU performance MP which is defined as $\min(AI_i * BW_{i+1})$. To satisfy this fixed performance MP, we have the requirements for both bandwidth and block size as follows:

$$L1 : BW_1 \geq MP/(\frac{1}{2}) \qquad b_1 = 1 \tag{32}$$

$$L2 : BW_2 \geq MP/(1) \qquad b_2 \geq \frac{MP}{BW_3} \tag{33}$$

$$on - package : BW_3 \geq MP/(b_2) \qquad b_3 \geq \frac{MP}{BW_4} \tag{34}$$

$$DRAM : BW_4 \geq MP/(b_3) \qquad b_4 \geq \frac{MP}{BW_5} \tag{35}$$

$$NVM : BW_5 \geq MP/(b_4) \qquad b_5 \geq \frac{MP}{BW_6} \tag{36}$$

$$\tag{37}$$

where BW_6 donate the bandwidth needed for the lower level of storage below NVM (for example, disk).

From above analysis of requirements of bandwidth and block size, we can see the needed bandwidth is getting smaller and smaller with a increasing block size such that introducing on-package memory after L2 cache would not harm the CPU performance. And the on-package memory can be used to alleviate the stress of low bandwidth of DRAM espeacilly we want a bigger L2 cache.

Although NVM has slower bandwidth as well as higher latency, we can still place it below DRAM to relax the requirement for disk bandwidth. Latency is not much a concern here as we an always use prefetch to hide latency.