ECE 587 – Hardware/Software Co-Design
Lecture 12 Memory Optimizations

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

February 18, 2026

Outline

Reading Assignment

- This lecture: Memory Optimizations
- Next lecture: 3.3, 3.4

Outline

Performance Measurements

Memory Tiling

GEMM Performance Measurements

Table: GEMM avg running time (ms), OMP_NUM_THREADS=8

N	1024	2048	4096	8192
ikj	78	662	10702	89436
ikj+OMP	12	79	1969	14444
OMP speedup	6.5	8.4	5.4	6.2

- Measured on a specific server computer.
- Makes use of available cores to speedup GEMM via OpenMP.
- The ideal speedup is the number of cores, but the actual speedup is usually less than ideal.
  - E.g. when cores need to share a limited memory bandwidth.
  - Additional cache available for individual cores may lead to superlinear speedup.
- Does the running time increase as expected as N increases?

GEMM Performance Measurements (Cont.)

Table: GEMM average GFLOPS, OMP_NUM_THREADS=8

N	1024	2048	4096	8192
ikj	26	24	12	11
ikj+OMP	167	203	65	71

- The performance can be measured as GFLOPS
  - Billion floating-point operations per second
  - For GEMM, the number of floating-point operations is $2N^3$.
- What happens when N increases from 2048 to 4096?

Understand GEMM Performance for Large N

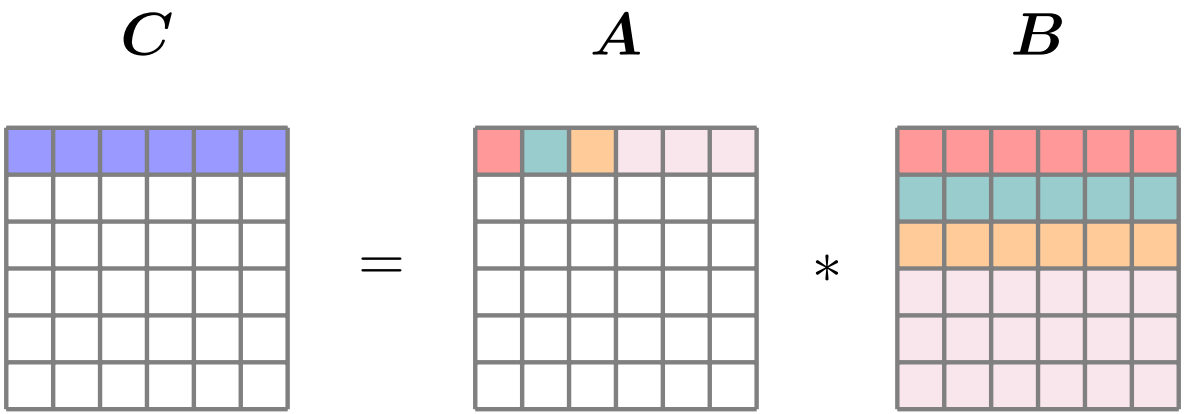
Table: GEMM average GFLOPS, OMP_NUM_THREADS=8

N	1024	2048	4096	8192
ikj	26	24	12	11
ikj+OMP	167	203	65	71

- GFLOPS decreases as we spend more time on communication per each floating-point operation.
- Matrix size: $4N^2$, 16MB for N=2048, 64MB for N=4096
- On-chip cache size is usually < 100MB
- We cannot hold the whole matrix in cache anymore and need to access off-chip main memory more often for larger N.
- Can we make better use of on-chip cache?
  - Reduce main memory access by reusing data already in cache.

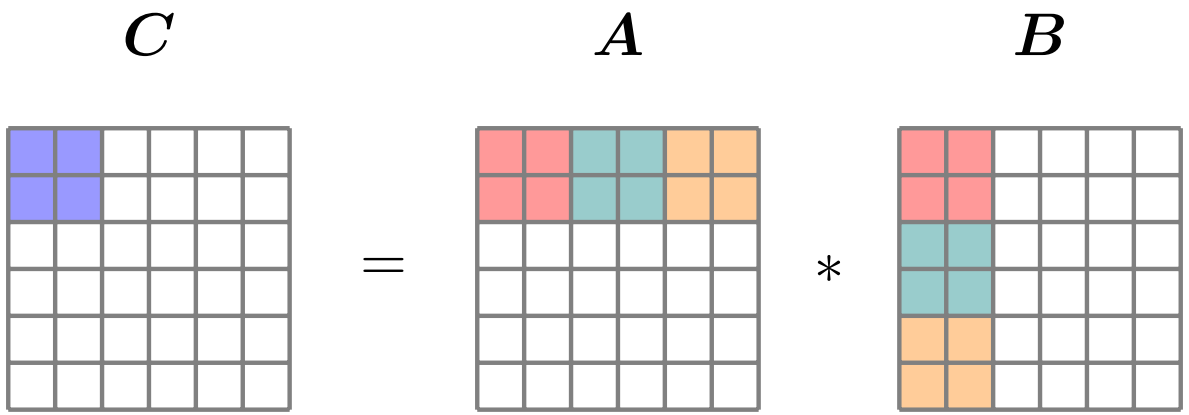
Outline

The ikj Loops



- For the ikj loops, we read A from main memory once.
- We read every row of B to calculate a row of C.
  - We compute rows of C one after another so that each row of C can stay in cache during the computation.
  - This implies that we need to read a row of B N times.
  - If B can fit in cache, we read its rows from cache and enjoy good performance.
  - When B is large, we need to read B N times from the main memory, which slows down the computation.

Block Matrix Multiplication



- A block of C can be computed by blocks from A and B.
  - $C_{1:2,1:2} = A_{1:2,1:2}B_{1:2,1:2} + A_{1:2,3:4}B_{3:4,1:2} + A_{1:2,5:6}B_{5:6,1:2}$
- What if we read a block from A and a block from B into cache to update the block of C in cache?
  - For $M \times M$ blocks, need $12M^2$ bytes in cache.
  - A block of A is needed for blocks of C on the same row.
  - A block of B is needed for blocks of C on the same column.
- Overall, we need to read both A and B N/M times from the main memory.
  - Better than reading B N times if M is reasonable.

Tiled Matrix Storage

- We can store the matrix in a tiled format to facilitate reading and writing blocks.
- Two levels of hierarchy: blocks and elements within blocks.
  - The $\frac{N^2}{M^2}$ blocks are stored in row-major order, i.e. blocks are stored row by row in the memory array.
  - The M^2 elements within a block are stored in row-major order as well.
- For a fixed block size M, all matrices can be stored in the same tiled format.

Block Matrix Multiplication Implementation

```
void gemm_tiled(const float* At, const float* Bt, float* Ct,
    std::size_t N, std::size_t M) {
    std::fill(Ct, Ct + N * N, 0.0f);

    const std::size_t tiles = N / M;
    const std::size_t M2 = M * M;
    for (std::size_t ti = 0; ti < tiles; ++ti) {
        for (std::size_t tj = 0; tj < tiles; ++tj) {
            const std::size_t baseC_tile = (ti * tiles + tj) * M2;
            for (std::size_t tk = 0; tk < tiles; ++tk) {
                const std::size_t baseA_tile = (ti * tiles + tk) * M2;
                const std::size_t baseB_tile = (tk * tiles + tj) * M2;
                ...
```

- We will need two groups of loops
  - One to go through the blocks as shown here.
  - Another one to compute a block of C using those of A and B.
- `baseA_tile`, `baseB_tile`, and `baseC_tile` are the starting indices of the block of A, B, and C respectively.
- You may notice that we are using the ijk loops here.
  - What is the difference if the ikj loops is used here?

Block Matrix Multiplication Implementation (Cont.)

```
for (std::size_t ii = 0; ii < M; ++ii) {
    const std::size_t baseCt = baseC_tile + ii * M;
    const std::size_t baseAt = baseA_tile + ii * M;
    for (std::size_t kk = 0; kk < M; ++kk) {
        const float a = At[baseAt + kk];
        const std::size_t baseBt = baseB_tile + kk * M;
        for (std::size_t jj = 0; jj < M; ++jj) {
            Ct[baseCt + jj] += a * Bt[baseBt + jj];
        }
    }
}
```

- `baseAt`, `baseBt`, and `baseCt` are the starting indices of the rows within the blocks.
- We are using the ikj loops here.
  - The memory hierarchy may consist of multiple levels of cache.
  - The ikj loops may perform better than the ijk loops here as we have analyzed for N=1024 in Lecture 04.

Parallelization

```
#pragma omp parallel for collapse(2)
for (std::size_t ti = 0; ti < tiles; ++ti) {
    for (std::size_t tj = 0; tj < tiles; ++tj) {
        ...
```

- The outer two loops can be parallelized with OpenMP.
  - `collapse(2)` allows OpenMP to parallelize the two loops together since blocks in C are computed independently.
- If we have used ikj instead of ijk loops to go through the blocks, we cannot easily parallelize the outer two loops.
  - We have to use certain synchronization mechanism to make sure that the blocks in C are updated correctly.
- Why can't we just parallelize the outermost `ti` loop?
  - We may choose a large M to reduce the need to access matrices from the main memory.
  - `tiles` (N/M) will be small and there won't be enough parallelism to utilize all cores.

GEMM Performance Updates

Table: GEMM avg running time (ms), OMP_NUM_THREADS=8, M=256

N	1024	2048	4096	8192
ikj	78	662	10702	89436
ikj+OMP	12	79	1969	14444
OMP speedup	6.5	8.4	5.4	6.2
Tiled	60	479	3852	30843
Tiled+OMP	10	80	593	4633
Tiled OMP speedup	5.8	6.0	6.5	6.7

- For M=256, each core needs $12M^2 = 784$KB cache to hold the three blocks.
- We also improve the performance for N=1024 and 2048 because there are multiple levels of cache and the tiled implementation can make better use of them.

GEMM Performance Updates (Cont.)

Table: GEMM average GFLOPS, OMP_NUM_THREADS=8, M=256

N	1024	2048	4096	8192
ikj	26	24	12	11
ikj+OMP	167	203	65	71
Tiled	33	33	33	33
Tiled+OMP	200	200	216	221

- The choice of M can have a significant impact on the performance.
- You may see different performance improvements, or degradations, for different M's on different machines on different N's

Summary

- Hardware features like on-chip and off-chip memory shape the communication architecture and thus have a significant impact on the performance of implementations.
- What other features in hardware and software system we should be aware of when designing systems and optimizing implementations?