CSC4005 Project 2 Report

How to compile the programs

```
cd /path/to/project2

mkdir build && cd build

cmake ..

make -j4
```

How to execute the programs

```
cd /path/to/project2/build

sbatch ./../src/sbatch.sh
```

How does each parallel programming model do computation in parallel

SIMD (Single Instruction, Multiple Data):

It performs a single operation on multiple data points simultaneously.

OpenMP (Open Multi-Processing):

Uses compiler directives to automatically parallelize code. It allows shared-memory parallelism, where threads can be forked to perform tasks concurrently.

MPI (Message Passing Interface):

It uses a message-passing paradigm where individual processes communicate with each other by explicitly sending and receiving messages.

What kinds of optimizations have you tried to speed up your parallel program, and how does them work?

Memory Locality:

Loop Ordering (Row-wise traversal for `matrix1` and Row-wise traversal for `matrix2`):

- The order of the loops is crucial for optimizing memory accesses. The outermost loop iterates over the rows of `matrix1`, the middle loop iterates over the columns of `matrix1` (which correspond to the rows of `matrix2`), and the innermost loop iterates over the columns of `matrix2`. The given algorithm accesses the elements of `matrix1` in a row-wise manner and the elements of `matrix2` in a row-wise manner. This approach is optimized for row-major storage, where consecutive elements of the same row are stored adjacently in memory. By accessing the elements of `matrix1` and `matrix2` row-wise, the algorithm maximizes spatial locality because consecutive memory accesses are likely to hit the same cache line, reducing the need to fetch data from main memory frequently. This organization increases cache reuse for rows of `matrix1` and `matrix2`.

Prefetching rows and columns:

- Variables like `result_i`, `matrix1_i`, and `matrix2_k` are used to prefetch a whole row pointer. This reduces the overhead of calculating the memory address for every single element in the innermost loop. The inner loops can then use these pointers to access elements in a cache-friendly manner. In addition, the compiler might proactively prefetch the memory regions pointed to by the pointers, such as `result_i`, `matrix1_i`, and `matrix2_k`, into the cache. This anticipatory memory loading, based on the predictability of access patterns, can significantly reduce cache miss penalties and further boost the performance of the matrix multiplication.

Data Level Parallelism:

SIMD Vectorization:

- SIMD instructions allow for the simultaneous processing of multiple data elements. In the context of this matrix multiplication, each SIMD instruction operates on eight `int` elements at once, as suggested by the use of `__m256i`, which is a 256-bit wide vector type from Intel's AVX2 instruction set.

Loading and Storing with SIMD:

- The `_mm256_load_si256` function is used to load eight consecutive `int` values from `matrix2_k` into a SIMD vector. Similarly, the `_mm256_store_si256` function stores the computed results from the SIMD vectors back into the `result` matrix. These operations ensure efficient data transfer between memory and the SIMD registers.

Parallel Arithmetic Operations:

- The `_mm256_set1_epi32` function is utilized to broadcast a single `int` value (in this case, `matrix1_ik`) across all eight lanes of a SIMD vector. This facilitates the element-wise multiplication with another vector from `matrix2`. Operations such as `_mm256_mullo_epi32` and `_mm256_add_epi32` perform element-wise multiplication and addition on two SIMD vectors, respectively. These operations are parallelized, meaning that eight multiplications or additions are performed in a single instruction.

- `__m256i prod_vec = _mm256_mullo_epi32(m1_vec, m2_vec);` This instruction multiplies eight pairs of 32-bit integers concurrently. Essentially, for each element in the vector `m1_vec` (eight same element in `m1_vec`), it is multiplied with the corresponding element in the vector `m2_vec`. For example, `m1_vec` is [a11, a11, a11, a11, a11, a11, a11, a11], `m2_vec` is [b11, b12, b13, b14, b15, b16, b17, b18] and `prod_vec` is [a11 * b11, a11 * b12, a11 * b13, a11 * b14, a11 * b15, a11 * b16, a11 * b17, a11 * b18]. Here, each value of `a` in `m1_vec` refers to an element from `matrix1`, and each value of `b` in `m2_vec` refers to an element from `matrix2`.
- `result_vecs[j / 8] = _mm256_add_epi32(result_vecs[j / 8], prod_vec);` Once the products are computed, they need to be accumulated into the result matrix. This instruction takes the previously stored partial sum (from `result_vecs[j / 8]`) and adds the newly computed products (from `prod_vec`) element-wise, again operating on eight integers at once. For example, [c11, c12, c13, c14, c15, c16, c17, c18] = [a11 * b11, a11 * b12, a11 * b13, a11 * b14, a11 * b15, a11 * b16, a11 * b17, a11 * b18] + [a12 * b21, a12 * b22, a12 * b23, a12 * b24, a12 * b25, a12 * b26, a12 * b27, a12 * b28] + ... + [a1n * bn1, a1n * bn2, a1n * bn3, a1n * bn4, a1n * bn5, a1n * bn6, a1n * bn7, a1n * bn8], where n = 1024 in the third test case.

Thread Level Parallelism:

Parallelization using OpenMP:

- The most evident OpenMP optimization here is the use of the `#pragma omp parallel for` directive. This directive instructs the compiler to create multiple threads to concurrently execute iterations of the subsequent for-loop. By default, OpenMP creates as many threads as there are cores available, ensuring that the CPU's processing power is used more effectively.
- The matrix's rows are effectively divided among available threads, leading to a data decomposition parallelism model where each thread works on its subset of the data. Each thread calculates the result for one or more full rows of `matrix1`. Since each thread works on contiguous memory segments (rows), cache efficiency is enhanced as threads are less likely to interfere with each other's cache lines.

Process Level Parallelism:

Domain Decomposition:

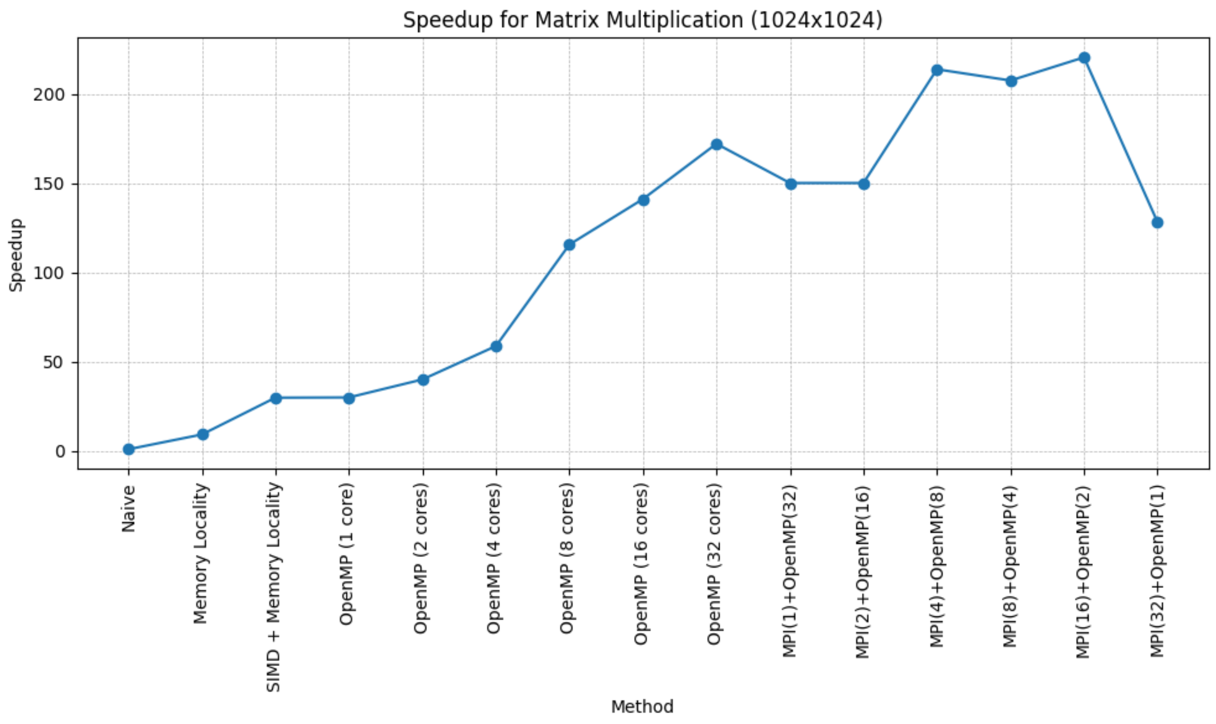
- The matrices are decomposed by rows across different MPI processes. This means that each process is responsible for computing a subset of the rows of the resulting matrix. The row assignment logic (`rows` array) ensures that if the number of rows is not divisible evenly among the processes, some processes will just compute an additional row. This is a simple yet effective load balancing strategy.

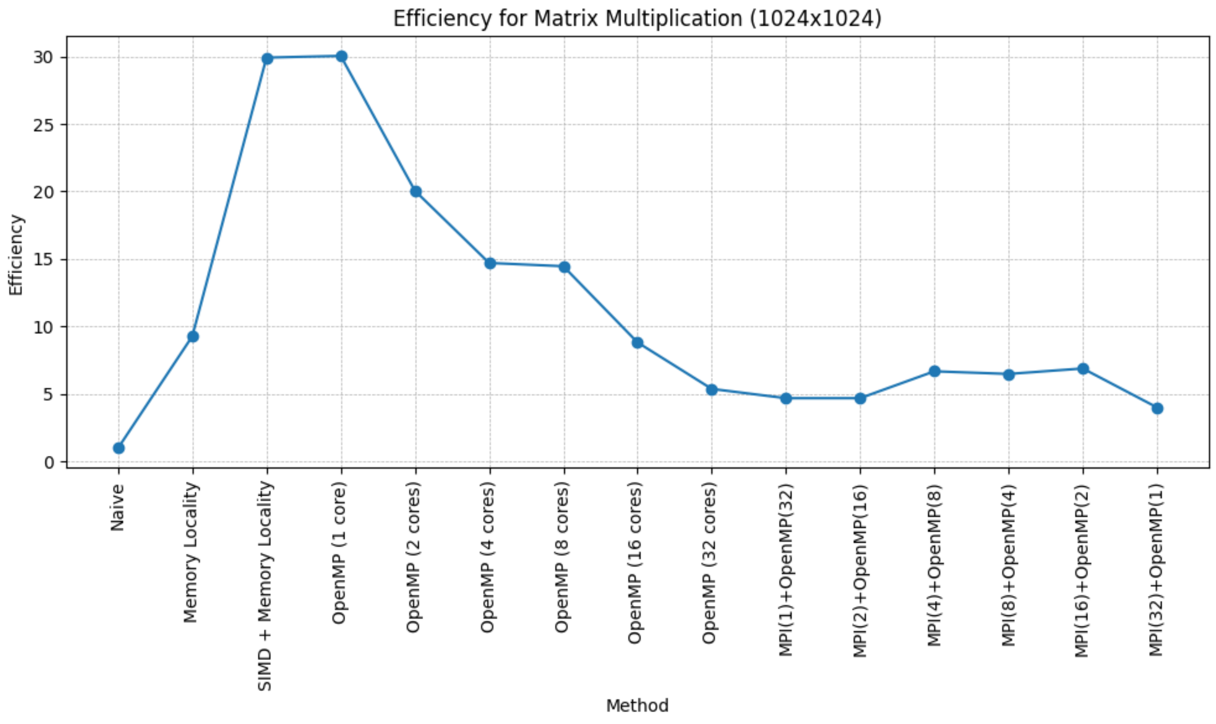
Localized Fragment Storage:

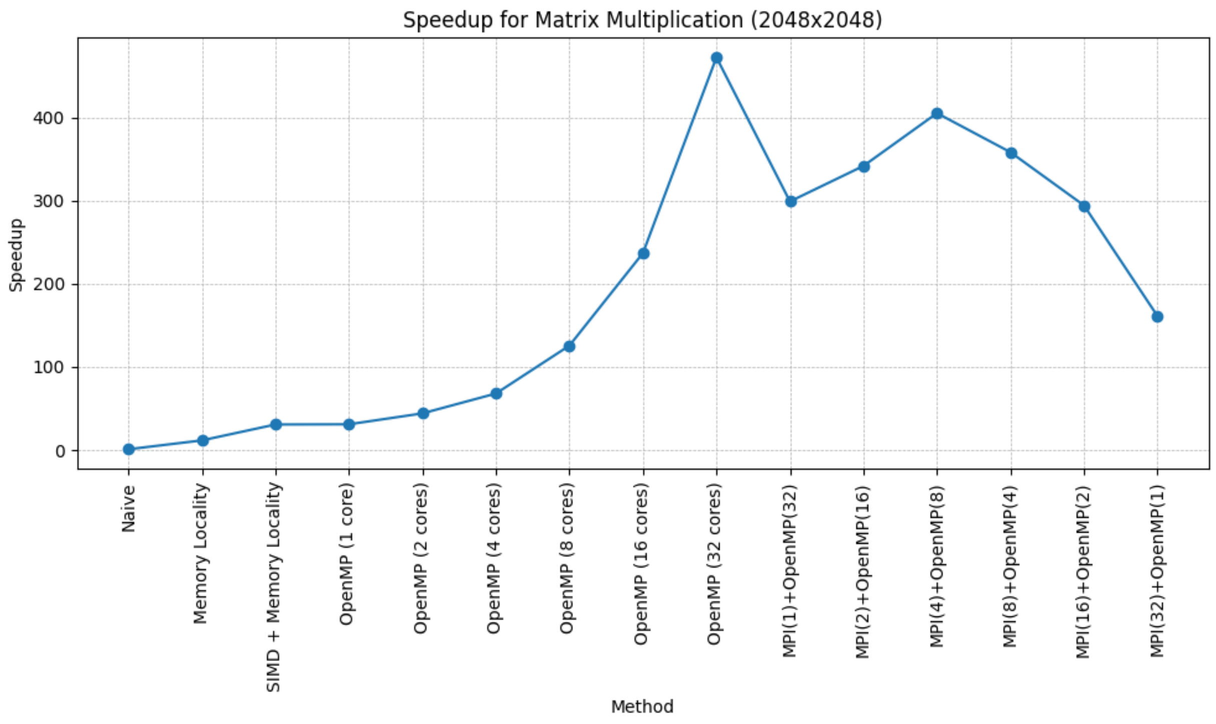
- For each process, memory is allocated only for the portion of the result that it is responsible for computing. This is a significant memory-saving measure when compared to allocating memory for the entire matrix on every process. Each process is solely concerned with the rows of the result it's responsible for, leading to efficient memory utilization.

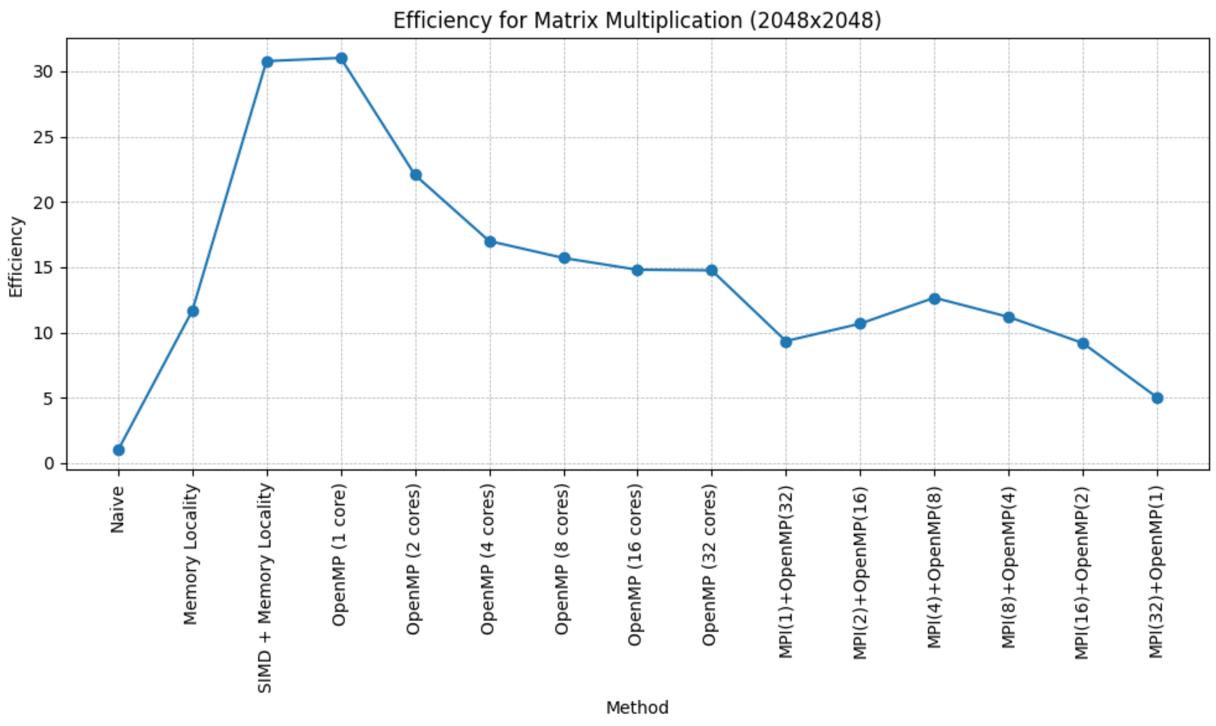
Performance

Methods	Matrices 1024x1024	Matrices 2048x2048
Naive (Optimized with -O2)	7059 ms	74141 ms
Memory Locality (Optimized with -O2)	761 ms	6355 ms
SIMD + Memory Locality (Optimized with -O2)	236 ms	2409 ms
OpenMP + SIMD + Memory Locality (1 core)	235 ms	2390 ms
OpenMP + SIMD + Memory Locality (2 cores)	176 ms	1680 ms
OpenMP + SIMD + Memory Locality (4 cores)	120 ms	1090 ms
OpenMP + SIMD + Memory Locality (8 cores)	61 ms	590 ms
OpenMP + SIMD + Memory Locality (16 cores)	50 ms	313 ms
OpenMP + SIMD + Memory Locality (32 cores)	41 ms	157 ms
MPI(1) + OpenMP(32) + SIMD + Memory Locality	47 ms	248 ms
MPI(2) + OpenMP(16) + SIMD + Memory Locality	47 ms	217 ms
MPI(4) + OpenMP(8) + SIMD + Memory Locality	33 ms	183 ms
MPI(8) + OpenMP(4) + SIMD + Memory Locality	34 ms	207 ms
MPI(16) + OpenMP(2) + SIMD + Memory Locality	32 ms	252 ms
MPI(32) + OpenMP(1) + SIMD + Memory Locality	55 ms	460 ms


Speedup for Matrix Multiplication (1024x1024)
Speedup
0
50
100
150
200
Naive
Memory Locality
SIMD + Memory Locality
OpenMP (1 core)
OpenMP (2 cores)
OpenMP (4 cores)
OpenMP (8 cores)
OpenMP (16 cores)
OpenMP (32 cores)
MPI(1)+OpenMP(32)
MPI(2)+OpenMP(16)
MPI(4)+OpenMP(8)
MPI(8)+OpenMP(4)
MPI(16)+OpenMP(2)
MPI(32)+OpenMP(1)
Method


Efficiency for Matrix Multiplication (1024x1024)
Efficiency
0
5
10
15
20
25
30
Naive
Memory Locality
SIMD + Memory Locality
OpenMP (1 core)
OpenMP (2 cores)
OpenMP (4 cores)
OpenMP (8 cores)
OpenMP (16 cores)
OpenMP (32 cores)
MPI(1)+OpenMP(32)
MPI(2)+OpenMP(16)
MPI(4)+OpenMP(8)
MPI(8)+OpenMP(4)
MPI(16)+OpenMP(2)
MPI(32)+OpenMP(1)
Method





What have you found from the experiment results?

1. **SIMD and Memory Locality**: Adding SIMD (Single Instruction, Multiple Data) to the memory locality optimization provides a further reduction in execution time. SIMD allows operations to be performed on multiple data elements simultaneously, thus speeding up the computation.

2. **OpenMP Scaling**: As cores are added using OpenMP, there is a consistent reduction in execution time up to a point. The time decreases from using a single core to using 32 cores, but the reduction is not strictly linear. There are diminishing returns as more cores are added, which is a classic sign of Amdahl's law in parallel computing. The overhead of synchronization, communication between threads, and contention can impact the speedup.
3. **MPI and OpenMP**: Combining MPI (Message Passing Interface) with OpenMP shows how distributed memory (across MPI processes) and shared memory (within OpenMP threads) parallelism can be mixed. The best time for 1024x1024 matrices is achieved with MPI(4) + OpenMP(8), whereas for 2048x2048 matrices, the combination MPI(4) + OpenMP(8) and MPI(16) + OpenMP(2) give close optimal times. It's interesting to note that merely increasing MPI processes and reducing OpenMP threads (or vice-versa) does not guarantee the best performance. There's a balance to be struck.
4. **Problem Size Matters**: The benefits of parallelism are more pronounced for larger problems. For the 2048x2048 matrix size, the speedup from using parallel techniques is more evident than the 1024x1024 matrix size. This is a common trend in parallel computing: larger problems tend to benefit more from parallel execution because the overhead of parallelization becomes a smaller fraction of the total computation time.
5. **Overhead of Too Much Parallelism**: The execution time for MPI(32) + OpenMP(1) is higher than some configurations with fewer cores or processes. This indicates that there's overhead in managing too many MPI processes, which might outweigh the benefits of parallel execution for this specific problem size and configuration.

Profiling Results & Analysis with `perf`

Naive

```
Naive Matrix Multiplication 1024*1024 (Optimized with -O2)
Output file to: /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_naive.txt
Multiplication Complete!
Execution Time: 7055 milliseconds

 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/naive /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix5.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix6.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_naive.txt':

    20,893,112,073      cpu-cycles:u
           168,765      cache-misses:u
             8,388      page-faults:u

       7.539049580 seconds time elapsed

       7.244484000 seconds user
       0.029935000 seconds sys



Naive Matrix Multiplication 2048*2048 (Optimized with -O2)
Output file to: /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_naive.txt
Multiplication Complete!
Execution Time: 65800 milliseconds

 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/naive /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix7.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix8.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_naive.txt':

   191,808,053,017      cpu-cycles:u
       121,725,276      cache-misses:u
            43,633      page-faults:u

      66.985174537 seconds time elapsed

      66.349639000 seconds user
       0.095769000 seconds sys
```

Memory Locality

```
Memory Locality Matrix Multiplication 1024*1024 (Optimized with -O2)
Output file to: /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_locality.txt
Multiplication Complete!
Execution Time: 761 milliseconds

 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/locality /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix5.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix6.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_locality.txt':

     2,740,727,455      cpu-cycles:u
           170,921      cache-misses:u
             3,523      page-faults:u

       1.068078821 seconds time elapsed

       0.948745000 seconds user
       0.023917000 seconds sys
```

```
Memory Locality Matrix Multiplication 2048*2048 (Optimized with -O2)
Output file to: /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_locality.txt
Multiplication Complete!
Execution Time: 6371 milliseconds

 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/locality /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix7.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix8.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_locality.txt':

    20,470,969,120      cpu-cycles:u
       129,968,001      cache-misses:u
            31,031      page-faults:u

       7.565526881 seconds time elapsed

       7.084774000 seconds user
       0.114818000 seconds sys
```

SIMD + Memory Locality

```
SIMD + Memory Locality Matrix Multiplication 1024*1024 (Optimized with -O2)
Output file to: /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_simd.txt
Multiplication Complete!
Execution Time: 236 milliseconds

 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/simd /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix5.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix6.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_simd.txt':

     1,130,691,965      cpu-cycles:u
           155,287      cache-misses:u
             3,523      page-faults:u

       0.539193091 seconds time elapsed

       0.428839000 seconds user
       0.017993000 seconds sys



SIMD + Memory Locality Matrix Multiplication 2048*2048 (Optimized with -O2)
Output file to: /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_simd.txt
Multiplication Complete!
```

```
Execution Time: 2416 milliseconds

 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/simd /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix7.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix8.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_simd.txt':

     8,117,666,455      cpu-cycles:u
       129,542,222      cache-misses:u
            24,093      page-faults:u

       3.602833149 seconds time elapsed

       3.156346000 seconds user
       0.088812000 seconds sys
```

OpenMP + SIMD + Memory Locality (Extracted)

```
OpenMP + SIMD + Memory Locality Matrix Multiplication 1024*1024 (Optimized with -O2)
Number of cores: 8
Output file to: /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_openmp_8.txt
Multiplication Complete!
Execution Time: 59 milliseconds

 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/openmp 8 /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix5.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix6.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_openmp_8.txt':

     1,566,186,109      cpu-cycles:u
           378,737      cache-misses:u
             3,598      page-faults:u

       0.400696385 seconds time elapsed

       0.698240000 seconds user
       0.028927000 seconds sys



Number of cores: 16
Output file to: /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_openmp_16.txt
Multiplication Complete!
Execution Time: 40 milliseconds
```

```
 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/openmp 16 /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix5.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix6.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_openmp_16.txt':

     1,806,478,309      cpu-cycles:u
           422,464      cache-misses:u
             3,624      page-faults:u

       0.375367338 seconds time elapsed

       0.883176000 seconds user
       0.023188000 seconds sys



Number of cores: 32
Output file to: /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_openmp_32.txt
Multiplication Complete!
Execution Time: 38 milliseconds

 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/openmp 32 /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix5.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix6.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_openmp_32.txt':

     2,660,138,918      cpu-cycles:u
           375,650      cache-misses:u
             3,676      page-faults:u

       0.475329131 seconds time elapsed

       1.348020000 seconds user
       0.055343000 seconds sys



OpenMP + SIMD + Memory Locality Matrix Multiplication 2048*2048 (Optimized with -O2)
Number of cores: 8
Output file to: /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_openmp_8.txt
Multiplication Complete!
Execution Time: 504 milliseconds

 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/openmp 8 /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix7.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix8.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_openmp_8.txt':
    11,298,040,988      cpu-cycles:u
        70,322,089      cache-misses:u
            12,923      page-faults:u

       1.723913583 seconds time elapsed

       4.489249000 seconds user
       0.087809000 seconds sys



Number of cores: 16
Output file to: /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_openmp_16.txt
Multiplication Complete!
Execution Time: 287 milliseconds

 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/openmp 16 /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix7.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix8.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_openmp_16.txt':

    12,116,049,526      cpu-cycles:u
        54,499,531      cache-misses:u
            12,957      page-faults:u

       1.513055145 seconds time elapsed

       4.960884000 seconds user
       0.089925000 seconds sys



Number of cores: 32
Output file to: /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_openmp_32.txt
Multiplication Complete!
Execution Time: 187 milliseconds

 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/openmp 32 /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix7.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix8.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_openmp_32.txt':

    11,182,466,326      cpu-cycles:u
        34,209,405      cache-misses:u
            13,025      page-faults:u

       1.422266578 seconds time elapsed
       4.883062000 seconds user
       0.120557000 seconds sys
```

MPI + OpenMP + SIMD + Memory Locality (Extracted)

```
MPI + OpenMP + SIMD + Memory Locality Matrix Multiplication 1024*1024 (Optimized
with -O2)
Number of Processes: 2, Number of Threads: 16
Output file to: /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_mpi_2.txt
Multiplication Complete!
Execution Time: 52 milliseconds

 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/mpi 16 /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix5.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix6.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_mpi_2.txt':

     1,902,768,438      cpu-cycles:u
           207,487      cache-misses:u
             4,571      page-faults:u

       0.599289252 seconds time elapsed

       0.867701000 seconds user
       0.036733000 seconds sys



 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/mpi 16 /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix5.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix6.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_mpi_2.txt':

     1,413,110,620      cpu-cycles:u
           247,240      cache-misses:u
             5,711      page-faults:u

       0.425505079 seconds time elapsed

       0.742175000 seconds user
       0.053651000 seconds sys



Number of Processes: 4, Number of Threads: 8
```

```
Output file to: /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_mpi_4.txt
Multiplication Complete!
Execution Time: 47 milliseconds

 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/mpi 8 /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix5.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix6.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_mpi_4.txt':

     1,231,652,963      cpu-cycles:u
           209,791      cache-misses:u
             4,293      page-faults:u

       0.852781137 seconds time elapsed

       0.550674000 seconds user
       0.032980000 seconds sys



 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/mpi 8 /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix5.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix6.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_mpi_4.txt':

     1,237,858,604      cpu-cycles:u
           193,752      cache-misses:u
             4,294      page-faults:u

       0.406346639 seconds time elapsed

       0.558972000 seconds user
       0.025859000 seconds sys



 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/mpi 8 /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix5.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix6.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_mpi_4.txt':

     1,318,926,801      cpu-cycles:u
           555,241      cache-misses:u
             3,779      page-faults:u

       0.799618686 seconds time elapsed

       0.557578000 seconds user
       0.026027000 seconds sys



 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/mpi 8 /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix5.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix6.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_mpi_4.txt':

       973,197,716      cpu-cycles:u
           325,524      cache-misses:u
             6,180      page-faults:u

       0.418305016 seconds time elapsed

       0.466437000 seconds user
       0.038786000 seconds sys

MPI + OpenMP + SIMD + Memory Locality Matrix Multiplication 2048*2048 (Optimized
with -O2)
Number of Processes: 2, Number of Threads: 16
Output file to: /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_mpi_2.txt
Multiplication Complete!
Execution Time: 221 milliseconds

 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/mpi 16 /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix7.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix8.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_mpi_2.txt':

     8,897,452,242      cpu-cycles:u
        28,612,976      cache-misses:u
            12,416      page-faults:u

       1.850219626 seconds time elapsed

       3.616515000 seconds user
       0.101901000 seconds sys



 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/mpi 16 /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix7.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix8.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_mpi_2.txt':

     7,176,297,461      cpu-cycles:u
        15,226,559      cache-misses:u
```

```
            16,662      page-faults:u

       1.674257723 seconds time elapsed

       2.978515000 seconds user
       0.109982000 seconds sys



Number of Processes: 4, Number of Threads: 8
Output file to: /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_mpi_4.txt
Multiplication Complete!
Execution Time: 261 milliseconds

 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/mpi 8 /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix7.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix8.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_mpi_4.txt':

     6,179,767,150      cpu-cycles:u
        23,061,086      cache-misses:u
            11,353      page-faults:u

       1.969581281 seconds time elapsed


       2.413662000 seconds user
       0.071930000 seconds sys


 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/mpi 8 /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix7.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix8.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_mpi_4.txt':

     5,979,462,972      cpu-cycles:u
        57,633,822      cache-misses:u
            11,344      page-faults:u

       1.913728396 seconds time elapsed

       2.319228000 seconds user
       0.085897000 seconds sys
 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/mpi 8 /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix7.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix8.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_mpi_4.txt':

     6,225,134,153      cpu-cycles:u
        25,348,347      cache-misses:u
            11,354      page-faults:u

       1.498649092 seconds time elapsed

       2.421724000 seconds user
       0.090727000 seconds sys



 Performance counter stats for '/nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/src/mpi 8 /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix7.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../matrices/matrix8.txt /nfsmnt/120090638/CSC4005-
2023Fall/project2/build/../build/result_mpi_4.txt':

     5,227,020,518      cpu-cycles:u
        38,790,384      cache-misses:u
            17,245      page-faults:u

       1.511520547 seconds time elapsed

       2.088433000 seconds user
       0.099638000 seconds sys
```

Memory Locality

1. **Page Faults**: There's a significant reduction in page faults when optimizing for memory locality. This indicates that the optimization is accessing memory in a way that results in fewer page swaps between RAM and secondary storage (like a hard drive or SSD), which can be a costly operation in terms of time.
2. **Cache Misses**: Interestingly, for the 1024x1024 matrix, cache misses increased slightly with memory locality optimization. However, this did not adversely affect the execution time. For the 2048x2048 matrix, cache misses also increased with memory locality optimization. This suggests that while the memory locality optimization improved the access pattern for matrix elements, it didn't necessarily reduce cache misses for this computation. It's also possible that while there were more cache misses, they might have been less costly due to improved memory access patterns.

3. **User vs. System Time**: The 'user' time, which represents the time the CPU spends on executing user-level applications, is dominant in both naive and optimized cases. The 'sys' time, which denotes the time the CPU spends on executing kernel or system-level operations, remains minimal. This indicates that most of the computation time is spent on the matrix multiplication operation itself, rather than on system overheads.

SIMD + Memory Locality

1. **Cache Misses**: With the introduction of SIMD, cache misses decreased slightly for both matrix sizes. This suggests that SIMD operations might be efficiently utilizing the CPU cache, thereby reducing the number of times the CPU has to fetch data from main memory.
2. **Page Faults**: The page faults further decreased with SIMD optimization for the 2048x2048 matrix but remained the same for the 1024x1024 matrix. This further reduction for the larger matrix size suggests better memory management with the combination of SIMD and memory locality optimizations.
3. **User vs. System Time**: As with the previous results, the 'user' time dominates the computation, indicating that the matrix multiplication operation itself takes up most of the computation time. The system time remains relatively minimal in both scenarios.

OpenMP + SIMD + Memory Locality

- **Cache Misses**: With the integration of OpenMP and increasing cores, cache misses increase for the 1024x1024 matrix (from 8 cores to 16 cores) but significantly decrease for the 2048x2048 matrix. This suggests that for larger matrices, the parallelism with more cores leads to more efficient cache utilization.
- **Page Faults**: For both matrices, the OpenMP method, irrespective of the number of cores, shows a slight increase in page faults. This might hint at the additional overhead of managing multiple threads concurrently.
- **User vs. System Time**: As with the previous results, the 'user' time dominates the computation, indicating that the matrix multiplication operation itself takes up most of the computation time. The system time remains relatively minimal in both scenarios.
- For OpenMP + SIMD + Memory Locality of matrix size 1024x1024, it appears that as the number of cores increases, the cache misses remain relatively stable, and there's not a significant reduction in cache misses or page faults. However, the execution time decreases with more cores, indicating improved parallelism. Here, for the larger matrix size of 2048x2048, we observe a significant reduction in cache misses as the number of cores increases. This suggests that the optimizations in the code are more effective for larger matrices and parallelism helps reduce cache misses.

MPI + OpenMP + SIMD + Memory Locality

1. **Matrix Size 1024x1024 with 2 Processes and 16 Threads:**
   - Cache Misses: Approximately 207,487
   - Page Faults: Approximately 4,571
   - Execution Time: 52 milliseconds

   This configuration demonstrates good performance with relatively low cache misses and page faults. The use of multiple threads per process and MPI parallelism contributes to the overall efficiency.
2. **Matrix Size 1024x1024 with 4 Processes and 8 Threads:**
   - Cache Misses: Approximately 209,791
   - Page Faults: Approximately 4,293
   - Execution Time: 47 milliseconds

   This configuration also shows low cache misses and page faults. Distributing the computation across more processes while still utilizing multiple threads within each process seems to be effective.
3. **Matrix Size 2048x2048 with 2 Processes and 16 Threads:**
   - Cache Misses: Approximately 28,612,976
   - Page Faults: Approximately 12,416
   - Execution Time: 221 milliseconds

   The larger matrix size results in significantly higher cache misses. However, the execution time is reasonable, indicating that the optimizations are still effective.
4. **Matrix Size 2048x2048 with 4 Processes and 8 Threads:**
   - Cache Misses: Approximately 23,061,086
   - Page Faults: Approximately 11,353
   - Execution Time: 261 milliseconds

   Similar to the previous case, the larger matrix size leads to more cache misses. However, the execution time remains reasonable.

Overall, the optimizations in the code appear to be effective, resulting in low cache misses and page faults even for larger matrix sizes. The combination of MPI for distributed computing and OpenMP for intra-process parallelism, along with SIMD and memory locality optimizations, seems to be well-suited for this matrix multiplication task.