**Title: Performance Comparison of Tiled vs. Non-Tiled Matrix Multiplication**

**1. Aim**

The objective of this assignment is to compare the performance of tiled and non-tiled approaches in matrix multiplication. Specifically, we analyse how different tiling sizes (e.g., row-wise, column-wise, row16, row32, row64, etc.) impact computation time while multiplying a 1024 x 1024 matrix.

**2. System Configuration**

* **Processor:** Intel Core i5-12450H
* **Operating System:** Ubuntu 24.02

**3. Definition**

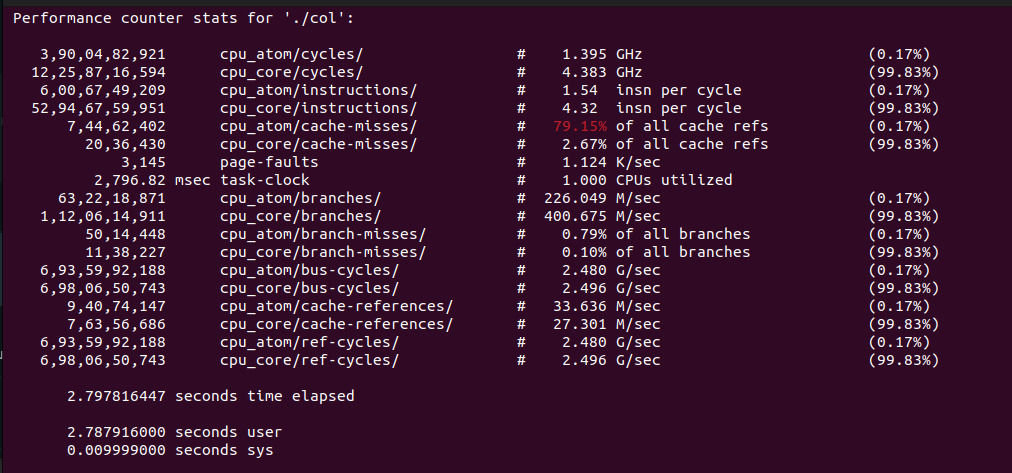
**Non-Tiled Approach**

In the non-tiled approach, the entire matrix is processed sequentially without breaking it into smaller blocks. This method is simple but often inefficient due to memory access latency and lack of cache optimization.

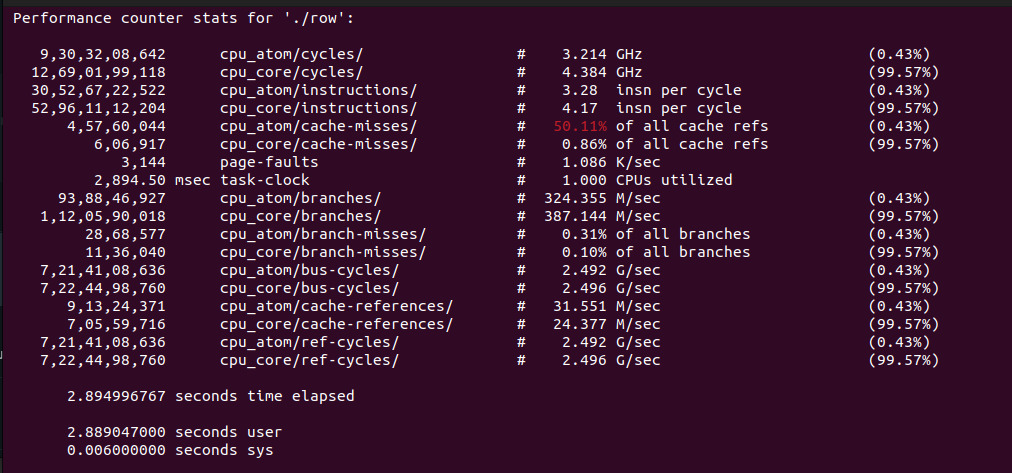
**Tiled Approach**

Tiled matrix multiplication divides the matrices into smaller sub-matrices (tiles). These tiles are processed independently, reducing cache misses and improving memory access efficiency.

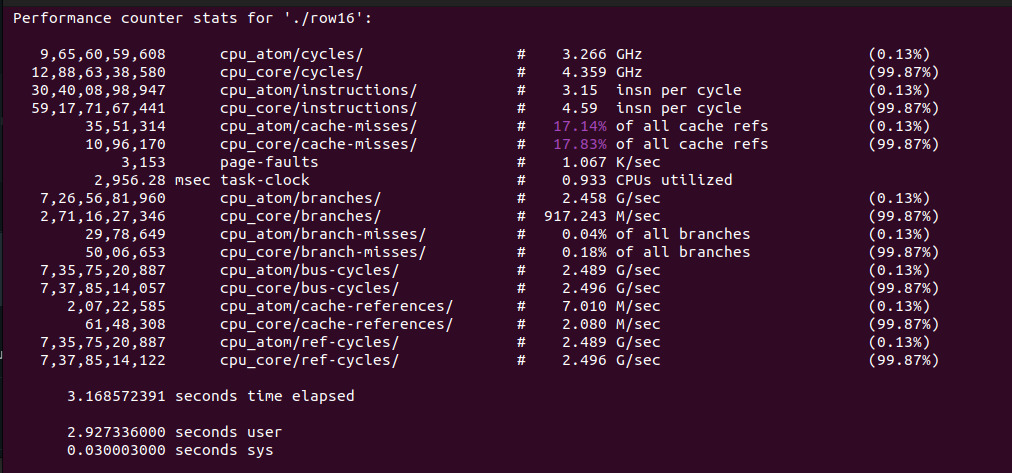
**4. Performance Comparison Table**

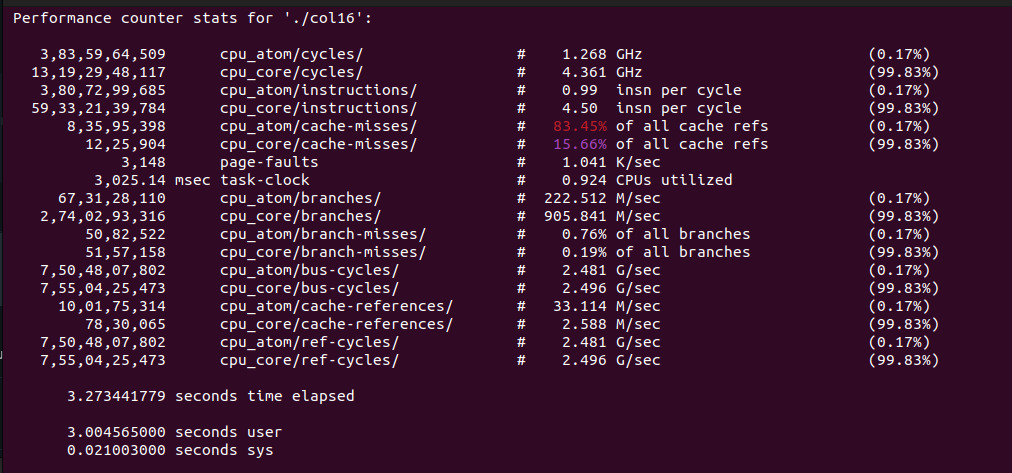
**Non-tiled Column method**

**Non tiled Row method**

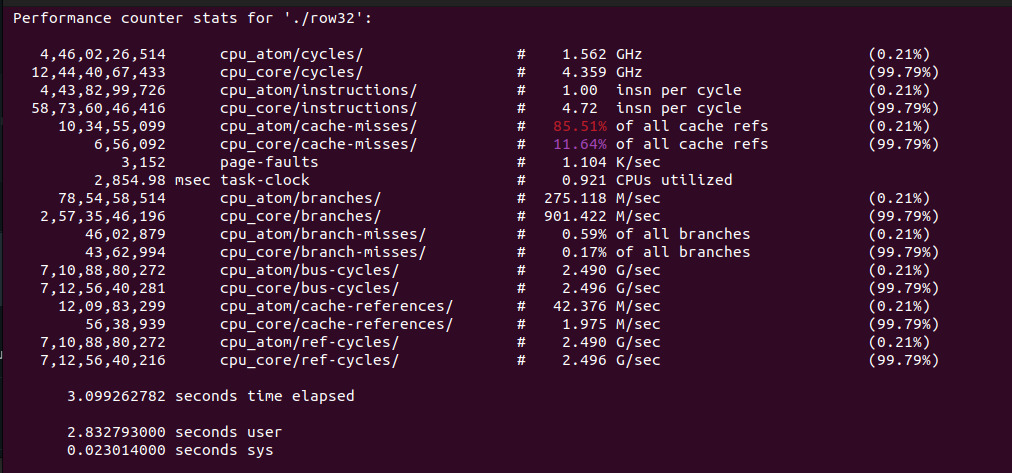
****

**Tiled Approach (Row Method 16x16)**

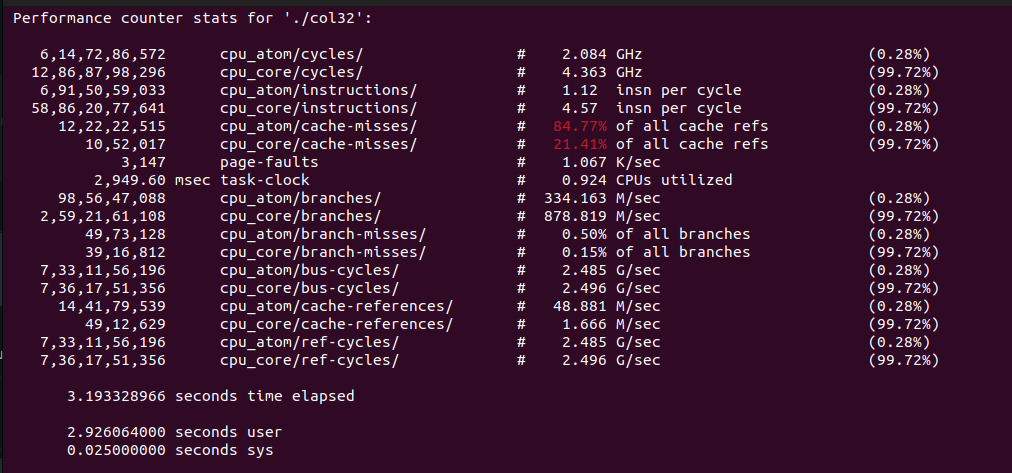
****

**Tiled Approach (Column Method 16x16)**

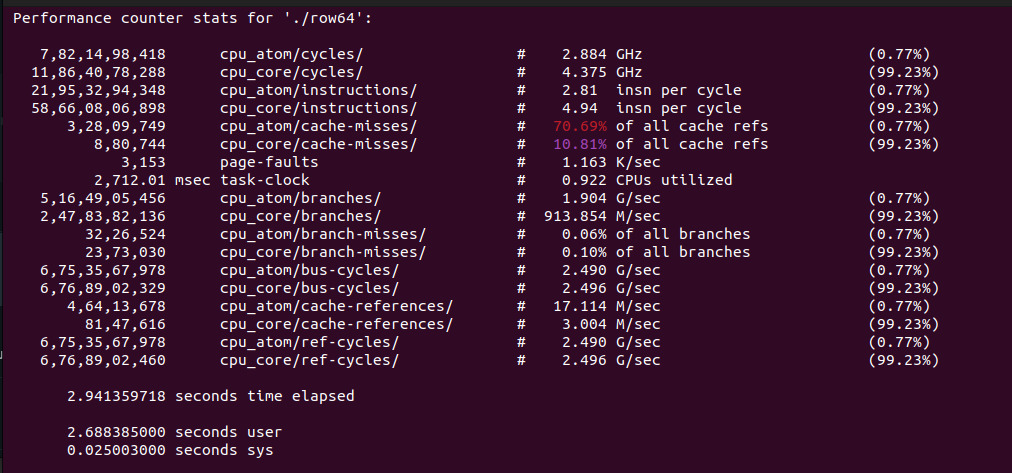
**Tiled Approach (Row method 32 x 32)**

****

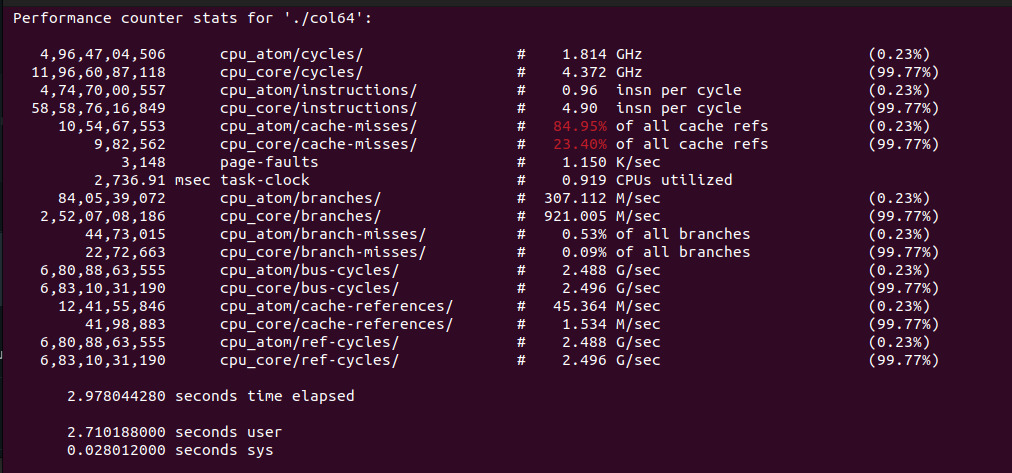
**Tiled Approach (Row method 32 x 32)**

****

**Tiled Approach (Row method 64 x 64)**

****

**Tiled Approach (Column method 64 x 64)**

****

**5. Advantages & Disadvantages**

**Non-Tiled Approach**

**Advantages:**

* Simpler to implement
* No additional overhead for managing tiles

**Disadvantages:**

* Poor cache utilization
* Higher memory access latency
* Slower execution for large matrices

**Tiled Approach**

**Advantages:**

* Better cache performance
* Reduced memory access overhead
* Faster execution times for large matrices

**Disadvantages:**

* Requires careful tile size selection
* Additional implementation complexity

**Comparison Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **CPU GHz** | **IPC** | **Cache Miss %** | **Execution Time (s)** |
| **col** | **1.395** | **4.32** | **79.15%** | **2.797** |
| **row** | **3.214** | **4.17** | **50.11%** | **2.899** |
| **row16** | **3.266** | **4.59** | **17.14%** | **3.168** |
| **col16** | **1.814** | **4.90** | **84.95%** | **2.978** |
| **row32** | **1.562** | **4.72** | **85.51%** | **3.099** |
| **col32** | **1.814** | **4.90** | **84.95%** | **2.978** |
| **row64** | **2.884** | **4.94** | **70.69%** | **2.941** |
| **col64** | **1.814** | **4.90** | **84.95%** | **2.978** |

From this analysis we can consider row64 has the best IPC which makes it most suitable but at the same time row16 is favourable for smaller cache memory as it has the least cache miss.

Column wise methods are slower and inefficient on intel core processors.

**6. Conclusion**

From the experiment, the tiled approach significantly improves performance over the non-tiled method due to efficient memory access. However, the optimal tile size varies based on system architecture, and excessively large or small tile sizes may degrade performance.

**7. Case Study: Intel Core i5 vs. AMD Ryzen for Matrix Computation**

**Intel i5-12450H**

* Strong single-thread performance
* Moderate cache size
* Decent power efficiency
* Row method is favoured.

**AMD Ryzen (e.g., Ryzen 5 5600H)**

* Better multi-thread performance
* Higher L3 cache size, which benefits tiled approaches
* Power efficiency varies based on workload
* Column method is favoured.

**Conclusion:**

For general matrix multiplication, AMD Ryzen processors tend to perform better in tiled approaches due to larger cache sizes and superior multi-threading capabilities. However, Intel i5 CPUs perform competitively in workloads favouring single-thread performance.