Parallel Programming and System Administration Report – Maciej Orzol

1.1

Function	Computation Time (s) (10x10 pattern & mod)
generateMagicSquare	0.000112
isMagicSquare	0.313854
sumRow	0.000033
sumColumn	0.000032
allEqual	0.000000
isPairwiseDistinct	0.272737

These times were measured using omp_get_wtime() before and after the function was called and calculating the difference between them.

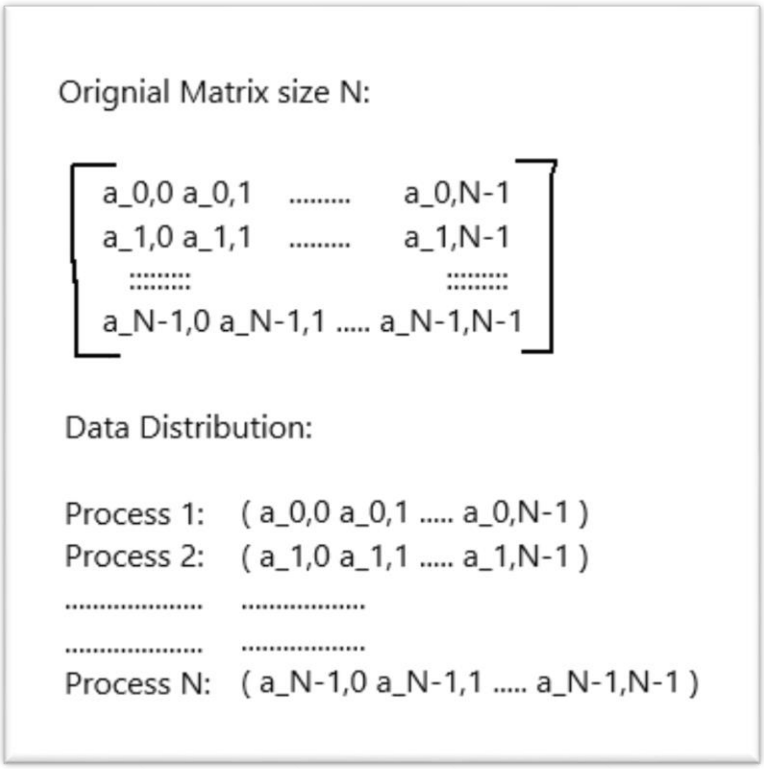
1.3

'#pragma omp declare target' declares a region for offloading computations to the GPU. 'target map()' specifies the data mappings between the host and target device, making sure necessary data is copied before and modifications are transferred after using 'map(to:)' and 'map(tofrom:)'. The 'collapse(2)' clause optimizes nested loops by collapsing them into a single loop for more efficient parallelisation, reducing overhead. The 'schedule(static)' statically assigns loop iterations among threads to enhance load balancing. 'Simd' exploits single instruction multiple data parallelism, applying a single instruction to multiple data elements simultaneously. The 'reduction()' clause specifies operations on loop variables to optimise computations of the sums and the 'aligned()' clause ensures proper memory alignment of data in memory for efficient vectorization. 'parallel' initiates a parallel region, executing code concurrently with multiple threads. 'atomic write' ensures atomicity during write operations, preventing data races among threads and maintaining variable correctness. 'barrier' provides synchronization points for the threads, which further prevents race conditions and ensures accurate results.

For an input matrix and modifier of size N=20, the runtime of the original code is 65.7675468922s whereas the GPU code is 206.3721199036s. This difference being 103.33%, indicating the GPU code is significantly slower than the original sequential code.

2.2

To parallelize magic_matrix.cpp using MPI (Message Passing Interface), a potential approach is to distribute the matrix rows across multiple processes. Each process handles a subset of rows and MPI functions facilitate communication and coordination between them.



Orignial Matrix size N:

$$\begin{bmatrix} a\_0,0 \; a\_0,1 & \cdots\cdots & a\_0,N\text{-}1 \\ a\_1,0 \; a\_1,1 & \cdots\cdots & a\_1,N\text{-}1 \\ \vdots & & \vdots \\ a\_N\text{-}1,0 \; a\_N\text{-}1,1 & \cdots\cdots & a\_N\text{-}1,N\text{-}1 \end{bmatrix}$$

Data Distribution:

```
Process 1:   ( a_0,0 a_0,1 ..... a_0,N-1 )
Process 2:   ( a_1,0 a_1,1 ..... a_1,N-1 )
...........  ...........
...........  ...........
Process N:   ( a_N-1,0 a_N-1,1 ..... a_N-1,N-1 )
```

The advantages of this approach are scalability as this method scales well with the number of MPI processes, enabling efficient parallelization on clusters as well as load balancing because distributing rows ensures even workload spread, optimizing performance.

These would be the main MPI functions used to parallelise magic_matrix.cpp:

MPI_Init and MPI_Finalize - Initialise and finalize MPI environment

MPI_Comm_rank and MPI_Comm_size - Determine the rank (process ID) and size (number of processes) to facilitate data distribution

MPI_Send and MPI_Recv - Implement point-to-point communication for exchanging boundary values between neighboring processes

MPI_Bcast - Broadcast the magic pattern and modifier matrices from one process to all others

MPI_Scatter: - Distribute rows of the matrix among MPI processes for parallel computation

MPI_Barrier: - Synchronize processes at specific points, ensuring coordination between computation phases, for example to ensure all sums have been calculated before checking if the new matrix is a magic matrix

MPI_Gather or MPI_Gatherv: - Gather the results from all processes to reconstruct the final magic square.

2.4

One way to make the installation of magic_matrix.cpp available to all users on the PVM is to use a NFS (Network File System). The executable would be stored on a centralised server which would be accessible by all the nodes. The NFS server exports directories from their local storage system to NFS clients who can mount them so that they can be accessed like local directories, making the file available to all users.

This approach ensures uniformity among nodes, as they all have access to the same version of magic_matrix.cpp. Updates and modifications can be efficiently managed from the central NFS server, simplifying maintenance. Furthermore, it minimizes storage redundancy as the file is stored centrally, reducing the risk of inconsistencies across nodes and therefore all the users.

However, this method also has some drawbacks to consider, mainly that it is network dependent and relies on a stable and fast connection, as a slow and unreliable network can significantly impact performance and may make the magic_matrix.cpp file unavailable. Additionally, NFS servers aren't ideal over the internet as latency issues can arise, especially in geographically dispersed clusters.