

# COMP3467 - Advanced Computer Systems: Parallel Programming and System Administration

Preface: I noticed that the CPU and GPU implementations of `magic_matrix.cpp` contain a function called `'isPairwiseDistinct'`. When this function is called by `'isMagicSquare'`, its return values are not interpreted correctly by `'isMagicSquare'` such that it has called a function called `'isNotPairwiseDistinct'` leading to incorrect results. To fix this, I amended the function `'isMagicSquare'` to ensure that it interprets a return false by `'isPairwiseDistinct'` to suggest that the matrix being processed is indeed not a magic square – and vice versa.

1.1)

a) Using “pattern3x3.dat” and “modifier3x3.dat”

Function	Completion Time (Average where necessary) - Seconds	Max Completion Time (for multiple function calls) - Seconds
generateMagicSquare	0.000001	0.000001
sumRow	0.00000011	0.000001
sumColumn	0.00000011	0.000001
allEqual	0.000000001	0.00000001
isPairwiseDistinct	0.000027	0.000027
isMagicSquare	0.000078	0.000078
main	0.000362	0.000362

b) Using “pattern10x10.dat” and “modifier10x10.dat”

Function	Completion Time (Average where necessary) - Seconds	Max Completion Time (for multiple function calls) - Seconds
generateMagicSquare	0.000000	0.00000001
sumRow	0.000001	0.000001
sumColumn	0.000001	0.000001
allEqual	0.0000005	0.000001
isPairwiseDistinct	0.228845	0.228845
isMagicSquare	0.230009	0.230009
main	0.230383	0.230383

### 1.3)

To efficiently port the code to GPUs in `magic_matrix_gpu.cpp`, I initiated with the existing codebase, utilizing OpenMP directives and timing functions (`omp_get_wtime()`) to identify performance bottlenecks, prominently focusing on the `'isPairwiseDistinct'` function. Initial attempts to parallelize individual loops using teams of threads did not yield significant optimization due to the function's inherent  $O(n^4)$  complexity.

Subsequently, I pursued a novel approach involving a complete overhaul of the `'isPairwiseDistinct'` function. This re-engineering incorporated the utilization of a large hash table structure within GPU memory. Parallel iteration across the matrix was established while maintaining a parallelised, unified iteration space. This strategy facilitated concurrent detection of duplicate elements within the matrix, accelerating processing time by allowing multiple element comparisons simultaneously and achieving constant-time responses.

The adopted directives for parallelization entailed restructuring loops within critical functions (`'generateMagicSquare'`, `'isMagicSquare'`, `'sumRow'`, `'sumColumn'`) using OpenMP techniques like parallel regions, loop parallelization, and data mapping to optimize memory access patterns. The hash table usage and the concurrent processing of matrix elements led to significant runtime improvements, overcoming the inherent computational complexity of the original `'isPairwiseDistinct'` function on the GPU. This approach showcases a drastic enhancement in performance, aligning with the GPU's architecture for parallel processing and memory management, resulting in immensely substantial speed-ups in run times.



The GPU program took 0.00554108619 seconds on average to complete.

The default program took 52.517722129821777 seconds on average to complete.

### 2.2)

To parallelize `magic_matrix.cpp` using MPI, I would distribute the matrix among different MPI processes, assigning each process a distinct subset of rows or columns. This approach, often termed as data decomposition, facilitates workload distribution among the MPI processes. For instance, if I have 'p' MPI processes, I would distribute the  $N \times N$  matrix in a block or cyclic manner across these processes. Each process would be responsible for computing a specific portion of the matrix.

(As seen in the diagram, "P 0" handles rows 0 and 1 of the example matrix, "P 1" handles rows 2 and 3).

Process	Matrix:			
P 0 	a	b	c	d
	e	f	g	h
P 1 	i	j	k	l

	m	n	o	p
--	---	---	---	---

And so and so forth...

I would utilize MPI functions such as `MPI_Send` and `MPI_Recv` to exchange boundary data between adjacent processes as necessary, ensuring correct computation involving matrix elements at the interface of these subsets. Additionally, MPI collective communication functions like `MPI_Bcast` or `MPI_Scatter` can broadcast or scatter necessary data across processes.

By using MPI's communication skills for inter-process synchronized, data transfer, alongside proper decomposition of the matrix among MPI processes, I would be able to achieve parallel computation of the magic square matrix without any data redundancies or conflicts. MPI also lends itself to ease in scalability of matrices, showing a linear improvement in computation time with an increasing number of MPI processes

2.4)

To make `magic_matrix.cpp` universally available on a PVM within a virtual machine cluster setup, I would follow a few approaches:

One approach would be to make use of a shared filesystem. Placing `magic_matrix.cpp` on a shared filesystem (e.g., NFS - Network File System) accessible to all nodes in the virtual machine cluster ensures universal availability. Users can access, compile, and execute the code from any node within the cluster. This approach simplifies accessibility and maintenance, ensuring consistent versions across the cluster.

The other approach would be to ensure a robust version control system is in place. Employing a version control system (e.g., Git) allows users to clone the repository housing `magic_matrix.cpp` onto their respective nodes. This method ensures version consistency while allowing individual users to work independently by branching and merging as necessary.

Advantages:

- **Universal Accessibility:** Users across the cluster can access the code from any node, promoting collaboration and code sharing.
- **Consistency:** Shared filesystems or version control systems maintain uniformity in code versions and updates.

Disadvantages:

- **Dependency on Network Stability:** Shared filesystems rely on network stability. Network issues may impact access and performance.
- **Synchronization Challenges:** Concurrent modifications might cause conflicts in version control systems, necessitating proper merging strategies.