# A Novel High Performance Inplace Transpose Algorithm

Amir Gholami[1], Bragadeesh Natarajan[2]
[1] University of Texas at Austin, [2] Advanced Micro Devices

## 1 Introduction

Transpose is an important algorithm in many applications applications such as Fast Fourier Transform (FFT). It is a memory-bound problem, because it only involves data movements. Hence, it cannot hide cache misses, bank conflicts, channel conflicts, etc. with computations as there are none. Therefore, getting a high performance inplace transpose kernel is challenging. However, an outplace transpose kernel is straightforward, but it has memory overhead and its largest size is limited to less than half of the allocatable memory on a device. Inplace transpose does not have this memory limitation, but it does not have a good performance since parallelizing it is not easy. This IDF presents a novel square transpose algorithm that solves this problem and provides orders of magnitude speedup over classical inplace algorithm. Our kernel achieves the same or better performance in comparison to outplace transpose kernel, even though we use 50% of the resources. We present a largely consistent high performance solution for very large sizes upto the maximum allocatable buffer.

## 2 Novelty

- A generic new triangular staggered algorithm that removes load imbalance and channel conflicts for square inplace transpose.
- The algorithm avoids extreme performance loss encountered by classic inplace transpose algorithms at important sizes.
- The Largest problem size that can be used, is extended to the full available memory, instead of half of it.
- Parity or better performance compared to outplace transpose for problem sizes that are below half the size of GPU global memory (due to outplace transpose memory overhead), and competing performance for very large transpose sizes.
- Patent infringement is easily detectable because the new triangular staggered algorithm uses specific formulas to compute indices. Those can be easily identified in machine or object code.

## 3 Algorithm Description

The transpose algorithm for a $N \times N$ matrix A is defined as follows:

$$A[i \times N + j] \xrightarrow{T} A[j \times N + i] \quad \forall \;\; 0 \leq i, \; j < N. \tag{1}$$

To perform the transpose, each work group has to exchange its block of data with its transpose block, without using any additional global memory buffer. Therefore, only work groups whose index falls above or on the diagonal of the input matrix are busy. The rest of the work groups will remain idle (the highlighted area in Fig. 1-a). Another important problem is channel conflicts. Inherent in the transpose algorithm, each work group has to access both rows and columns of the matrix. Therefore, it is inevitable that linearly ordered work groups access the same channel in the global memory. When channel conflict happens, the GPU hardware serializes the load/store operation which severely affects performance. This severely hits performance as shown in Fig. 2.

It is possible to solve the load imbalance issue by using a triangular work group indexing as shown in Fig. 1-b. However, this distribution suffers from a similar channel conflict problem. Our novel approach is a new pivoted staggered distribution which solves both the load imbalance, and the channel conflicts. In this approach, we first compute the triangular indexing and then perform a column wise pivoting to get a triangular staggered distribution. As a result, consecutive work groups will not access the same global memory channel, and therefore the load/store operations will not be serialized. Our approach achieves more than 100× speedup as will be discussed in the Results section.

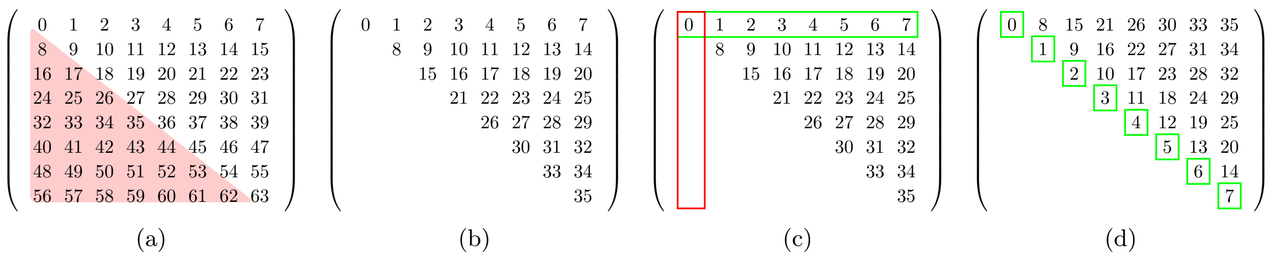


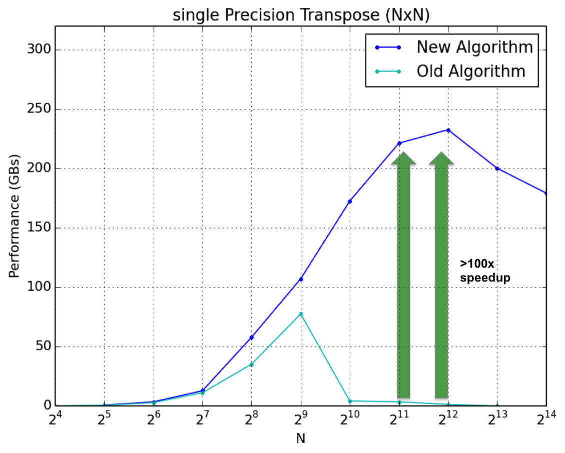
Figure 1: Work group indexing is shown for different cases. (a) a linear 2D work group indexing; work groups in the highlighted area are idle because of the load imbalance. (b) a triangular indexing that solves the load imbalance issue of (a). (c) Channel conflicts happen when each work group tries to load/store data at the transpose location (red), but no conflict for load/stores of upper triangular blocks (green). (d) staggered triangular indexing is shown, which can be computed by pivoting the triangular numbers for each column. This effectively removes channel conflicts, as well as load imbalance.

## 4 Results

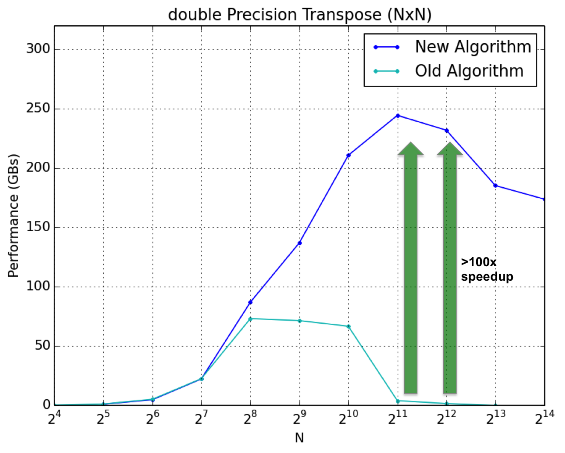
The performance of the new algorithm is tested on AMD's W9100 GPU for a range of transpose sizes, as shown in Fig. 2. The new algorithm (Fig. 1-d) is compared against the classical work group distribution scheme (Fig. 1-a). The results show more than **100**× speedup is achieved for large transpose sizes. Here, the largest size tested is 2GB, which corresponds to a transpose of size $16k \times 16k$. Since the inplace transpose does not have any memory overhead, we scaled the problem size upto 11.7 GB, as shown in Fig. 3. Furthermore, we compare our inplace tranpose with clFFT's outplace transpose kernel. Since outplace transpose has a 100% memory overhead, it cannot be used beyond $\sim$ 5GB transposes. This is indeed not a limitation for the inplace kernel, and it achieves very good performance upto the largest case of 11.7 GB, which is the maximum allocatable size on a W9100 card. A major significance of this result is that it extends AMD libraries to support very large sizes.

## 5 Conclusion

We presented a novel triangular staggered algorithm to perform square inplace transpose. Not only does this algorithm achieve competitive performance for general sizes, it also gets excellent performance for sizes that can pose challenges due to hardware memory limitations. This is a generic solution that handles any square size on any hardware with good performance.

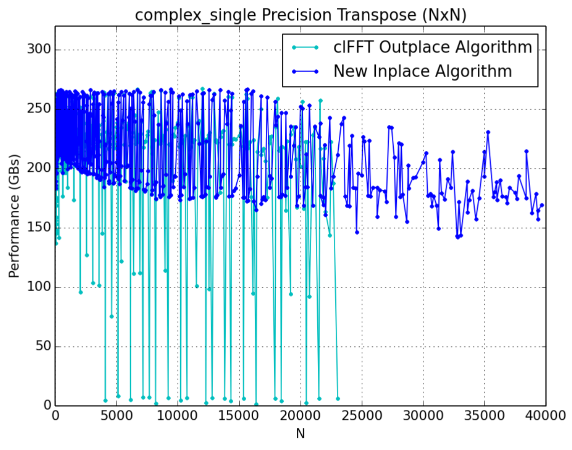


(a)

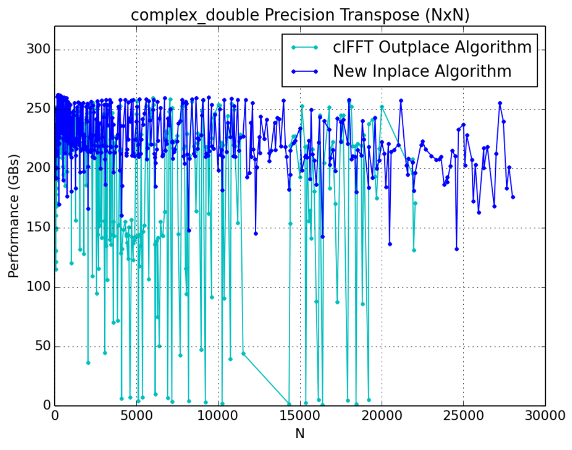


(b)

Figure 2: Performance of the new algorithm versus the classical inplace algorithm performed on AMD W9100 firepro card. The new algorithm achieves more than 100× speedup for powers-of-2 sizes. Results are shown for both single (left) and double precisions (right).



(a)



(b)

Figure 3: Performance of the new algorithm compared to clFFT's outplace transpose for sizes that are multiples of 2,3,5, or 7. Even though the inplace code uses 50% of the resources, it achieves the same or better performance. Due to memory overhead, clFFT's outplace transpose cannot be used for very large sizes. This is not a limitation for inplace transpose, and our algorithm handles those cases with a very good performance. The largest test case corresponds to 11.7GB which is the maximum allocatable size on W9100. Results are shown for both single (left) and double precisions (right). The peak bandwidth of W9100 is 320 GBs.