



# Discriminant Adaptive Nearest Neighbors Classification Implementation with OpenCL and OpenMP

Jonathan Lisic

# DANN Method

- Supervised Classification Method
- Uses Nearest Neighbor Classification within a localized neighborhood
- Localized neighborhoods are determined through the DANN metric
- The DANN metric is a distance measure that is adjusted for between class variance and is sphered by the pooled variance

# DANN Metric

$$d(y, x_0) = (y - x_0)^t \Sigma (y - x_0)$$

Where:

$$\Sigma = W^{-1/2} (W^{-1/2} B W^{-1/2} + \varepsilon I) W^{-1/2}$$

$$B = \sum_{k \in G} \pi_k (\bar{x}_k - \bar{x})(\bar{x}_k - \bar{x})^t \quad // \text{ Between class covariance}$$

$$W = \sum_{k \in G} \pi_k \text{COV}(x_k) \quad // \text{ Within class covariance}$$

$$W^{-1/2} = P \Lambda^{-1/2} P^t$$

# Neighborhoods Created by DANN

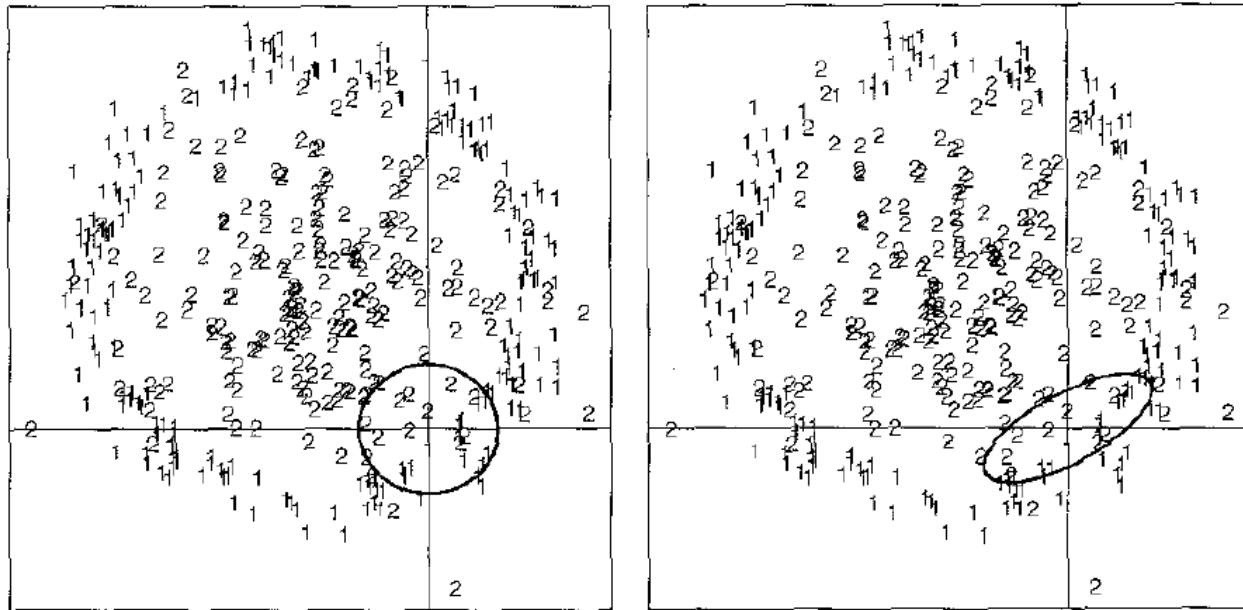
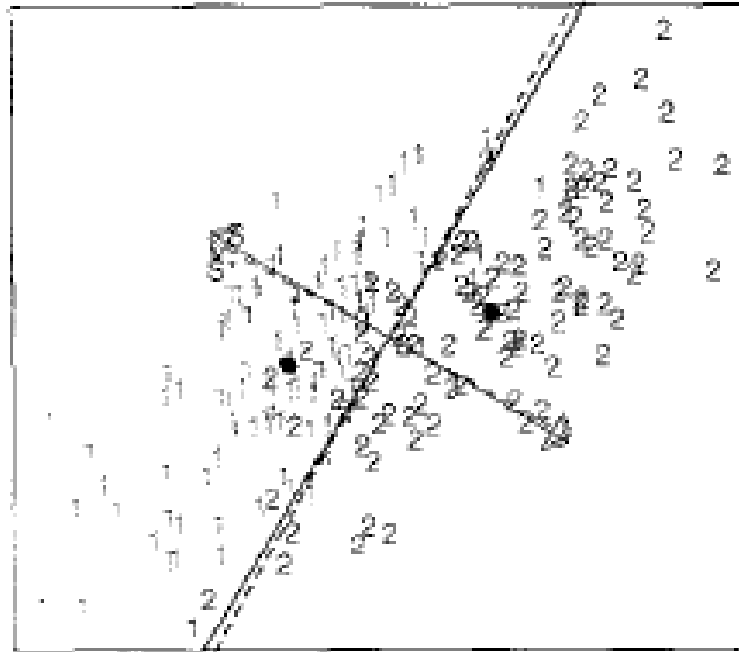


Fig. 2. The left panel shows the spherical neighborhood containing 25 points. The right panel shows the ellipsoidal neighborhood found by the DANN procedure, also containing 25 points. The latter is elongated along the true decision boundary, and flattened orthogonal to it.

*Discriminant adaptive nearest neighbor classification.* **Hastie, Trevor and Tibshirani, Robert.** 1996, IEEE Trns Pattern Analysis Machine Intelligence, pp. 607-616.

# Neighborhood With LDA Boundaries

LDA and Local Subspaces —  $K = 25$



The Shorter Boundary indicates DANN distances are larger in that direction per unit movement in the plane.

*Discriminant adaptive nearest neighbor classification.* **Hastie, Trevor and Tibshirani, Robert.** 1996, IEEE Trns Pattern Analysis Machine Intelligence, pp. 607-616.

# Motivation

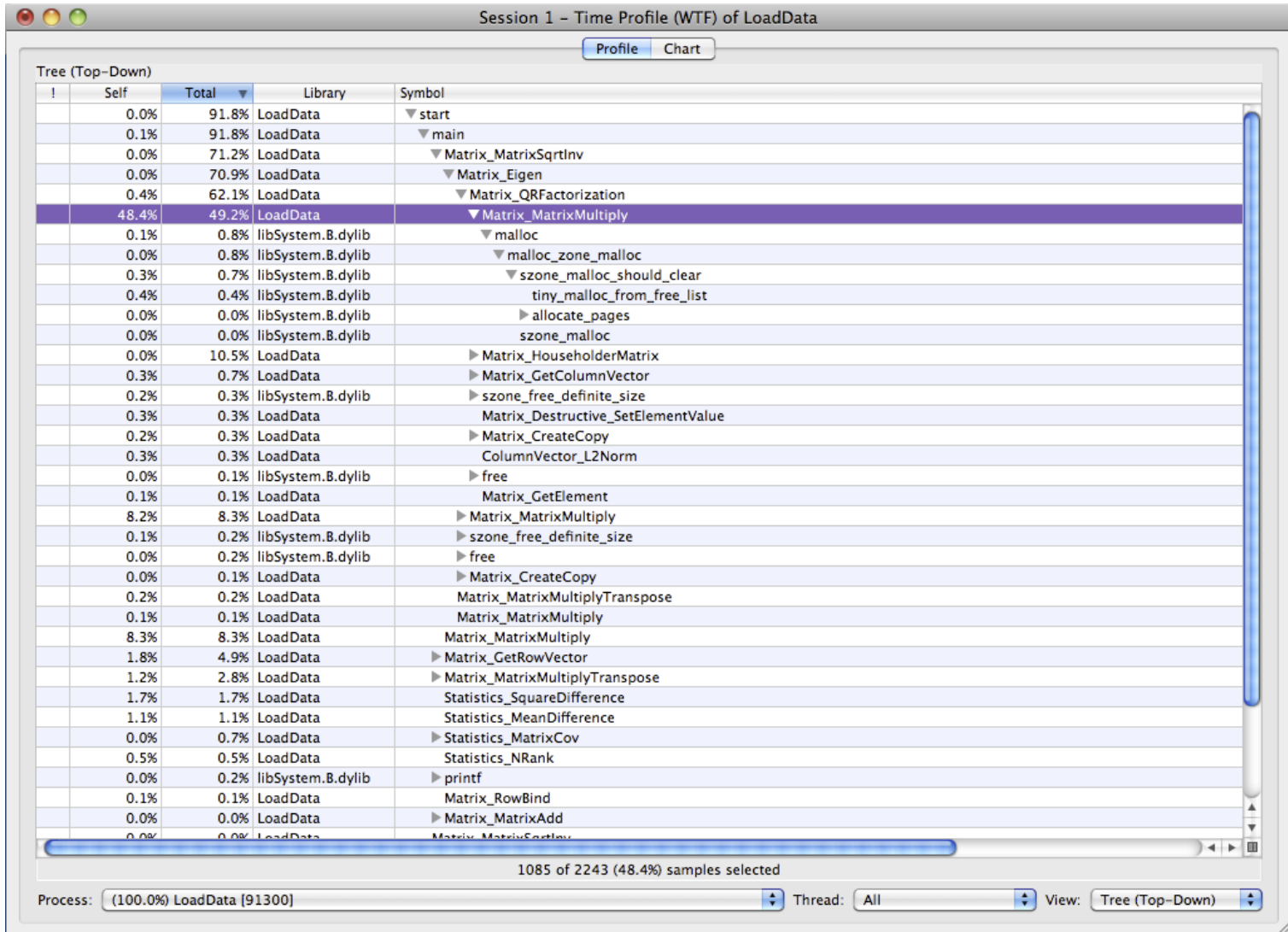
- Original R implementations are quite slow for large numbers of query points
- A C implementation would give further flexibility for future research
- OpenMP is a known method to improve performance of embarrassingly parallel problems
- OpenCL is new and promising method to improve performance
- Learn how to use OpenCL

# Implementation (C)

- The C based implementation was entirely written from scratch
- BLAS was not used
- Matrices were kept in row-major form



# Analysis of C Implementation



# Eigenvalue and EigenVector Calculation

- Method used was QR iteration with orthogonalization via Householder transformations
- QR did not use shifts
- QR is fairly fast and very simple to implement
- Householder transformations are fairly stable

# QR Algorithm

1.  $A_0 = A$
2. **FOR**  $k = 1, 2, 3, \dots$
3.  $Q_k R_k = A_{k-1}$
4.  $A_k = R_k Q_k$
5. **END**

**Heath, Michael T.** *Scientific Computing An Introductory Survey 2nd Ed.* New York : McGraw-Hill, 2002.

# Householder Algorithm

1.  $A_1 = A$
2. FOR  $k = 1$  TO  $n - 1$
3.  $\|A_k[:, k]\|_2$  // Calculate  $l_2$  norm of matrix  $A_k$ 's  $k^{\text{th}}$  column vector
4.  $a = -\text{sign}(A_k[k, k])$  // Get the opposite sign of the diagonal from  $A_k$
5.  $v_k = A[:, k] - a(\|A[:, k]\|_2)I[:, k]$  // Calculate the vector for the Householder transformation where  $I$  is the identity matrix
6.  $H_k = I - 2(v_k v_k') / (v_k' v_k)$  // Compute the Householder Matrix
7.  $A_{k+1} = H_k A_k$
8. END

**Heath, Michael T.** *Scientific Computing An Introductory Survey 2nd Ed.* New York : McGraw-Hill, 2002.

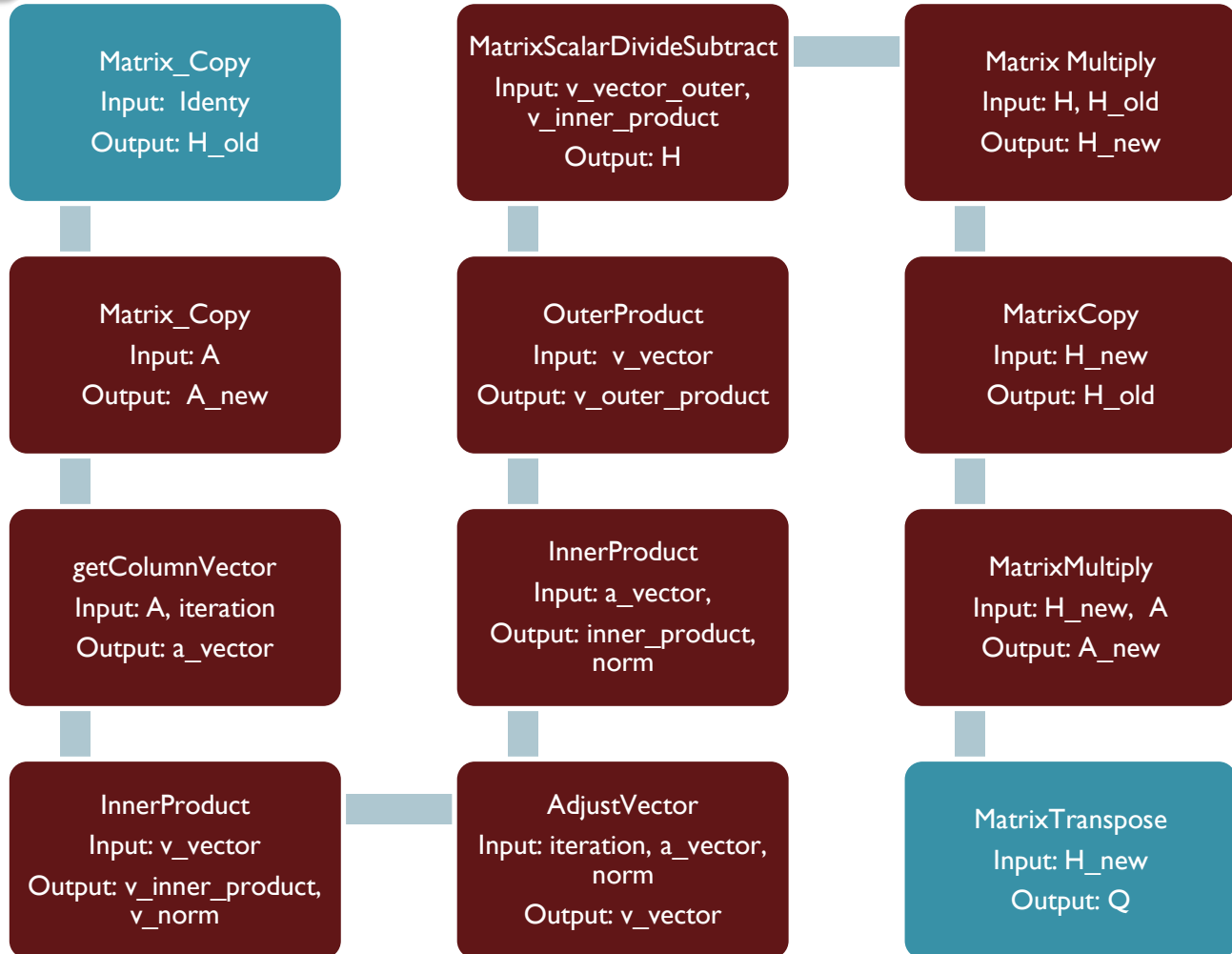
# OpenMP Implementation

- There was no real difference in performance for any combination of scheduling since each iteration required a similar number of eigenvalue computations to converge
- Fairly simple to implement
- Good Performance results

# OpenCL Implementation

- Initial implementation was torturous to get functioning
- Few examples of how to work with multiple kernels
- NVIDIA OpenCL Programming Guide was very useful, and based several kernels on examples provided
- There are numerous implementation specific incompatibilities within the examples and several errors
- Unoptimized performance was much worse than the serial C implementation

# Overview of Householder Algorithm on GPU



# Example Kernel

```
__kernel void matrixmultiply(
    __global int * A_height,
    __global int * A_width,
    __global float * A_elements,
    __global int * B_height,
    __global int * B_width,
    __global float * B_elements,
    __global int * C_height,
    __global int * C_width,
    __global float * C_elements)
{
    int i;
    float local_sum = 0;
    int row = get_global_id(0) / *A_width;
    int col = get_global_id(0) % *B_width;
    for ( i = 0; i < *A_width; i++ ) {
        local_sum += A_elements[row * (*A_width) + i ]
                    * B_elements[i * (*B_width) + col];
    }
    C_elements[get_global_id(0)] = local_sum;
    *C_height = *A_height;
    *C_width = *B_width;
}
```

# Performance Results

Performance for Classification of 2000 Query Points (Time in Seconds)

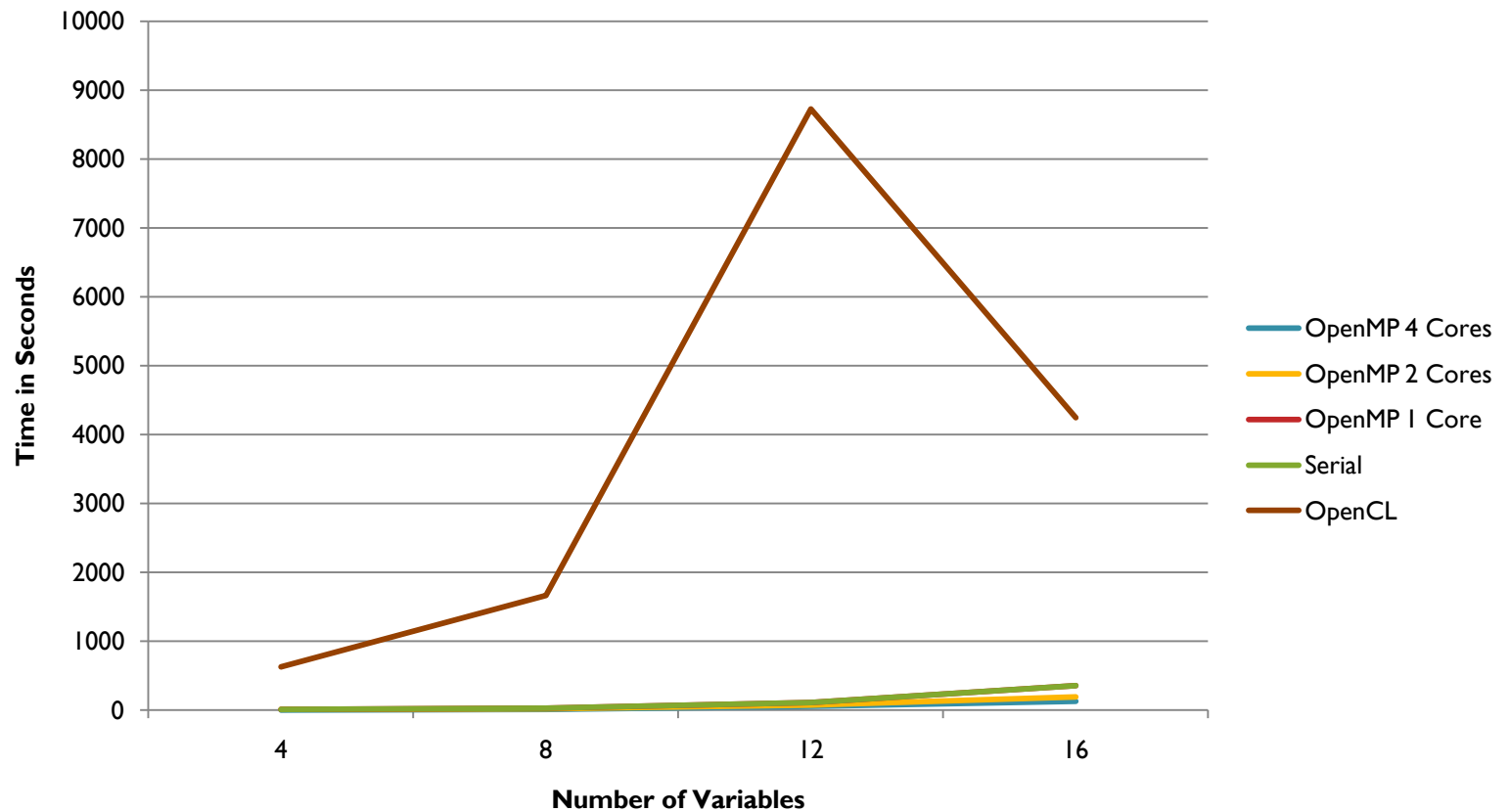
		4 Variables	8 Variables	12 Variables	16 Variables
R (Optimized)		38.810	40.307	N/A	N/A
OpenMP	4	1.069	9.68	47.714	123.75
	2	1.414	13.795	67.028	184.159
	1	2.275	24.206	110.118	350.368
Serial		2.189	24.239	109.644	350.52
OpenCL		626.55	1656.734	8725.735	4243.323

Epsilon = 1, 100 Training points for each population, Initial Neighborhood 10, Secondary Neighborhood 5

R Implementations crashed with over 10 variables

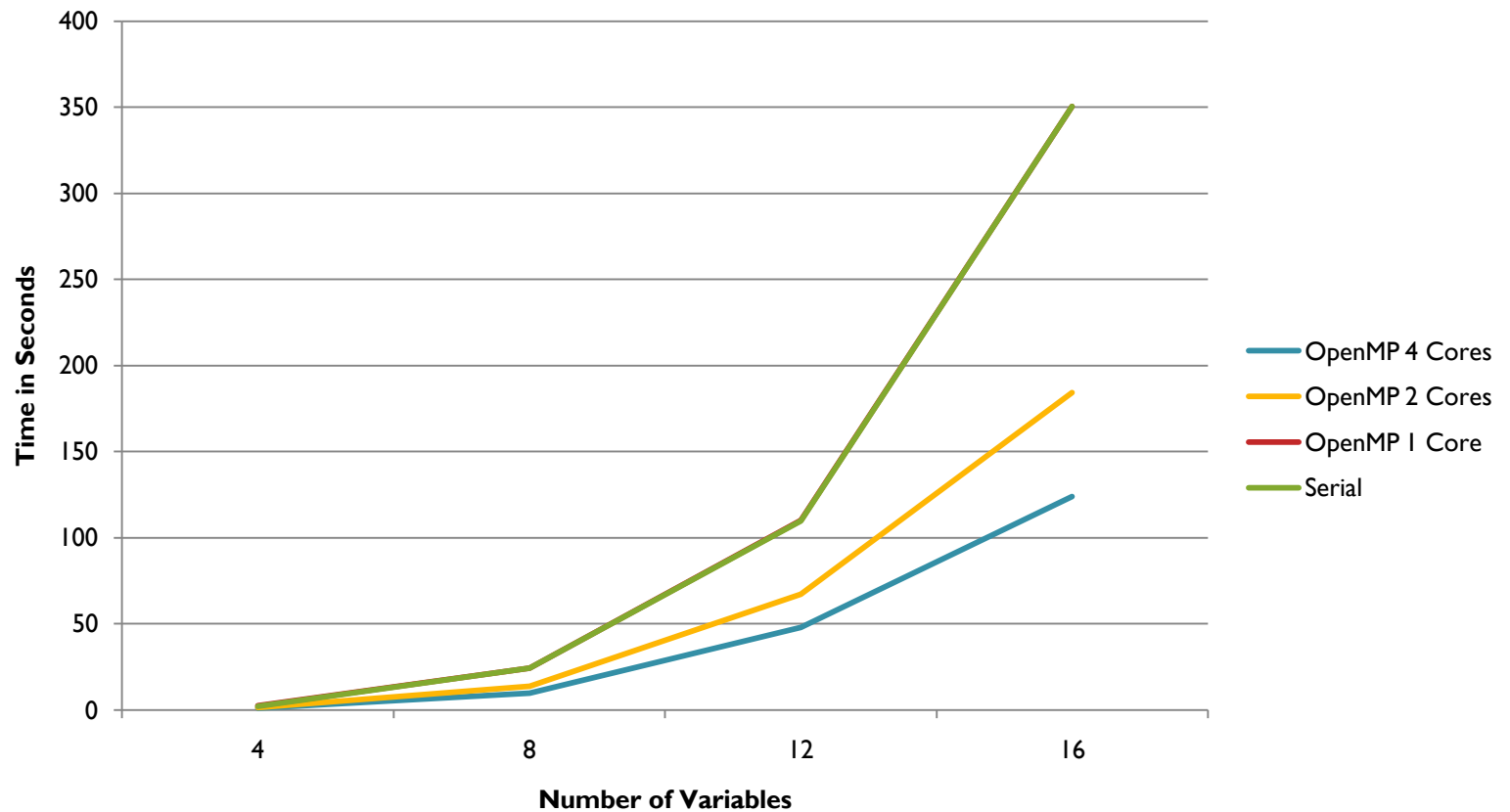
# Performance Results

## Performance for Classification of 2000 Query Points



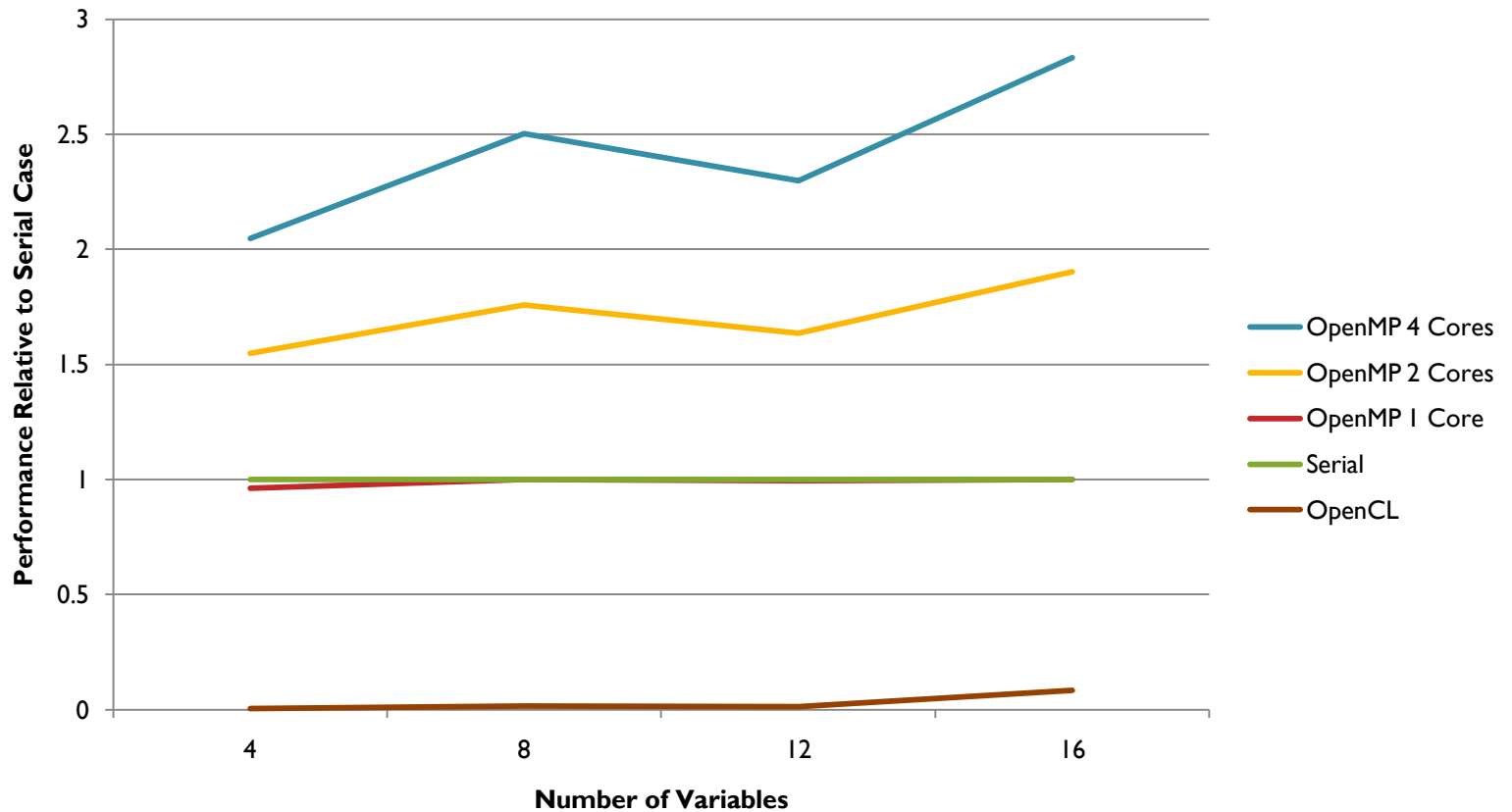
# Performance Results

## Performance for Classification of 2000 Query Points



# Performance Results

## Performance Improvement Over Serial Case for Classification of 2000 Query Points



# Time to Implement

- Time to Write Serial C Implementation:
  - 3 Weeks
- Time to Move Code to OpenMP
  - 1 Week
- Time to Move Code to OpenCL
  - 3 Weeks

# Conclusion

- Just moving to C improved the code tremendously
- Implementing OpenMP is quick and relatively painless, and provides a noticeable improvement in speed
- OpenCL works best for small numbers of well defined kernels, and can gain appreciable performance increases for certain problems
- An Implementation of DANN on GPU's should work well, but will require more time and better profiling and performance tools than the ones used for this project



**Questions?**