

“Accelerating Machine Learning Applications on Graphics Processors” by Narayanan Sundaram and Bryan Catanzaro

“cuSVM: A CUDA Implementation Of Support Vector Classification And Regression” by Austin Carpenter

presented by  
Mariusz Wzorek

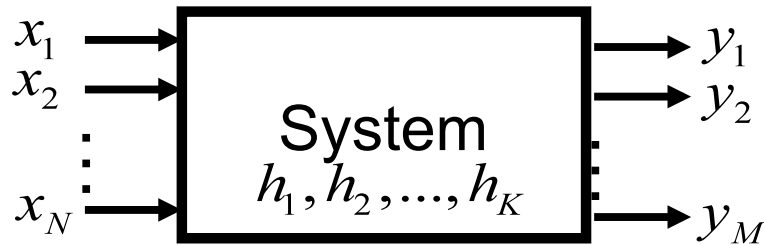
## Machine Learning

“The goal of machine learning is to build computer systems that can adapt and learn from their experience.”

– Tom Dietterich

Machine Learning algorithms discover the relationships between the variables of a system (input, output and hidden) from direct samples of the system

# A Generic System



Input Variables:  $\mathbf{x} = (x_1, x_2, \dots, x_N)$

Hidden Variables:  $\mathbf{h} = (h_1, h_2, \dots, h_K)$

Output Variables:  $\mathbf{y} = (y_1, y_2, \dots, y_K)$

3

## GPUs as proxy for manycore

- Transitioning from highly specialized pipelines to general purpose
- The only way to get performance from GPUs is through parallelism (No caching, branch prediction, prefetching etc.)
- Can launch millions of threads in one call

4

# GPUs are not for everyone

- Memory coalescing is really important
- Irregular memory accesses to even local stores is discouraged - up to 30% performance hit on some apps for local memory bank conflicts
- Cannot forget that it is a SIMD machine
- Memory consistency is non-existent & inter-SM synchronization is absent
- Hardware scheduled threads
- 20 us overhead for kernel call (20,000 instructions @ 1GHz)

5

## NVIDIA GeForce 8800 GTX Specifications

	8800 GTX (paper 1)	GTX 260 (paper 2)
Number of Streaming Multiprocessors	16	24
Multiprocessor Width	8	8
Local Store Size	16 KB	
Total number of Stream Processors	128	192
Peak SP Floating Point Rate	346 Gflops	715 Gflops
Clock	1.35 GHz	1.242 GHz
Device Memory	768 MB	896 MB
Peak Memory Bandwidth	86.4 GB/s	111.9 GB/s
Connection to Host CPU	PCI Express	PCI Express
CPU -> GPU bandwidth	2.2 GB/s*	
GPU -> CPU bandwidth	1.7 GB/s*	

\* measured values

6

# Machine Learning Algorithms

- Support Vector Machines
- Two classes of problems:
  - $(x_i, y_i), i = 1, \dots, l$   
where  $x_i \in \mathbb{R}^n$  and  $y \in \{1, -1\}$
  - $x_i \in \mathbb{R}^n, i=1, \dots, m$   
and target output  $z \in \mathbb{R}^m, z_i \in \mathbb{R}^1$
- 

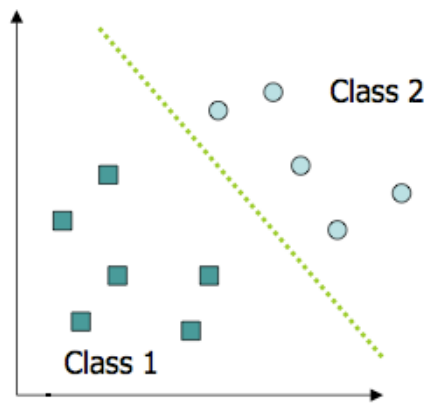
7

## Support Vector Machines

- Very popular machine learning technique
- Tries to find a hyperplane separating the different classes with “maximum margin”
- Non-linear surfaces can be generated through non-linear kernel functions
- Uses Quadratic Programming for training (specific set of constraints imply a wide variety of techniques for solving it)

8

# Linear Separable Case



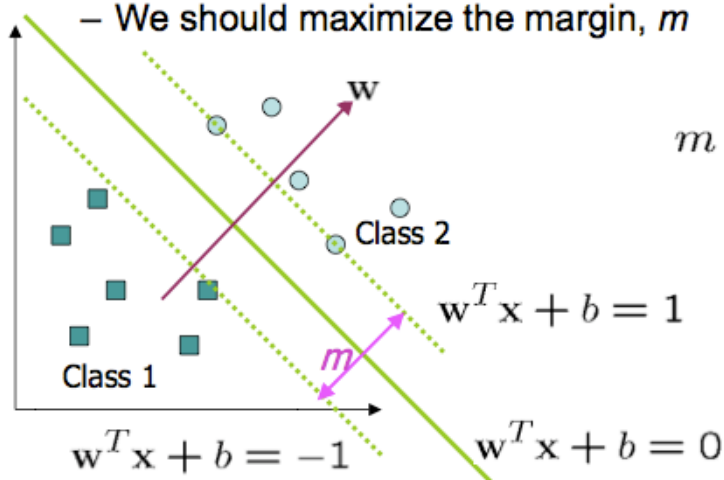
- Many decision boundaries can separate these two classes
- Which one should we choose?

9

# Linear Separable Case

- The decision boundary should be as far away from the data as possible

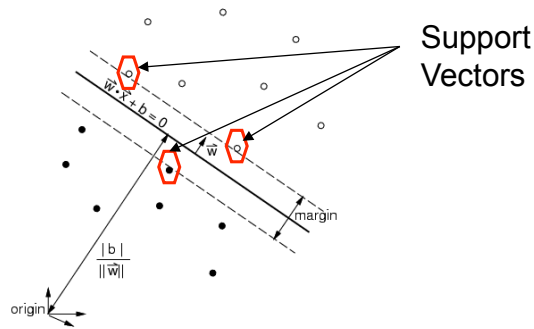
– We should maximize the margin,  $m$



$$m = \frac{2}{\|\mathbf{w}\|}$$

10

# Linear Separable Case



We'd like the hyperplane with maximum margin

- size of margin is  $\frac{2}{\|\vec{w}\|}$

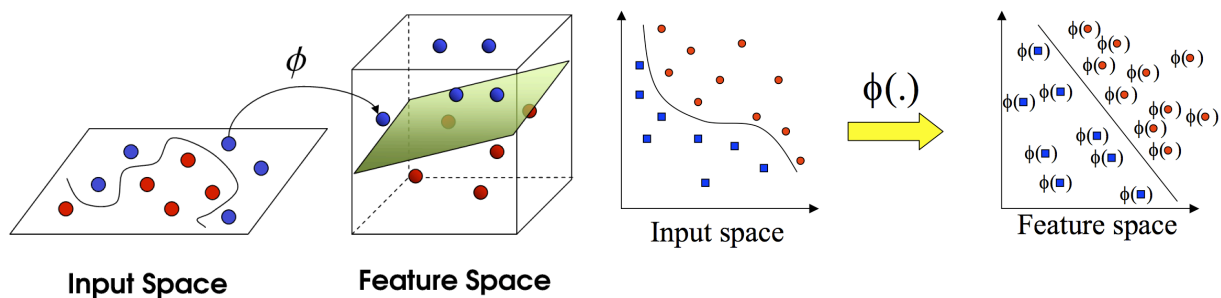
So view our problem as a constrained optimization problem:

Minimize  $\|\vec{w}\|^2$ , subject to

$$y_i(\vec{x}_i \cdot \vec{w} + b) - 1 \geq 0, (\forall i)$$

11

# Non-linear Case



- Define the kernel function  $K(\mathbf{x}, \mathbf{y})$  as
 
$$K(\mathbf{x}, \mathbf{y}) = (1 + x_1 y_1 + x_2 y_2)^2$$
- Consider the following transformation
 
$$\phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1)$$

$$\phi\left(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}\right) = (1, \sqrt{2}y_1, \sqrt{2}y_2, y_1^2, y_2^2, \sqrt{2}y_1)$$

$$\langle \phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right), \phi\left(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}\right) \rangle = (1 + x_1 y_1 + x_2 y_2)^2 = K(\mathbf{x}, \mathbf{y})$$
- The inner product can be computed by  $K$  without going through the map  $\phi(\cdot)$
- The relationship between the kernel function  $K$  and the mapping  $\phi(\cdot)$  is
 
$$K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$$
 – This is known as the *kernel trick*
- In practice, we specify  $K$ , thereby specifying  $\phi(\cdot)$  indirectly, instead of choosing  $\phi(\cdot)$
- Intuitively,  $K(\mathbf{x}, \mathbf{y})$  represents our desired notion of similarity between data  $\mathbf{x}$  and  $\mathbf{y}$  and this is from our prior knowledge
- $K(\mathbf{x}, \mathbf{y})$  needs to satisfy a technical condition (Mercer condition) in order for  $\phi(\cdot)$  to exist

12

# Non-linear Case kernel examples

LINEAR	$\Phi(x_i, x_j) = x_i \cdot x_j$
POLYNOMIAL	$\Phi(x_i, x_j; a, r, d) = (ax_i \cdot x_j + r)^d$
GAUSSIAN	$\Phi(x_i, x_j; \gamma) = \exp\{-\gamma\ x_i - x_j\ ^2\}$
SIGMOID	$\Phi(x_i, x_j; a, r) = \tanh(ax_i \cdot x_j + r)$

13

## SVM Training

- Quadratic Program

$$\max \sum_{i=1}^l \alpha_i - \frac{1}{2} \alpha^T Q \alpha$$

$$s.t. \quad 0 \leq \alpha_i \leq C, \quad \forall i \in [1, l]$$

$$y^T \alpha = 0$$

$$Q_{ij} = y_i y_j \Phi(x_i, x_j)$$

- Some kernel functions:

$$\Phi(x_i, x_j; a, r, d) = (ax_i \cdot x_j + r)^d$$

$$\Phi(x_i, x_j; \gamma) = \exp\{-\gamma\|x_i - x_j\|^2\}$$

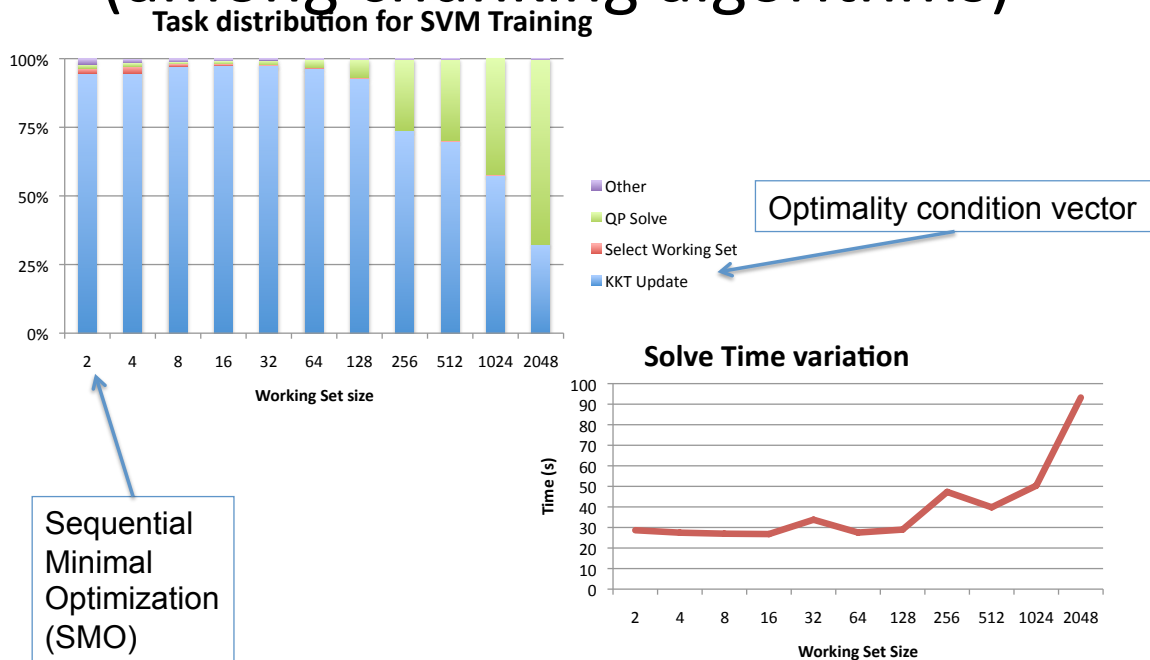
Variables:

$\alpha$ : Weight for each training point (determines classifier)

Data:

$l$ : number of training points  
 $C$ : trades off error on training set for generalization performance  
 $y$ : Label (+/- 1) for each training point  
 $x$ : training points

# Choice of parallel algorithm (among chunking algorithms)



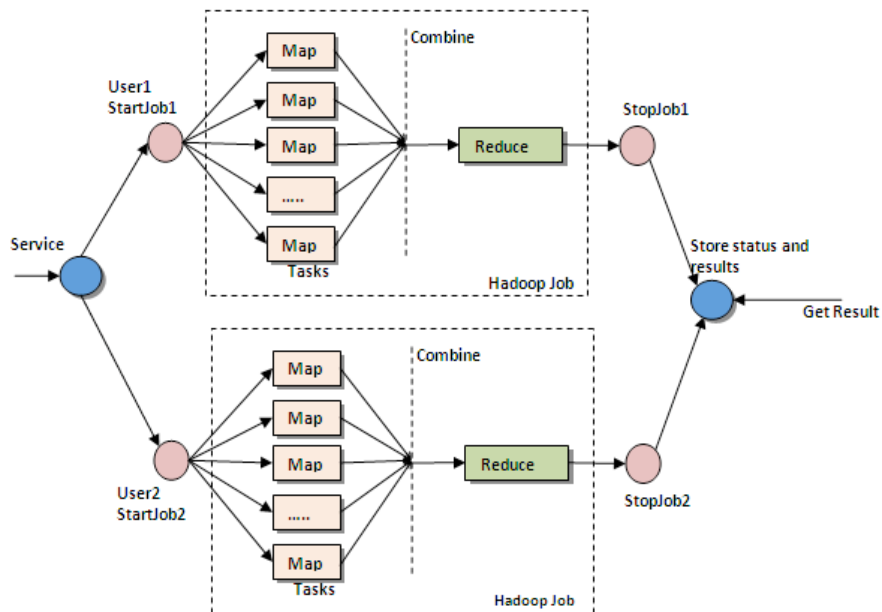
15

## Fitting SMO on a GPU

- Shared memory constraints on the GPU fits the algorithm as only two vectors need to be shared among all the threads
- Performance strongly dependent on the choice of the working set
- Several heuristics proposed – two are popular (1<sup>st</sup> and 2<sup>nd</sup> order)
- 2<sup>nd</sup> order heuristic is almost twice as costly, but saves on the number of iterations

16

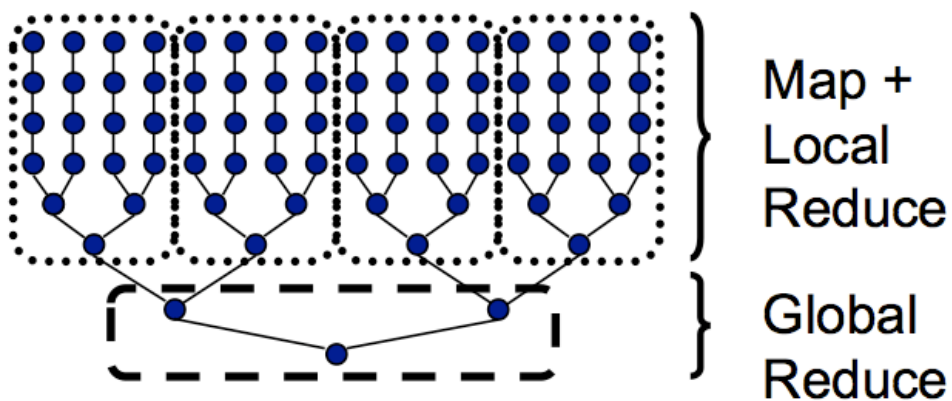
# Map Reduce by Google



17

# Map Reduce by Google

- Map function: optimality condition vector (KTT update) for each data point -> dedicated thread
- Reduce function: search for min/max values of  $b$  and  $l$



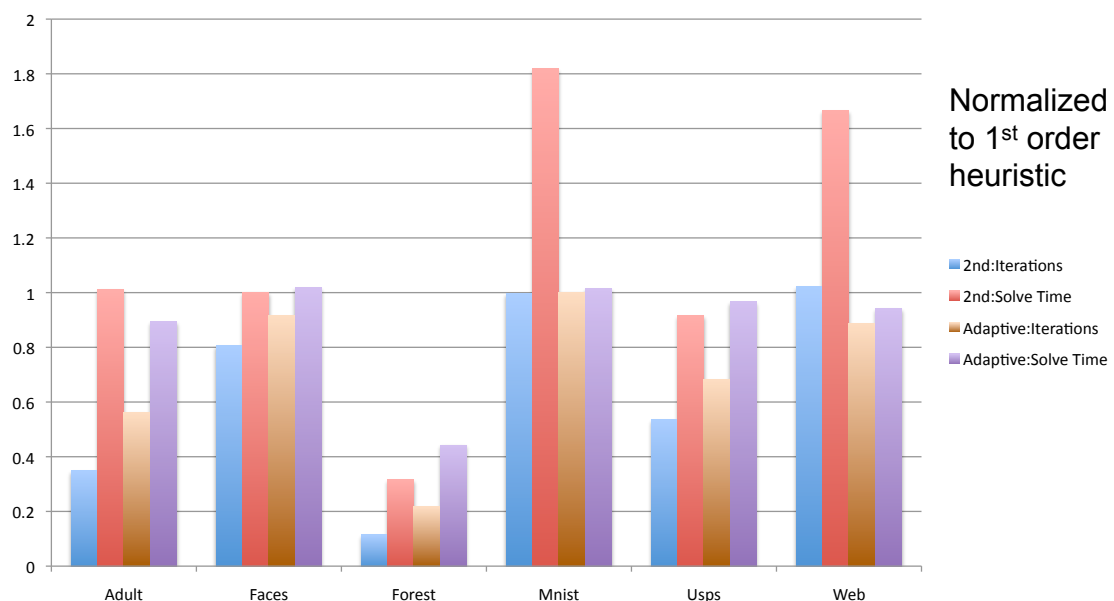
18

# Adaptive heuristic

- Both heuristics can be expressed as a series of “Map Reduce” stages
- A Map Reduce code generator was used to generate the code
- Sample periodically and adapt depending on the most converging heuristic at any given time
- Tightly coupled map-reduces are essential for machine learning algorithms

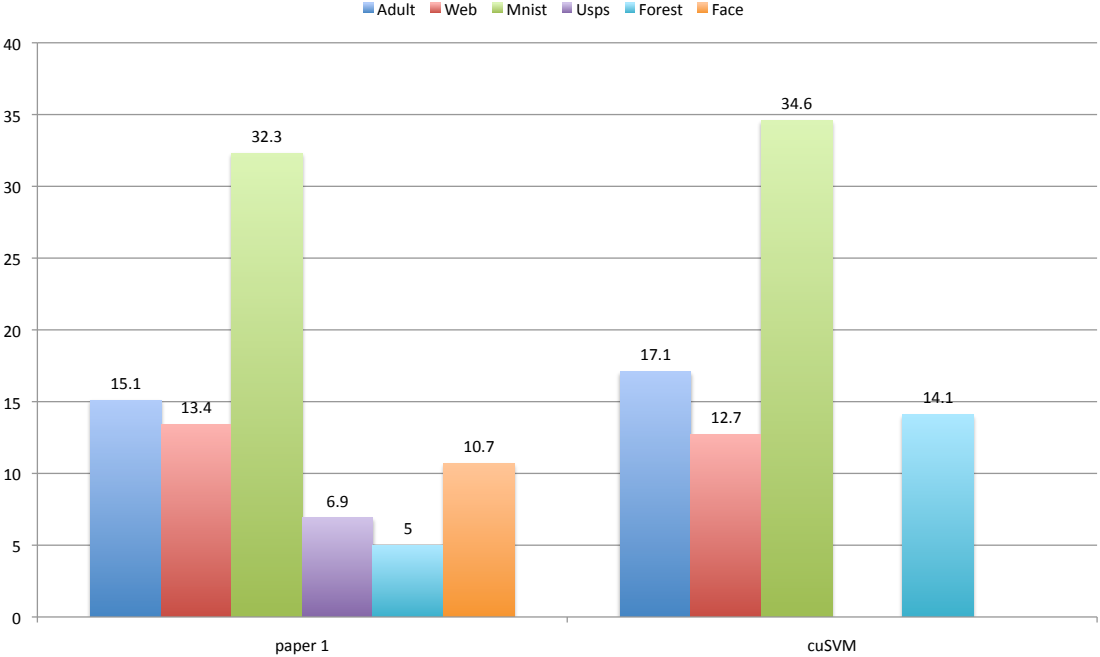
19

## Results (paper 1)

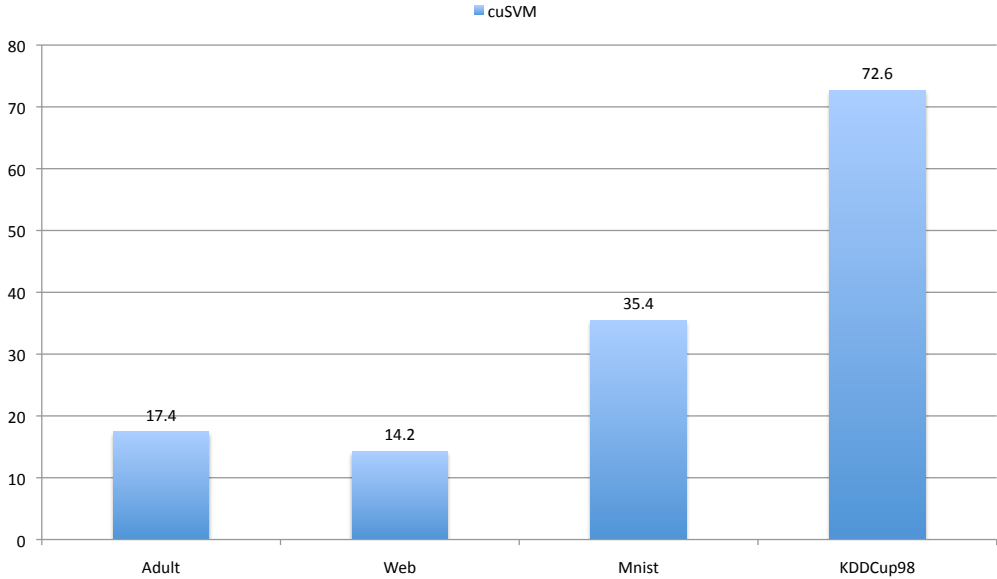


20

# Overall speedup compared to LIBSVM (C-SVC)



# Overall speedup compared to LIBSVM (e-SVR)



# SVM Classification

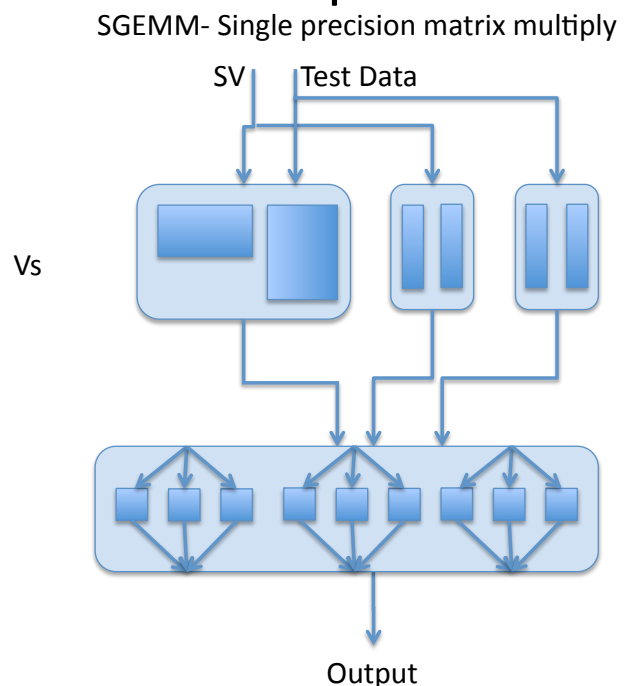
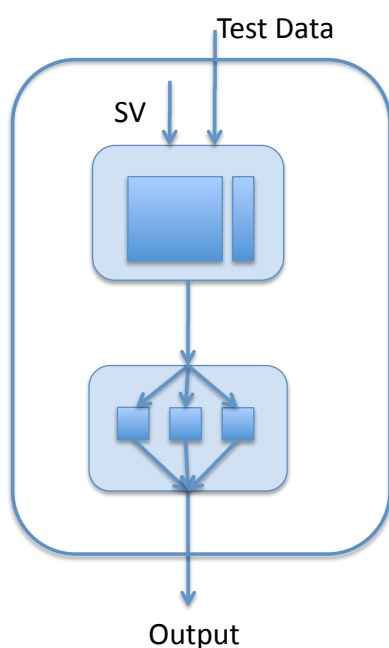
- SVM classification task involves finding which side of the hyperplane a point lies on
- Specifically,

$$y = \text{sign}\left(\sum_{i=1}^N \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b\right)$$

where  $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2}$

- Insight : Instead of doing this serially for all points, note that  $\|\mathbf{x}_i - \mathbf{x}_j\|^2 = \|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2 - 2\mathbf{x}_i \cdot \mathbf{x}_j$

## Restructuring the Classification problem

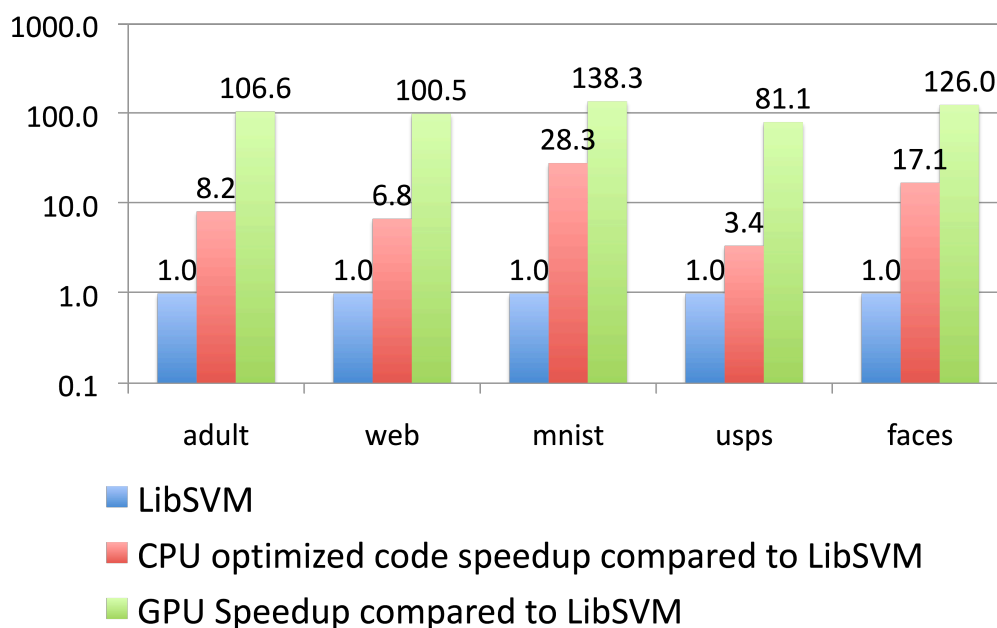


# Results (C-SVC)

Dataset	LibSVM time (s) Paper 1	CPU Optimized code time (s)	GPU time (s) Paper 1	LibSVM time (s) paper 2	cuSVM GPU time (s)
Adult	61.307	7.476	0.575	31.7	0.6
Web	106.835	15.733	1.063	58.8	2.7
MNIST	269.880	9.522	1.951	764.5	8.4
USPS	0.777	0.229	0.00958		
Face	88.835	5.191	0.705		
Forest				278.4	5.5

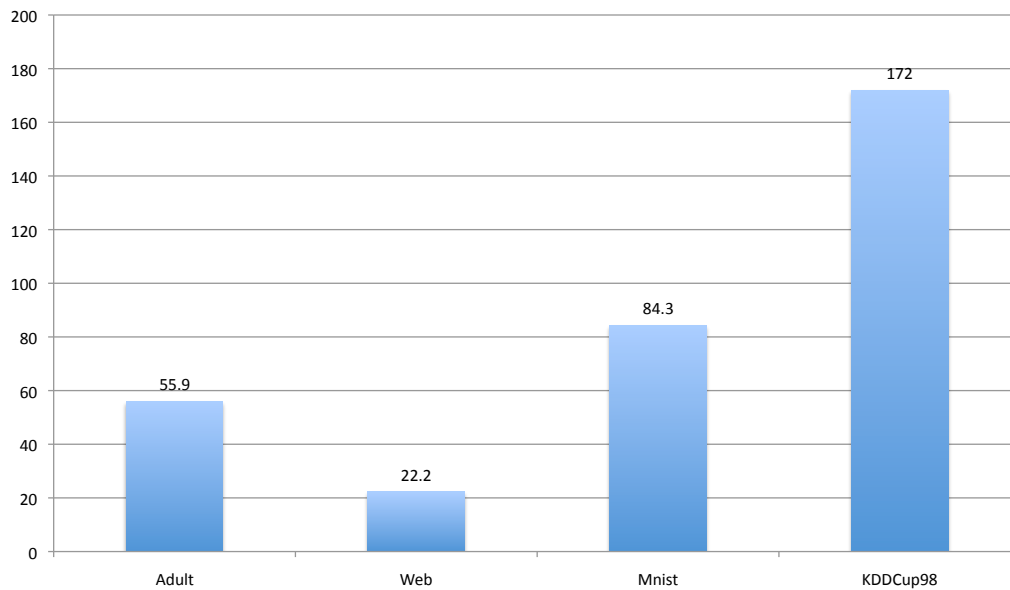
25

# Results (paper 1 C-SVC)



26

## Results (cuSVM – eSVR)



27

## Results

- 60000 input data set with 784 dimensions:  
<10 min. of GPU vs almost 5h on CPU
- Scanning images for faces: 34200 faces/sec on GPU vs 220 faces/sec on CPU

28

# Importance of memory coalescing

- In order to avoid non-coalesced memory accesses, carried both Data and Data<sup>T</sup> into GPU memory
- Letting 0.05% of memory accesses to be non-coalesced led to a 21% drop in performance for one case

29

## Machine Learning - Questions

- What type of classes of problems were used for SVM learning?
- Which calculation step takes the most time in SVM training?
- What technique was used for parallelization in SVM training?

30