# A GPU-Tailored Approach for Training Kernelized SVMs

Andrew Cotter, Nathan Srebro and Joseph Keshet {cotter,nati,jkeshet}@ttic.edu

## 1. Summary

We describe a method for efficiently training binary and multiclass kernelized SVMs on a Graphics Processing Unit (GPU).

- Supports both binary classification and multiclass objectives.
- Applies to a broad range of kernels, including the popular Gaussian kernel.
- Handles sparse data through the use of a novel clustering technique.
- Designed for GPU: different priorities than CPU.

We created an easy-to-use, freely-available library.

- Supports C, Matlab and command-line interfaces.
- Orders of magnitude faster than CPU optimizers.
- Up to several times faster than previous GPU SVM optimizers.

## 2. SVM Objective

Let $x_i$ be a list of training vectors, with associated labels $y_i$. Let $K(x,x')$ be a kernel function. The binary SVM objective is:

$$\underset{\alpha\in\mathbb{R}^n,\, b\in\mathbb{R}}{\text{minimize}} : \frac{1}{2}\alpha^T Q\alpha + C\sum_{i=1}^{n} \max\left(0, 1 - y_i\left(b + c_i\right)\right)$$

where:

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

and the vector of "responses" $c$ is:

$$c_i = \sum_{j=1}^{n} \alpha_j y_j K(x_i, x_j)$$

Most kernel SVM solvers work in the dual:

$$\begin{aligned}\underset{\alpha\in\mathbb{R}^n}{\text{maximize}} &: \mathbf{1}^T\alpha - \frac{1}{2}\alpha^T Q\alpha \\ \text{subject to} &: \forall i\left(0 \le \alpha_i \le C\right) \\ &: \sum_{i=1}^{n} y_i\alpha_i = 0\end{aligned}$$

A prototypical dual-decomposition kernel SVM optimizer will repeatedly perform the following steps:

1. Choose a "working set" of $k$ training points.
2. Find the dual-optimal values of $\alpha_i$ in the working set, while holding all other $\alpha_i$s fixed.

Performing many inexpensive updates generally leads to faster convergence than performing fewer expensive ones. Hence, the best-performing CPU-based solvers tend to use very small working sets--in the case of SMO (Platt, 1998) and LIBSVM (Chang and Lin, 2001), the working set size is 2, while SVM-Light (Joachims, 1998) defaults to 10.

The GPU SVM optimizer of Catanzaro et al. (2008) follows the lead of SMO and LIBSVM, and uses size-2 working sets.

## 3. GPU Challenges

Optimization on the GPU has different trade-offs than on the CPU:

- The dominant performance consideration, by far, is main memory accesses, not computation.
- A key issue is making use of "coalesced" memory accesses: accesses to adjacent addresses by adjacent threads can be "coalesced" into a single access.

Reducing the number of memory accesses, and making maximal use of coalescing, are of the highest importance. Basing the design of a GPU SVM optimizer on these concerns leads to different choices than are favored by CPU optimizers.

## 4. Algorithm Overview

Our algorithm works by repeatedly performing the following steps:

1. On the GPU, heuristically choose a working set.
2. On the CPU, optimize the dual objective restricted to the working set.
3. On the GPU, update the responses $c_i$ based on the changed dual variables.

Step 3 is the most expensive step, requiring *nk* kernel evaluations, where *k* is the size of the working set.

- Smaller *k* gives more "bang for the buck"--more progress is made per working set element.
- Larger *k* gives fewer memory accesses per element: step 3 requires one pass over the training data.
- If *k* is too large, we can't fully exploit the GPU's cache-like "shared memory".

We found that size-16 working sets give good performance, balancing the above considerations better than the smaller sizes preferred on the CPU.

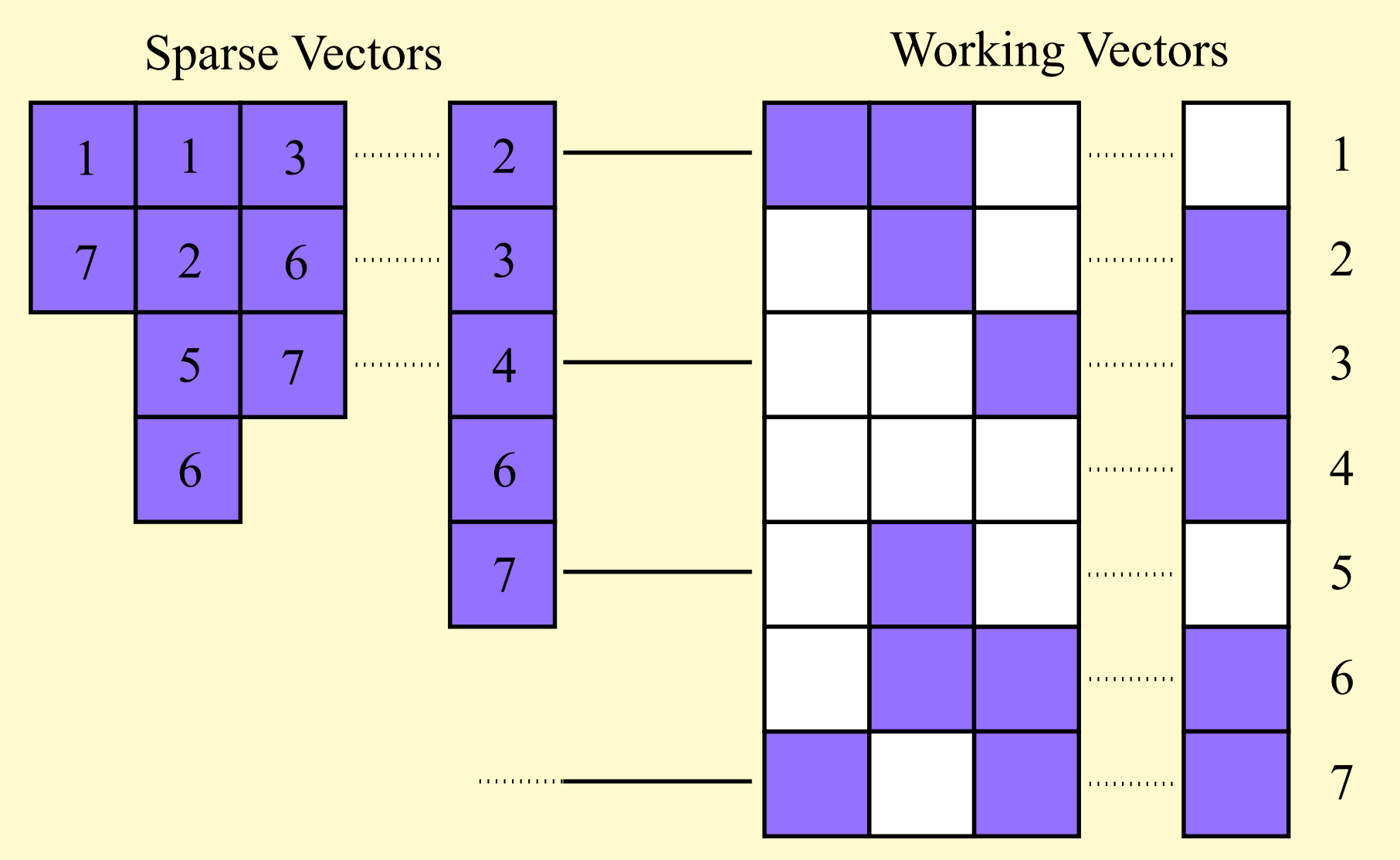
## 5. Heuristics

Dual-decomposition based SVM optimizers choose a working set heuristically.

- First-order heuristics are based on the first derivatives of the dual objective (easily derived from the responses $c_i$).
- Second-order heuristics are based on the first and second derivatives.
- The best CPU optimizers use second-order heuristics (Fan et al., 2005), which are easily formulated for size-2 working sets.
- For larger working sets, only first-order heuristics seem to be practical.

We have found that, on the GPU, the benefit of using larger working sets outweighs that of using a second-order heuristic.

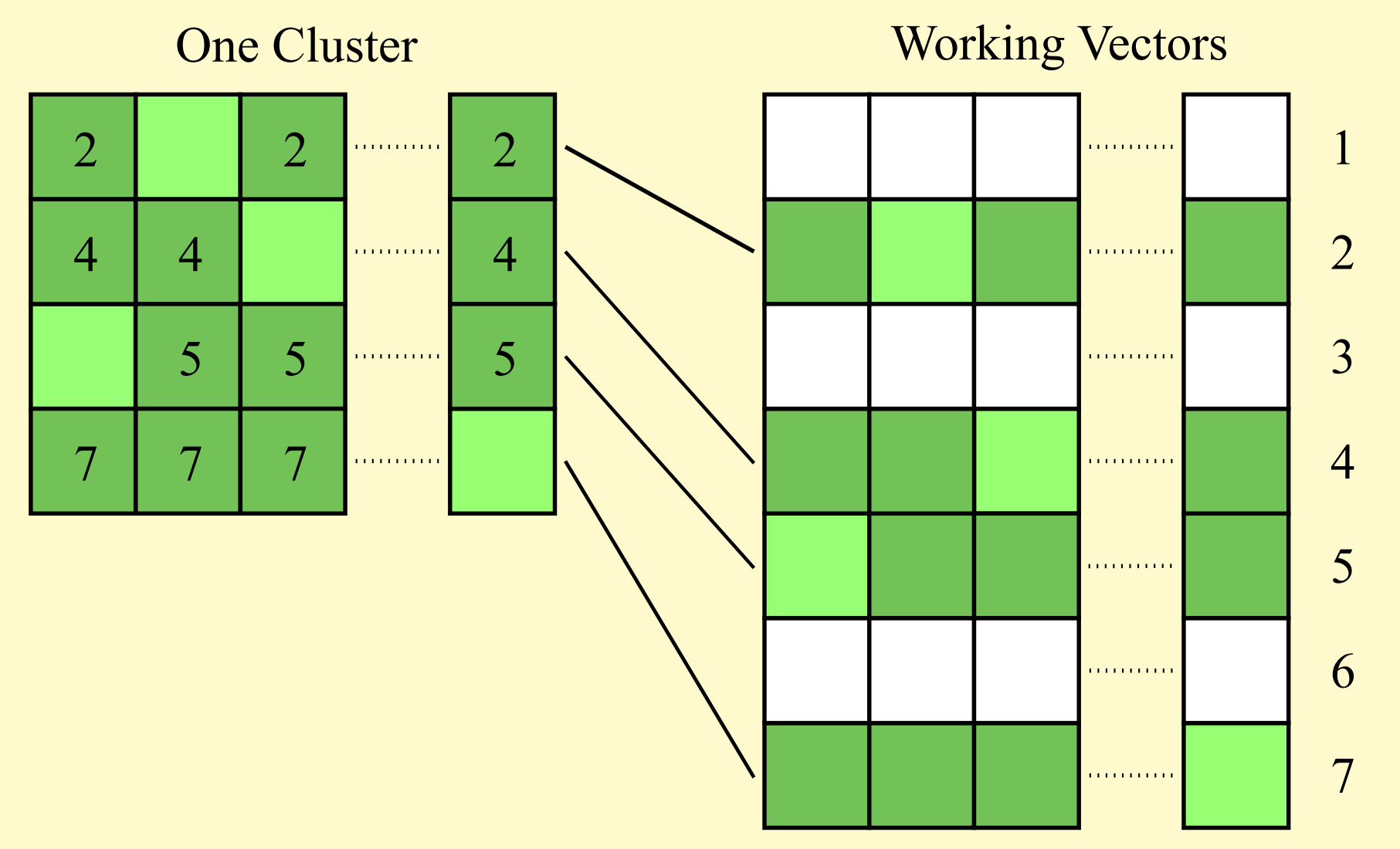
## 6. The Problem of Sparsity



When calculating kernel inner products between a set of sparse training vectors and dense working set vectors, adjacent threads will not access adjacent memory locations.

As a result, memory cannot be accessed using coalesced reads, dramatically hurting performance. This problem would only be worsened if the working set vectors, like the training vectors, were sparse.

Due to the difficulty of accessing sparse vectors in a coalesced manner, previous GPU SVM optimizers have treated all input vectors as dense.



Our solution is to cluster the data by sparsity pattern, using a simple greedy heuristic.

As illustrated on the left, when adjacent threads work on chunks of vectors with the same sparsity pattern, they only access adjacent memory locations, permitting memory accesses to be coalesced.

This approach is motivated by the observation that on many sparse machine learning datasets, certain features will be more common than others, and certain features will tend to co-occur, resulting in distinct training vectors often having similar sparsity patterns.

## 7. Experimental Setup

| Data set | Training size | Testing size | Dimension | Avg. Nonzeros | Classes |
|---|---|---|---|---|---|
| Adult | 31562 | 16282 | 123 | 13.9 | 2 |
| COV1 | 522911 | 58101 | 54 | 12.0 | 2 |
| MNIST | 60000 | 10000 | 768 | 150 | 10 |
| TIMIT | 63881 | 22257 | 39 | 39 | 6 |

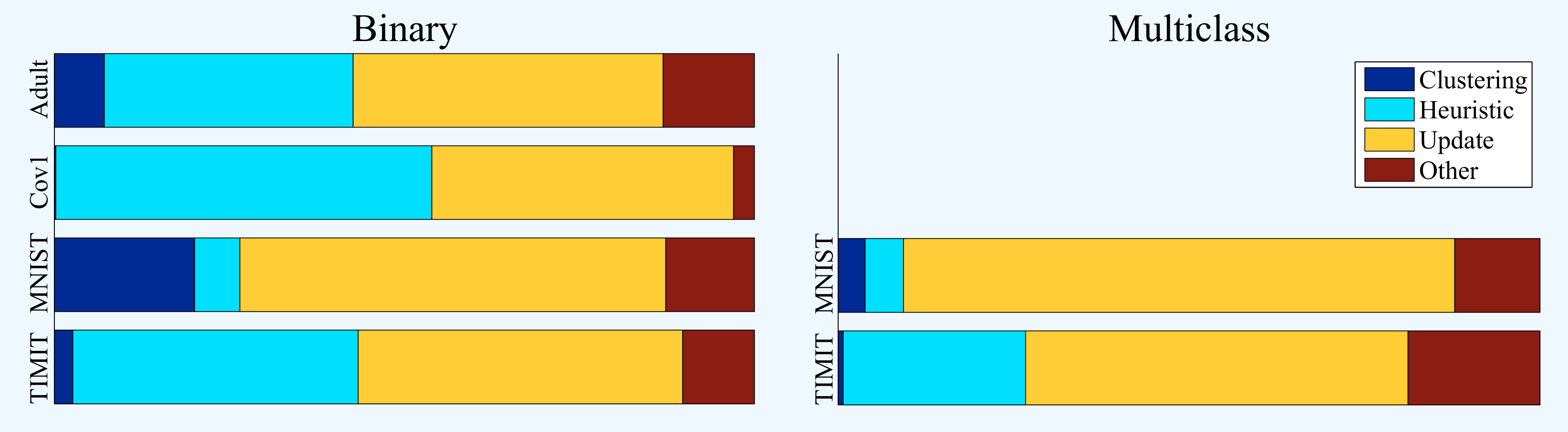
We compared our implementation against the following:

- LIBSVM (Chang & Lin, 2001), a CPU implementation using a second-order heuristic.
- GPUSVM (Catanzaro et al. 2008), a LIBSVM-like GPU implementation.
- Crammer & Singer (2001), a CPU multiclass implementation using a first-order heuristic.

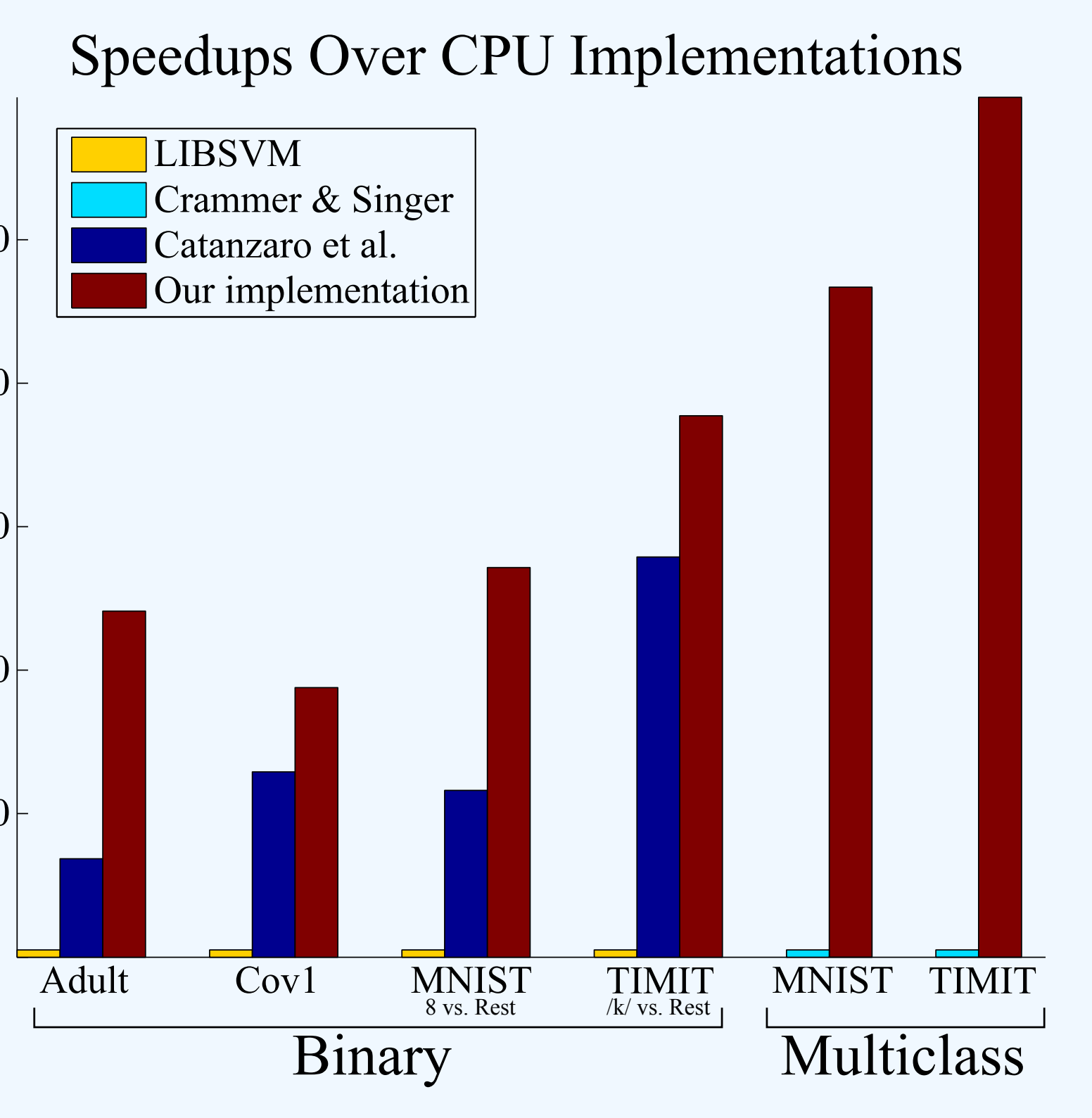
We tested using both the binary classification and multiclass (where applicable) objectives, and a Gaussian kernel. Additional details, including the regularization and Gaussian kernel parameters, may be found in our paper.

Our testing machine has an Intel Core i7 920 CPU and NVIDIA Tesla C1060 graphics card.

## 8. Runtime Breakdown



## 9. Speedups



## Availability

http://ttic.uchicago.edu/~cotter/projects/gtsvm

- Matlab, C and command-line interfaces.
- Binary and multiclass classification.
- Gaussian, polynomial and sigmoid kernels built-in.
- Supports sparse data.
- High-performance, and easy to use.

## References

- B. Catanzaro, N. Sundaram, and K. Keutzer. Fast support vector machine training and classification on graphics processors. In ICML'08, pages 104-111, 2008.
- K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. JMLR, 2:265-292, March 2002.
- C.-C. Chang and C.-J. Lin. LIBSVM: a library for support vector machines, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.
- R.-E. Fan, P.-S. Chen, and C.-J. Lin. Working set selection using second order information for training support vector machines. JMLR, 6:1889-1918, 2005.