Exploiting contemporary architectures for fast Nearest Neighbor classification

Isaac Jurado Peinado

Master CANS

January 2010

Outline

The Nearest Neighbor algorithm

- Definition of the problem
  - Given a set of samples and an element, find the closest match from the set of samples
- Both a problem and a tool
- Some applications
  - Pattern recognition
  - Statistical classification
  - Data compression
  - DNA sequencing

The Nearest Neighbor algorithm

- Definition of the problem
  - Given a set of samples and an element, find the closest match from the set of samples
- Both a problem and a tool
- Some applications
  - Statistical classification

Input data

- Database
  - M training elements
  - N testing elements
  - D features
- Training set
  - TRAIN, a $M \times D$ matrix
  - $\text{classof}(b)$, the class label of an element b (a row in matrix TRAIN)
- Testing set
  - TEST, a $N \times D$ matrix
  - $\text{classof}(a)$, same as with TRAIN

Analytical model

- Performance measured in Normalized Cycles

$$NC = \frac{\mathrm{CPU\_time\_in\_cycles}}{N \cdot M \cdot D}$$

- NC modeled as

$$NC = NC_{cpu} + NC_{mem}$$

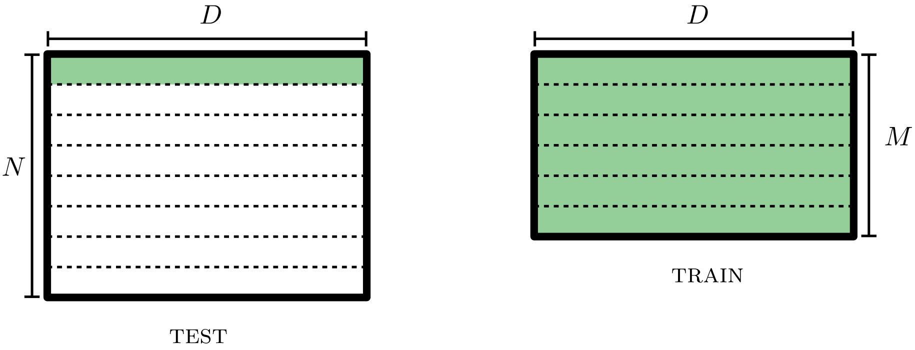
Hardware platform

- Intel Xeon E5520 “Gainestown” (*Nehalem* microarchitecture)
- Two quad-core processors
- Two 4GB memory modules
- NUMA, Quick Path Interconnect
- 256kB per core L2 cache
- 8MB per processor L3 cache
- SSE 4.2

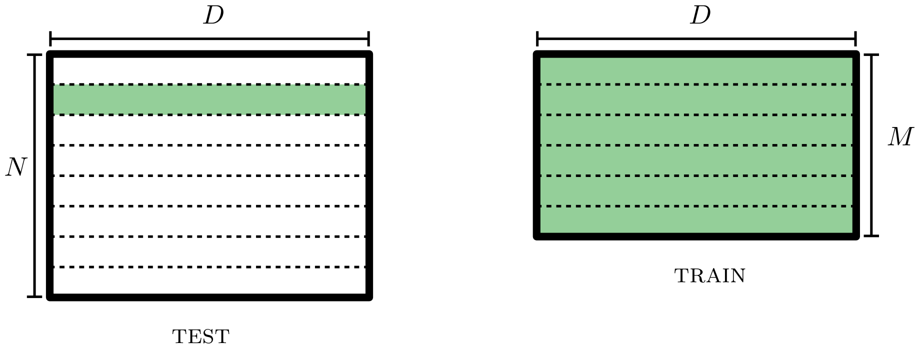
Basic algorithm

for all $a \in \text{TEST}$ **do**
  $min \leftarrow \infty$
  for all $b \in \text{TRAIN}$ **do**
    $dist \leftarrow 0$
    for $i = 0$ **to** D **do**
      $dist \leftarrow dist + (a_i - b_i)^2$
    end for
    if $dist < min$ **then**
      $min \leftarrow dist$
      $cls \leftarrow \mathrm{classof}(b)$
    end if
  end for
  $\mathrm{classof}(a) \leftarrow cls$
end for

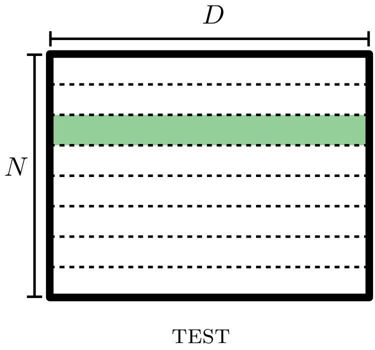
Memory access pattern

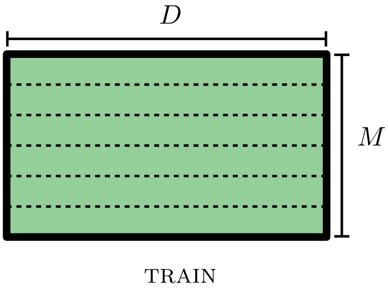


Memory access pattern

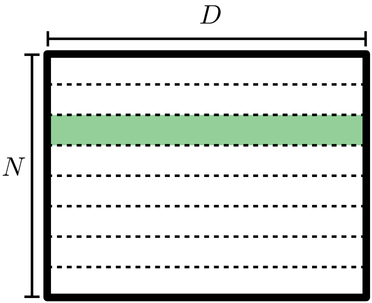


Memory access pattern

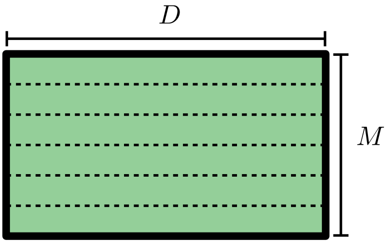




Memory access pattern



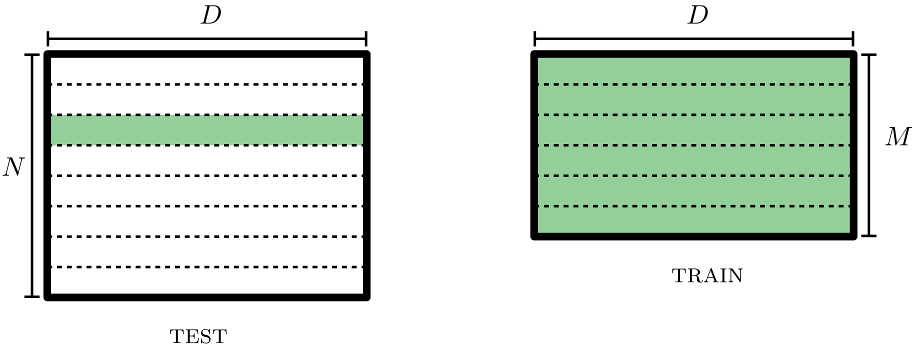
TEST



TRAIN

- D influences NC_{cpu}

Memory access pattern



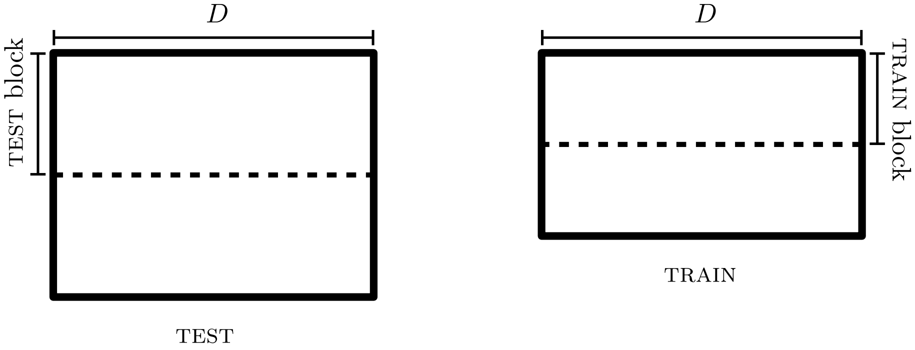
- D influences NC_{cpu}
- M and N influence NC_{mem}

Outline

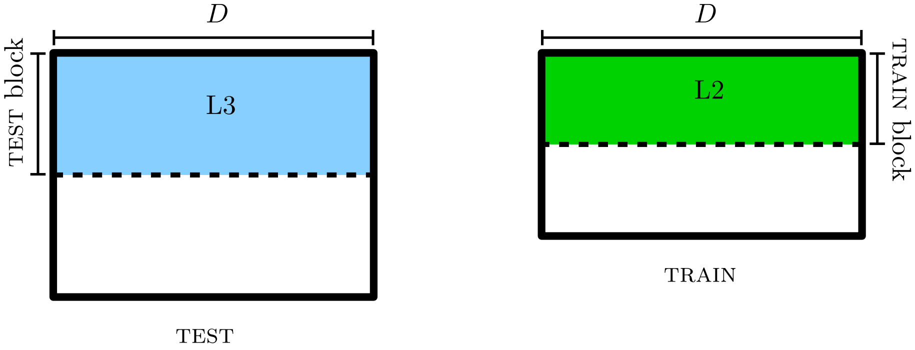
Blocking

for all $a \in \text{TEST}$ **do**
  $min \leftarrow \infty$
  for all $b \in \text{TRAIN}$ **do**
    $dist \leftarrow 0$
    for $i = 0$ **to** D **do**
      $dist \leftarrow dist + (a_i - b_i)^2$
    end for
    if $dist < min$ **then**
      $min \leftarrow dist$
      $cls \leftarrow \text{classof}(b)$
    end if
  end for
  $\text{classof}(a) \leftarrow cls$
end for

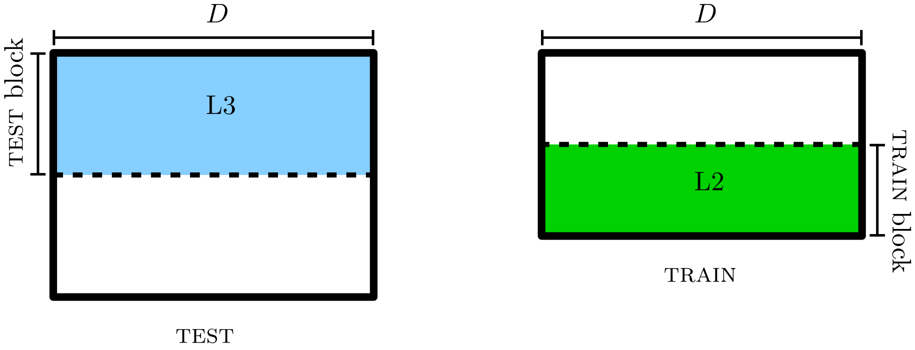
Blocking scheme



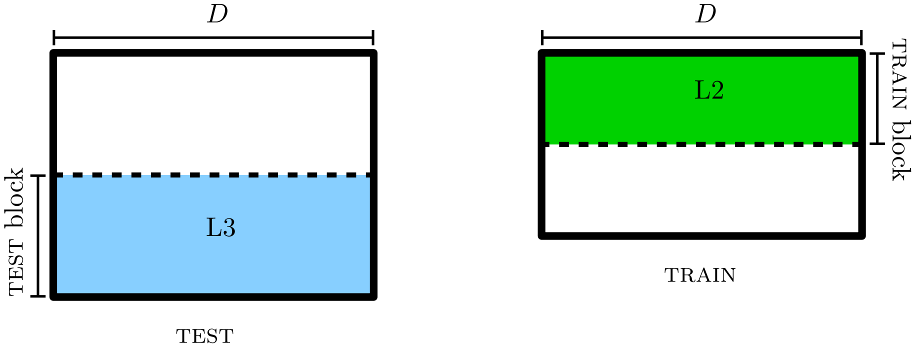
Blocking scheme



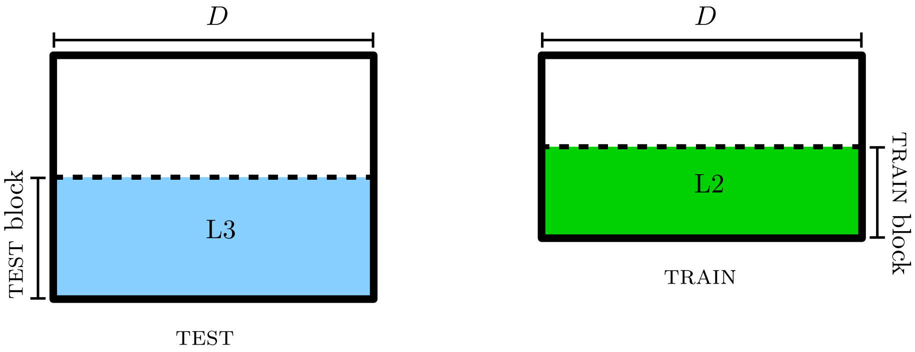
Blocking scheme



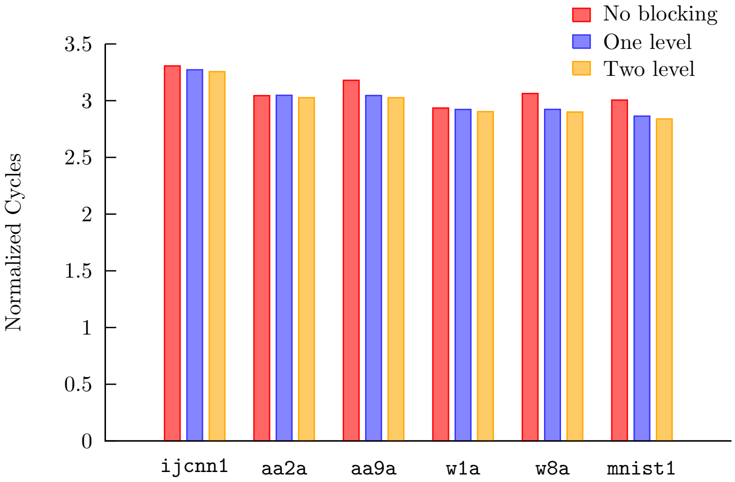
Blocking scheme



Blocking scheme



Blocking results



NC_{mem} reduction

Algorithm	NC(Small)	NC(Large)
No blocking	3.03	3.14
One level blocking	3.03	3.04
Two level blocking	3.03	3.03

- Stalls due to cache misses are reduced

NC_{mem} reduction

Algorithm	NC(Small)	NC(Large)
No blocking	3.03	3.14
One level blocking	3.03	3.04
Two level blocking	3.03	3.03

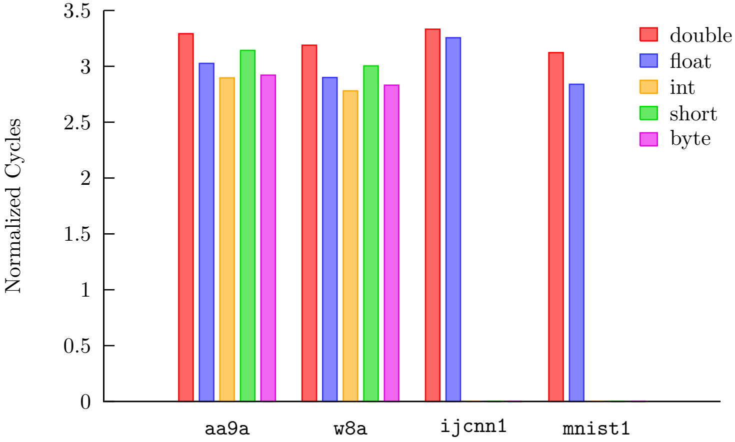
- Stalls due to cache misses are reduced

$$NC = NC_{cpu} + NC_{mem}$$

$$NC_{mem} \approx 0$$

$$NC \approx NC_{cpu}$$

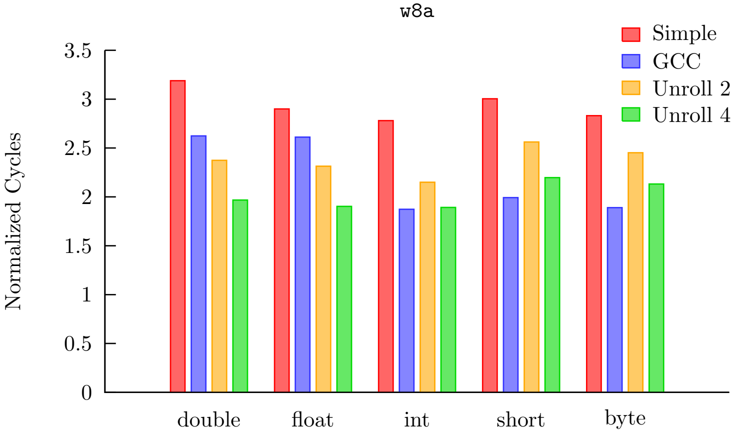
Changing data type



Loop unrolling

for all $a \in \text{TEST}$ **do**
    $min \leftarrow \infty$
    for all $b \in \text{TRAIN}$ **do**
        $dist \leftarrow 0$
        for $i = 0$ **to** D **do**
            $dist \leftarrow dist + (a_i - b_i)^2$
        end for
        if $dist < min$ **then**
            $min \leftarrow dist$
            $cls \leftarrow \mathrm{classof}(b)$
        end if
    end for
    $\mathrm{classof}(a) \leftarrow cls$
end for

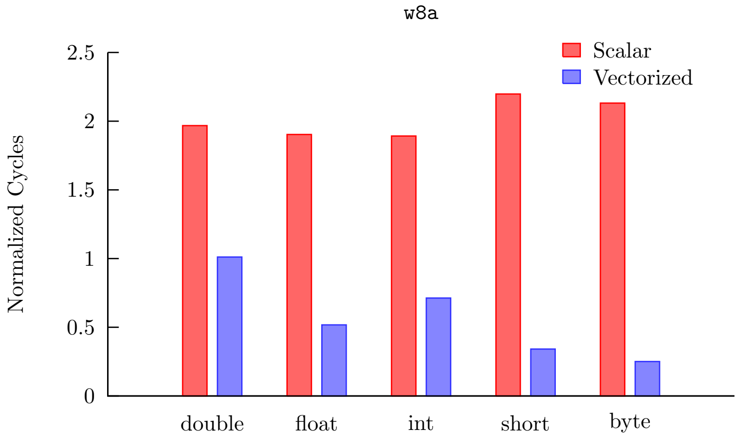
Unrolling results



Vectorization

for all $a \in \text{TEST}$ **do**
  $min \leftarrow \infty$
  for all $b \in \text{TRAIN}$ **do**
    $dist \leftarrow 0$
    for $i = 0$ **to** D **do**
      $dist \leftarrow dist + (a_i - b_i)^2$
    end for
    if $dist < min$ **then**
      $min \leftarrow dist$
      $cls \leftarrow \text{classof}(b)$
    end if
  end for
  $\text{classof}(a) \leftarrow cls$
end for

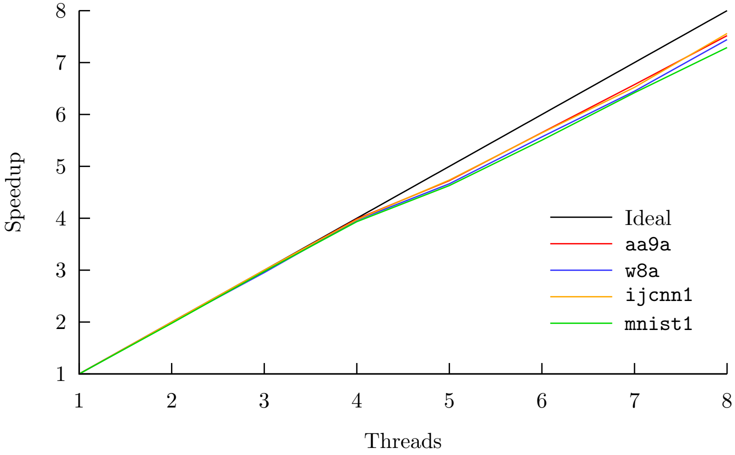
Vectorization results



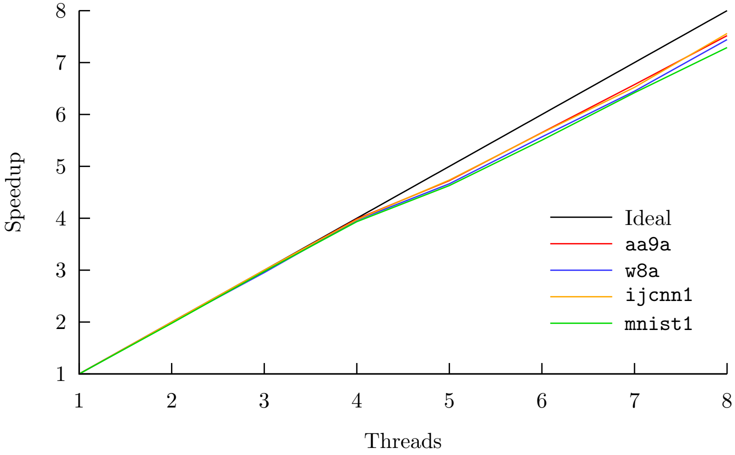
Parallelization

for all $a \in \text{TEST}$ **do**
  $min \leftarrow \infty$
  for all $b \in \text{TRAIN}$ **do**
    $dist \leftarrow 0$
    for $i = 0$ **to** D **do**
      $dist \leftarrow dist + (a_i - b_i)^2$
    end for
    if $dist < min$ **then**
      $min \leftarrow dist$
      $cls \leftarrow \text{classof}(b)$
    end if
  end for
  $\text{classof}(a) \leftarrow cls$
end for

Scalability results

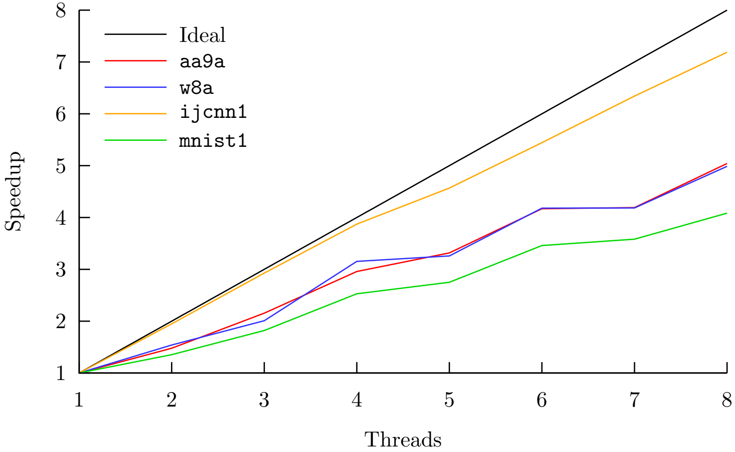


Scalability results



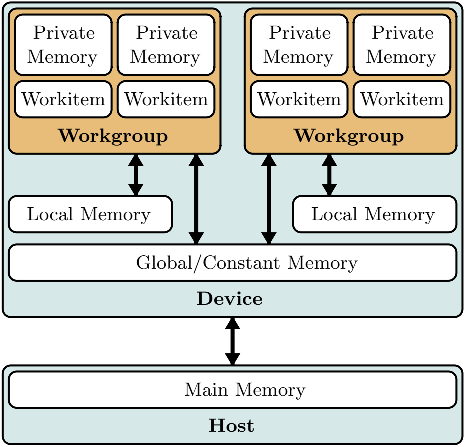
- Good scalability thanks to blocking

Scalability without blocking



Outline

OpenCL overview

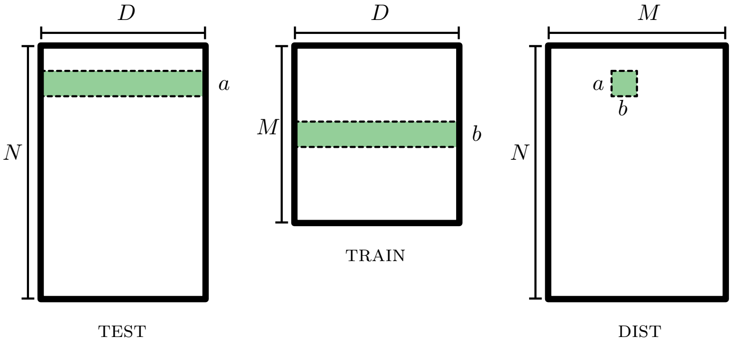


Algorithm adaptation

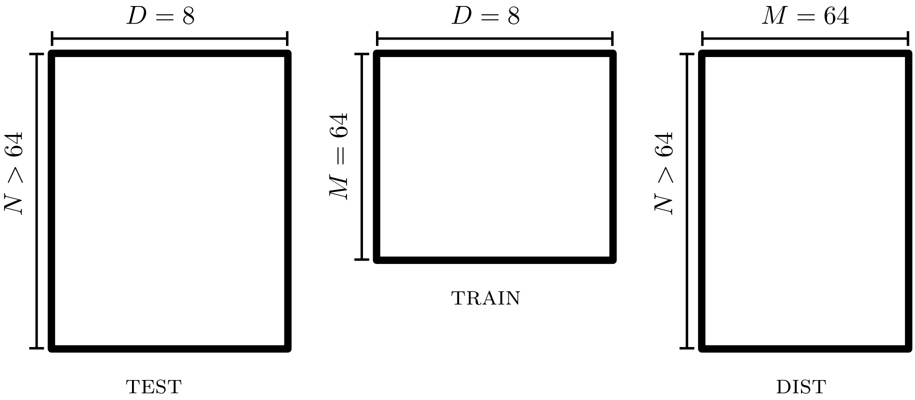
- Inspired by `sgemmnt` from CUBLAS 2.0

Algorithm adaptation

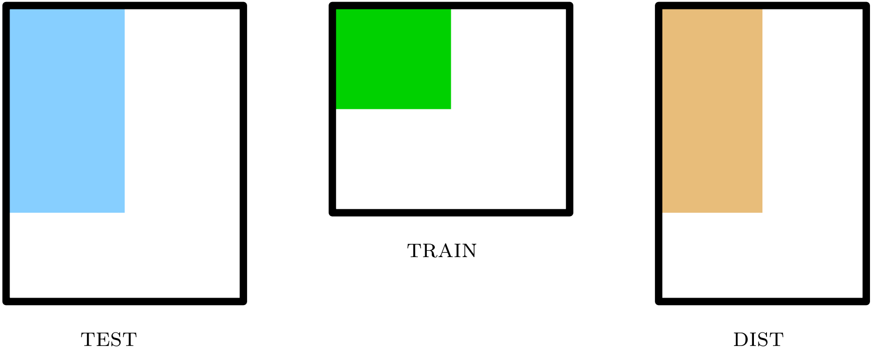
- Inspired by `sgemmnt` from CUBLAS 2.0



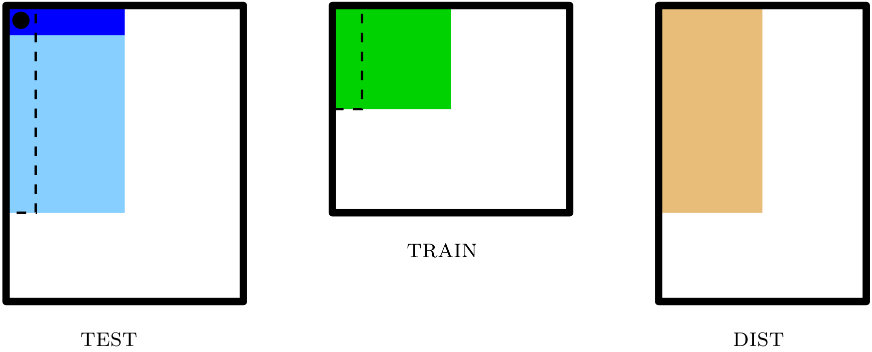
GPU memory access pattern



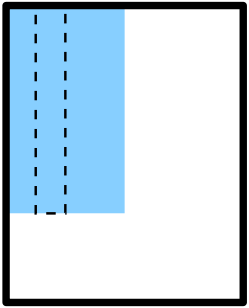
GPU memory access pattern



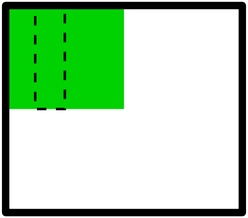
GPU memory access pattern



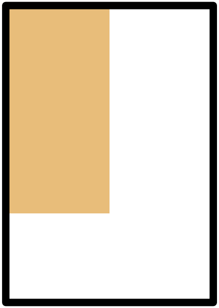
GPU memory access pattern



TEST

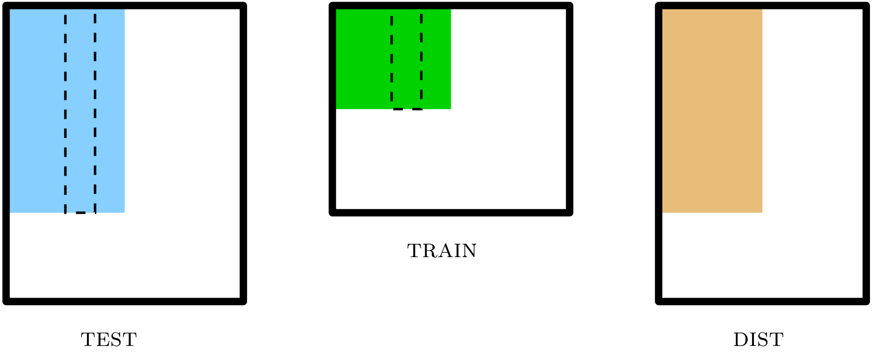


TRAIN

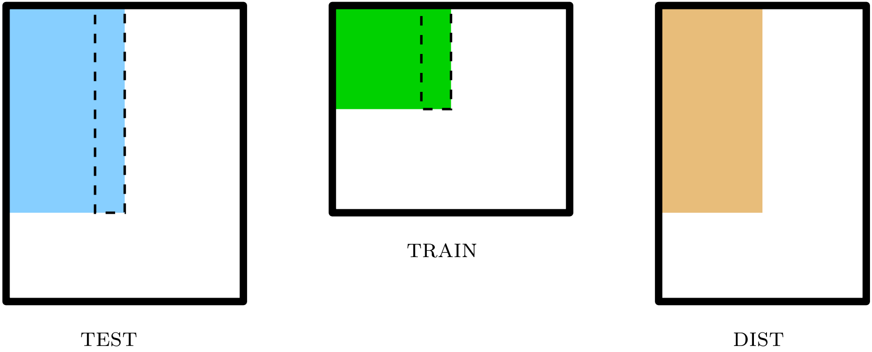


DIST

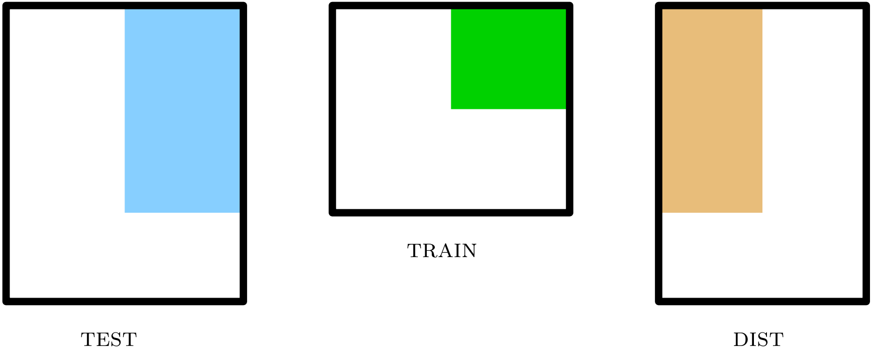
GPU memory access pattern



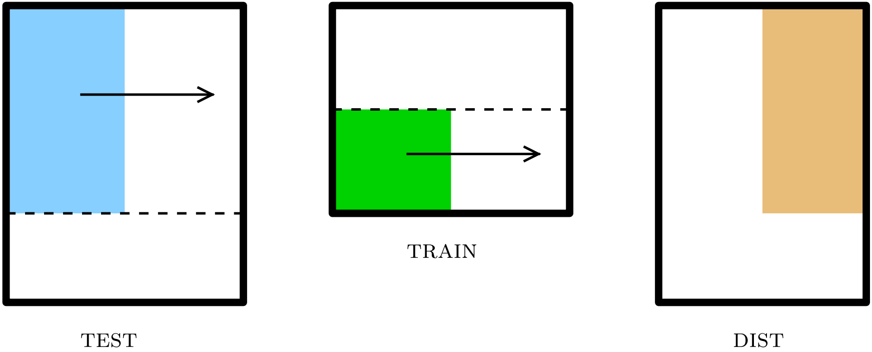
GPU memory access pattern



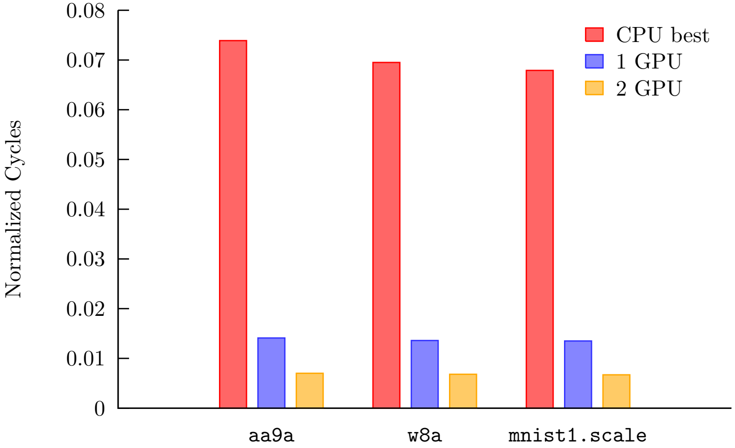
GPU memory access pattern



GPU memory access pattern



GPU performance



Outline

1. Introduction
2. CPU optimizations
3. GPU port
4. Other classification methods
   - Overview
   - Classification accuracy
   - Performance
5. Conclusions

Overview

- k-Nearest Neighbors
- Support Vector Machines
  - Originally conceived for two class problems
  - LibSVM and LibLINEAR

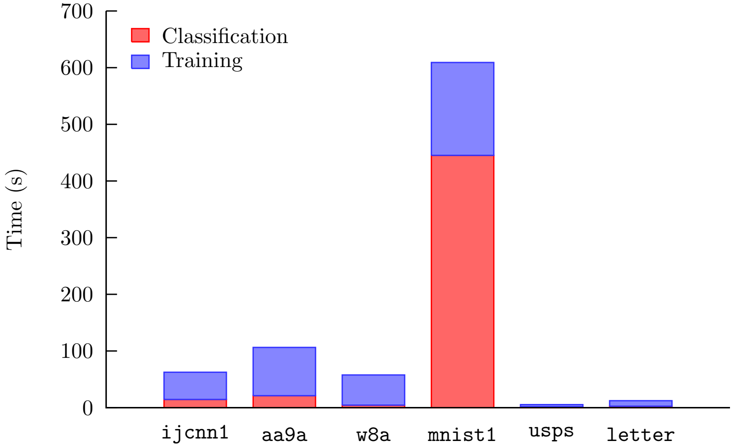
Overview

- k-Nearest Neighbors
- Support Vector Machines
  - Originally conceived for two class problems
  - LibSVM and LibLINEAR
  - Classification procedure:
    1. Find model parameters or cross-validation (optional)
    2. Generate a model from the training set
    3. Use the model to classify elements from the testing set

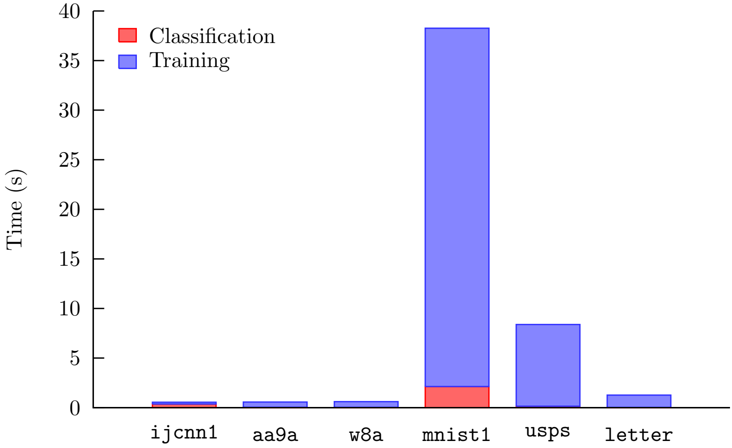
Accuracy comparison

Database	1-NN	3-NN	13-NN	23-NN	LibSVM	LibLINEAR
ijcnn1	97.39%	97.09%	95.63%	94.71%	97.82%	91.79%
aa9a	79.51%	81.73%	83.69%	84.09%	85.03%	84.96%
w8a	97.93%	98.74%	98.36%	94.48%	99.18%	90.54%
mnist1	95.71%	95.93%	95.24%	98.15%	97.70%	90.07%
svmguide3	100%	70.73%	43.90%	43.90%	82.93%	24.39%

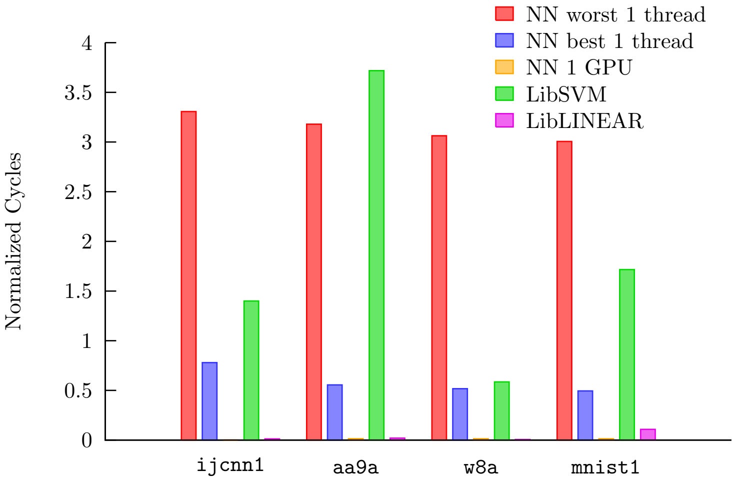
LibSVM times



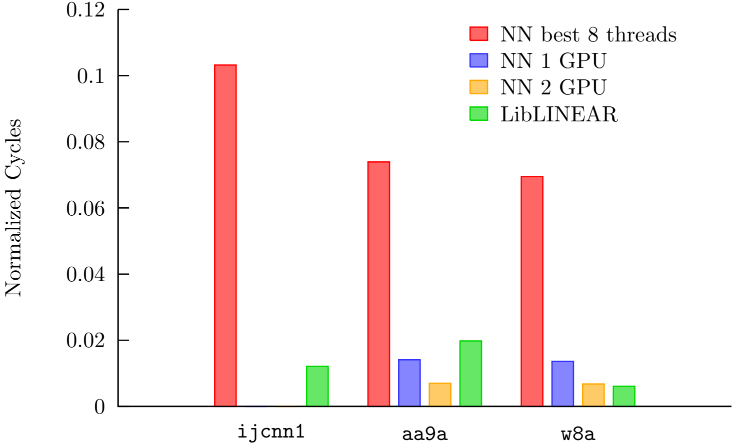
LibLINEAR times



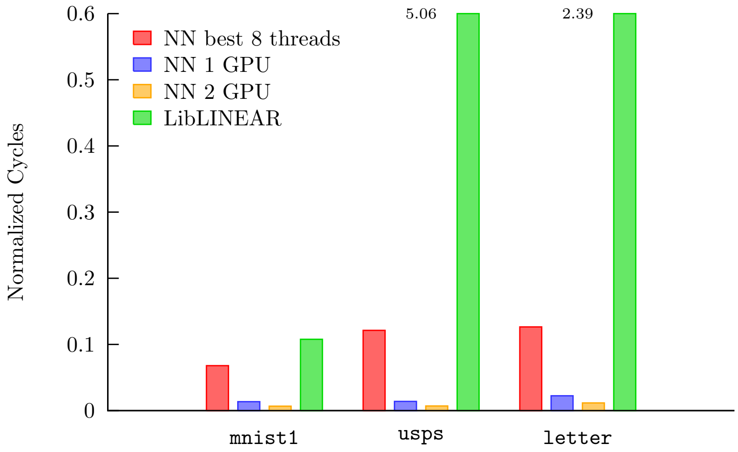
Performance comparison (I)



Performance comparison (II)



Performance comparison (III)



Outline

Conclusions

- Simpler algorithms are easier to optimize
- Parallelism can be exploited in several granularities
- Controlling memory access patterns brings more optimization opportunities
  - Specially when programming with GPGPUs
- There is no single best classification solution
- Performance of optimized NN rivals that of state of the art classifiers

Future work

- Overcome current limitations
  - Support out of core operation
- Compare against other nearest neighbor search algorithms
  - Locality sensitive hashing
  - Space partitioning structures
- Extend the GPU implementation to k-NN
- Find techniques to shrink the training set (i.e. TRAIN matrix)

END