OPENMP IN CLASSIFIER DESIGN

Boguslaw Cyganek, Michal Wozniak

AGH University of Science and Technology
Cracow, Poland

OpenMPCon – DEVELOPERS CONFERENCE
3-5 October 2016, Nara, Japan
An overview:

- Data classification – new trends.
- Classifier design for parallel run.
- Implementation and practical issues.
- Case study (tensor classifier)
- Conclusions.
Classifiers are algorithms that, given features, respond class of unknown objects.

These can be **supervised**, i.e. trained from known examples, or **unsupervised** which can discover some regularities in data.

Their examples are ample, from spam filters, text search engines, up to face recognition, car security systems with road signs recognition, driver sleepiness alert, unmanned vehicle maneuvering, and many more.
Main parameters of classifiers are **accuracy** and **operation speed**. In this talk we concentrate on the latter and provide some design patterns and recipes on parallel implementations of selected classifiers using OpenMP.

Our main application domain are classifiers used in **computer vision** and **streams of big data**, though the presented methods can be easily exploited in other branches of classification problems.
Data classification
- Intro and latest breakthroughs
Classification of multi-dimensional data
Classification of multi-dimensional data

Classification of multi-dimensional data

Fisher's Iris Data

<table>
<thead>
<tr>
<th>Sepal length</th>
<th>Sepal width</th>
<th>Petal length</th>
<th>Petal width</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
<td><em>I. setosa</em></td>
</tr>
<tr>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>0.2</td>
<td><em>I. setosa</em></td>
</tr>
<tr>
<td>4.7</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
<td><em>I. setosa</em></td>
</tr>
<tr>
<td>4.6</td>
<td>3.1</td>
<td>1.5</td>
<td>0.2</td>
<td><em>I. setosa</em></td>
</tr>
<tr>
<td>5.0</td>
<td>3.6</td>
<td>1.4</td>
<td>0.2</td>
<td><em>I. setosa</em></td>
</tr>
<tr>
<td>5.4</td>
<td>3.9</td>
<td>1.7</td>
<td>0.4</td>
<td><em>I. setosa</em></td>
</tr>
<tr>
<td>4.6</td>
<td>3.4</td>
<td>1.4</td>
<td>0.3</td>
<td><em>I. setosa</em></td>
</tr>
<tr>
<td>5.0</td>
<td>3.4</td>
<td>1.5</td>
<td>0.2</td>
<td><em>I. setosa</em></td>
</tr>
<tr>
<td>4.4</td>
<td>2.9</td>
<td>1.4</td>
<td>0.2</td>
<td><em>I. setosa</em></td>
</tr>
<tr>
<td>4.9</td>
<td>3.1</td>
<td>1.5</td>
<td>0.1</td>
<td><em>I. setosa</em></td>
</tr>
<tr>
<td>5.4</td>
<td>3.7</td>
<td>1.5</td>
<td>0.2</td>
<td><em>I. setosa</em></td>
</tr>
</tbody>
</table>

We need an expert for feature engineering ...

Classification of multi-dimensional data

Feature vector

CLASSIFIER
Classifiers – main types, and latest breakthroughs

Let’s name the most popular ones and their basic properties:

- k nearest neighbor (the simplest idea)
- Bayes classifiers (need probability estimations)
- Neural networks (different types)
- Support vector machines (max. Separation)
- Decision trees (intuitive rules)
- Subspace projections (PCA, Fisher, canonical correlation, etc.)
- Multi-linear algebra
- Adaptive boost (AdaBoost)
Classifiers – main types, and latest breakthroughs

Let’s name the most popular ones and their basic properties:

- k nearest neighbor (the simplest idea)
- Bayes classifiers (need probability estimations)
- Neural networks (different types) ➔ deep architectures (CNN)
- Support vector machines (max. Separation)
- Decision trees (intuitive rules)
- Subspace projections (PCA, Fisher, canonical correlation, etc.)
- Multi-linear algebra
- Adaptive boost (AdaBoost)
Classifiers – main types, and latest breakthroughs

Let’s name the most popular ones and their basic properties:

- k nearest neighbor (the simplest idea)
- Bayes classifiers (need probability estimations)
- Neural networks (different types) ➔ deep architectures (CNN)
- Support vector machines (max. Separation) ➔ kernel methods
- Decision trees (intuitive rules)
- Subspace projections (PCA, Fisher, canonical correlation, etc.)
- Multi-linear algebra
- Adaptive boost (AdaBoost)
Classifiers – main types, and latest breakthroughs

Let’s name the most popular ones and their basic properties:

- k nearest neighbor (the simplest idea)
- Bayes classifiers (need probability estimations)
- Neural networks (different types) → deep architectures (CNN)
- Support vector machines (max. Separation) → kernel methods
- Decision trees (intuitive rules)
- Subspace projections (PCA, Fisher, canonical correlation, etc.)
- Multi-linear algebra → tensor methods
- Adaptive boost (AdaBoost)
Classifiers – main types, and latest breakthroughs

Let’s name the most popular ones and their basic properties:

- k nearest neighbor (the simplest idea)
- Bayes classifiers (need probability estimations)
- Neural networks (different types) → deep architectures (CNN)
- Support vector machines (max. Separation) → kernel methods
- Decision trees (intuitive rules)
- Subspace projections (PCA, Fisher, canonical correlation, etc.)
- Multi-linear algebra → tensor methods
- Adaptive boost (AdaBoost) → classifier ENSEMBLES (here we go „parallel” as well)
Classifiers – literature I like


Classifiers – implementations / performance issues

• High accuracy requirement !!
• Massive data
• Massive data coming in streams (concept drift)
• Real-time answer requirement
• Training / re-training time requirement
• Mobile / embedded platform requirement
• …
• Choosing proper classifier to the task (no free lunch theorem)
• Choosing proper architecture (know your platform – do I need to learn assembly?)
• Choosing the right **TOOLS**

• Ready library (?) ok, but not always …
• **Go parallel !!**
Algorithm design for parallelism

- **Use domain parallelism** – analyse your domain for parallelism (pattern recognition)
- **Increase thread parallelism** – think of data sharing and high degree of independent tasks
- **Exploit data parallelism** – split data and apply the same algorithm
- **Improve data locality** – arrange the algorithms to minimize data movements, keep data compact, try to fit data into cache (can be forced by a programmer)

Know your hardware! (nodes, cores/threads, memory organization, Xeon, GPU, etc.)

But think of a generic platform (unless a special purpose tuned system is created)
Algorithm design for parallelism

- Designing for parallel implementation
- Think of a "task" rather than of a "thread", think of a parallel "data flow"
- Design work-flow

Factoring software for parallelism – decomposing into levels of parallelism but with data locality in mind.

Barriers and other synchronization mechanisms are the enemy of scaling (i.e. employing more cores, threads, vectors, etc.)
Classifiers – implementations / performance issues

- Parallel software – threads?
- GPU (CUDA, OpenCL)?
- FPGA?
- Hybrid solution?

- Create threads by yourself
- Use MPI
- Use TBB
- Use OpenMP
Classifiers – using OpenMP

- k nearest neighbor (the simplest idea)
- Bayes classifiers (need probability estimations)
- Neural networks (different types) ➔ deep architectures (CNN)
- Support vector machines (max. Separation) ➔ kernel methods
- Decision trees (intuitive rules)
- Subspace projections (PCA, Fisher, canonical correlation, etc.)
- Multi-linear algebra ➔ tensor methods
- Adaptive boost (AdaBoost) ➔ classifier ENSEMBLES (here we go „parallel” as well)
OpenMP

Especially for task level parallelism

Pros:

- Minimally interfere with code development
- Transparent to the not supporting compilers
- Allows easy code refactoring

Cons:

- Rather for shared memory space systems
- Some problems when dealing with objects (see Cyganek & Sieberts)
- Not all features implemented on some platforms (e.g. Microsoft Visual 2015 vs. Intel)

OpenMP – New features

Nested loop parallelism

Vectorization
- #pragma omp simd  
  do single-instruction-multiple-data
- #pragma omp vector  
  enables or disables vectorization of a loop
- #pragma ivdep  
  give a hint about data dependencies

Data alignement to help in vectorization
- Data are moved efficiently if aligned on specific byte boundaries (e.g. Xeon 64-byte boundary)

What to do?
- Align the data (e.g. __declspec( align(64) ) double array[ 1024 ]);
- Explicitly inform the compiler (e.g.  
  __assume_aligned( ptr, 64 )  
  #pragma vector aligned

Use vectorization advisor (Xeon Phi, Intel)
OpenMP – Would be nice…

OpenMP 4.0 standard includes support of accelerators (GPU, DSP, Xeon Phi, and so on)

OpenACC a similar idea to OpenMP and (potentially) easy to use. However, only few compilers (PGI, etc.).

Still lacking the tools on some platforms (porting workhours problem).


S. Antao (2014). LLVM Support for OpenMP 4.0 Target Regions on GPUs Supercomputing
Design of Parallel Architectures of Classifiers Suitable for Intensive Data Processing
Design of Parallel Architectures of Classifiers

At first - two of the most popular architectures of ensembles of classifiers: the serial and the parallel ones.
Design of Parallel Architectures of Classifiers

At first - two of the most popular architectures of ensembles of classifiers: the serial and the parallel ones.

An example of such a system is the face detection method by Viola and Jones. Training is done with the AdaBoost which amplifies response on poorly classified examples. Such strategy imposes data decomposition into sets of usually decreasing number of elements.

Data processing in a serial chain of classifiers is effective if member classifiers are able to operate in a pipeline mode. One of the requirements in this case is that each classifier in the chain consumes the same time quant for data processing. The penalty of using a cascade is a delay necessary to fill up the processing line which is proportional to the number of used classifiers. However, in practice these requirements are not easy to fulfill.


At first - **two** of the most popular architectures of ensembles of classifiers: the serial and the **parallel** ones.
Design of Parallel Architectures of Classifiers

At first - two of the most popular architectures of ensembles of classifiers: the serial and the parallel ones.

In this case all classifiers are assumed to operate independently which is a big advantage considering implementation and execution time. However, all partial responses need to be synchronized and collected by the answer fusion module which outputs a final response. There are different methods of training of the member classifiers $C_i$.

Some of the most popular are data bagging and data clustering.

There are many examples of the parallel classifier systems organized as ENSEMBLES.

In this talk I’ll present the tensor based classifiers (HOSVD) trained with different data partitions obtained with data bagging.
Design of Parallel Architectures of Classifiers

Data Splitters

The role of a data splitter is to arrange the training process in order to obtain the best accuracy of the ensemble. The two tested methods are as follows:

- **Bagging** - consists of creating a number of data sets $D_i$ from the training set $D$ with a uniform data sampling with replacement. As shown by Grandvalet, bagging reduces variance of a classifier and improves its generalization properties. Each set $D_i$ is used to train a separate member of the ensemble, which contains less data than $D$. Thanks to this data decomposition a better accuracy can be obtained due to a higher diversity. Also, the problem of processing massive data can be greatly reduced. It is also possible to extend the ensemble with a new classifier if new training data are available at a later time.


Design of Parallel Architectures of Classifiers

Data Splitters

The role of a data splitter is to arrange the training process in order to obtain the best accuracy of the ensemble. The two tested methods are as follows:

- **Data clustering** - consists in usually unsupervised partitioning of the input dataset $D$ into typically disjoint sets $D_i$. In our previous systems the k-means as well as their fuzzy and kernel versions were used. In this case a first step is the choice of data centers. Then data distances to each center are computed and the points are assigned to their nearest centers. After that, positions of the centers are recomputed to account for new members of that partition. The procedure follows until there are no changes in data partitioning. Similarly to bagging, splitting by clustering also allows better accuracy and data decomposition useful in parallel realizations.

**References**

L.I. Kuncheva, *Combining Pattern Classifiers. Methods and Algorithms*. Wiley Interscience, 2005


Design of Parallel Architectures of Classifiers

Selection of the Member Classifiers

Choice of the member classifiers depends on many factors, such as type and dimensionality of data. However, the classifiers need to be chosen in a way to assure the best accuracy and speed of operation, especially when processing massive vision data.

Good results were obtained in the tested systems using the tensor classifiers, as well as using the OC-SVMs.


Case study – just a taste of …
Classification of multi-dimensional data

Variety of dimensions and formats …
Classification of multi-dimensional data
Classification of multi-dimensional data
Classification of multi-dimensional data
Classification of multi-dimensional data
Classification of multi-dimensional data
Classification of multi-dimensional data

Color image is a 3D tensor
Patterns, tensors and their decompositions
Tensors for multi-dimensional data processing:

A hyperspectral image with selected regions of different reflectance properties. Series of hyperspectral images naturally form 3D tensors with two spatial dimensions \((x,y)\) and one spectral \(\lambda\).
Tensors for multi-dimensional data processing:

An example of a 3D tensor representing MRI signals
Tensors for multi-dimensional data processing:

A tensor $\mathbf{T}$ of $P$ dimensions of values $N_1, N_2, \ldots, N_P$, is denoted as follows

$$\mathbf{T} \in \mathbb{R}^{N_1 \times N_2 \times \ldots \times N_P}$$

A single element $t$ of $\mathbf{T}$ is addressed providing its precise position by a series of indices $n_1, n_2, \ldots, n_P$,

$$t_{n_1n_2\ldots n_P} \equiv \mathbf{T}_{n_1n_2\ldots n_P} \quad 1 \leq n_1 \leq N_1, \ 1 \leq n_2 \leq N_2, \ldots, \ 1 \leq n_P \leq N_P$$

in the equivalent function-like notation

$$t_{n_1n_2\ldots n_P} \equiv \mathbf{T}(n_1, n_2, \ldots, n_P)$$
Tensors for multi-dimensional data processing – an example:

A $3 \times 3 \times 3$ tensor and possible representation with three frontal 2D slices.

$$\mathbf{T} = \begin{bmatrix}
12 & 0 & 0 & 1 & -1 & 1 & 0 & 0 & -2 \\
0 & 4 & 1 & 4 & 0 & 4 & 4 & 10 & 0 \\
2 & 3 & -1 & 5 & 2 & 1 & 2 & 1 & 0
\end{bmatrix}$$
Albert Einstein in 1921

Tullio Levi-Civita

Bernhard Riemann in 1863

https://en.wikipedia.org/wiki/Albert_Einstein
The impossible world of MC Escher
The impossible world of MC Escher

Types of tensor decompositions:

- Higher Order Singular Value Decomposition (HOSVD, ~Tucker)
- Rank-1 (Canonical decomposition, CANDECOMP/PARAFAC or CP)
- Best Rank-$R_k$ (~Tucker)
- Nonnegative matrix and tensor decompositions
Types of tensor decompositions:

- **Higher Order Singular Value Decomposition** (HOSVD, ~Tucker)
- Rank-1 (Canonical decomposition, CANDECOMP/PARAFAC or CP)
- Best Rank-$R_k$ (~Tucker)
- Nonnegative matrix and tensor decompositions
More on

Higher Order Singular Value Decomposition

(HOSVD, ~Tucker)

applications come soon…
Basic concepts of the multilinear algebra – HOSVD:

A tensor: $\mathcal{T} \in \mathbb{R}^{N_1 \times N_2 \times \ldots \times N_m \times \ldots \times N_n \times \ldots \times N_P}$ can be decomposed to

$$\mathcal{T} = \mathcal{Z} \times_1 S_1 \times_2 S_2 \ldots \times_P S_P$$

$S_k$ denotes a mode matrix, which is a unitary matrix of dimensions $N_k \times N_k$ spanning the column space of the matrix $T^{(k)}$ obtained from the mode-n flattening of $T$;

$$\mathcal{Z} \in \mathbb{R}^{N_1 \times N_2 \times \ldots \times N_m \times \ldots \times N_n \times \ldots \times N_P}$$

is a core tensor of the same dimensions as $T$. 
Visualization of the HOSVD for a 3D tensor

\[ \mathcal{T} = \mathcal{Z} \times_1 S_1 \times_2 S_2 \times_3 S_3 \]
Basic concepts of the multilinear algebra – HOSVD:

Properties of the core tensor:

Two sutensors $Z_{n_k=a}$ and $Z_{n_k=b}$

obained by fixing the $n_k$ index to $a$, or $b$ respectively, are orthogonal

$Z_{n_k=a} \cdot Z_{n_k=b} = 0$

Subtensors can be ordered according to their norms

$\|Z_{n_k=1}\| \geq \|Z_{n_k=2}\| \geq \ldots \geq \|Z_{n_k=N_p}\| \geq 0$

$\|Z_{n_k=a}\| = \sigma^k_a$ is the $a$-mode singular value of $T$

Each $i$-th vector of the matrix $S_k$ is the $i$-th $k$-mode singular vector
Basic concepts of the multilinear algebra – HOSVD:

\[ \mathcal{T} = \mathcal{Z} \times_1 S_1 \times_2 S_2 \ldots \times_P S_P \]

The basis tensors \( \mathcal{T}_h \) are orthogonal. Hence, \( \mathcal{T}_h \) constitute a basis. This result allows construction of classifiers based on the HOSVD decomposition!

The basis tensors are:

\[ \mathcal{T}_h = \mathcal{Z} \times_1 S_1 \times_2 S_2 \ldots \times_{P-1} S_{P-1} \]

\( s^h_P \) are columns of the unitary matrix \( S_P \).

\( \mathcal{T} = \sum_{h=1}^{N_P} \mathcal{T}_h \times_P s^h_P \)
Basic concepts of the multilinear algebra – HOSVD:

\[ \mathbf{T} = \mathbf{Z} \times_1 \mathbf{S}_1 \times_2 \mathbf{S}_2 \times_p \mathbf{S}_p \]

The basis tensors \( \mathbf{T}_h \) are orthogonal. Hence, \( \mathbf{T}_h \) constitute a basis. This result allows construction of classifiers based on the HOSVD decomposition!

\[ \mathbf{T}_h = \mathbf{Z} \times_1 \mathbf{S}_1 \times_2 \mathbf{S}_2 \times_{p-1} \mathbf{S}_{p-1} \]

\( \mathbf{s}_{hP} \) are columns of the unitary matrix \( \mathbf{S}_p \).
Pattern recognition in the HOSVD spanned multilinear spaces:

Pattern recognition with HOSVD boils down to testing a distance of a given test pattern $P_x$ to its projections in each of the spaces spanned by the set of the bases $\mathbf{T}_h$. This can be done by computing the following minimization problem

$$\min_{i,c_h^i} \left\| P_x - \sum_{h=1}^{N} c_h^i \mathbf{T}_h^i \right\|^2$$

where $c_h^i$ are the coordinates of $P_x$ in the space spanned by $\mathbf{T}_h^i$, $N \leq N_P$ denotes a number of chosen dominating components.

Due to the orthogonality of the tensors $\mathbf{T}_h^i$, the above reduces to the maximization of the following parameter

$$\rho_i = \sum_{h=1}^{N} \left\langle \mathbf{T}_h^i, \hat{P}_x \right\rangle^2$$

$\left\langle .,. \right\rangle$ denotes the scalar product of the tensors, $P_x$ and $\mathbf{T}_h^i$ are normalized. Returned is a class $i$ for which the corresponding $\rho_i$ from is the largest.
The higher $N$, the better fit, though at an expense of computation time. The original tensor $T_i$ of a class $i$ is obtained from the available exemplars of the **prototype patterns** for that class (which can be of different number).
The higher $N$, the better fit, though at an expense of computation time. The original tensor $T_i$ of a class $i$ is obtained from the available exemplars of the **prototype patterns** for that class (which can be of different number).

Where to take the prototype tensors from?

For example, the patterns can be cropped from the training images (road signs) which are additionally rotated in a given range (e.g. $\pm12^\circ$ with a step of $2^\circ$) with additionally added normal noise. Such a strategy allows each pattern to be trained with different number of prototypes.

$$\rho_i = \sum_{h=1}^{N} \langle \hat{T}^i_h, \hat{P}_x \rangle^2$$
A word on implementations
Experimental setup – implementation issues:

The HOSVD algorithm:

1. For each $k=1, \ldots, P$ do:
   a. Flatten tensor $\mathcal{T}$ to obtain $T_k$;
   b. Compute $S_k$ from the SVD decomposition of the flattening matrix $T_k$

\[ T_k = S_k V_k D_k^T \]

2. Compute the core tensor from the formula:

\[ Z = \mathcal{T} \times_1 S_1^T \times_2 S_2^T \ldots \times_P S_P^T \]

The HOSVD algorithm relies on successive application of the matrix SVD decompositions applied to each of the flattened versions $T_{(k)}$ of the original tensor $T$. In result the $S_k$ matrices are obtained. The problem – how to avoid data copying for different modes of $T_{(k)}$ in HOSVD?
The HOSVD algorithm:

1. For each $k=1, \ldots, P$ do:
   a. Flatten tensor $\mathcal{T}$ to obtain $T_k$;
   b. Compute $S_k$ from the SVD decomposition of the flattening matrix $T_k$

\[ T_k = S_k V_k D_k^T \]

2. Compute the core tensor from the formula:

\[ \mathcal{Z} = \mathcal{T} \times_1 S_1^T \times_2 S_2^T \times_3 \cdots \times_P S_P^T \]

Experimental setup – implementation issues:

The HOSVD algorithm relies on successive application of the matrix SVD decompositions applied to each of the flattened versions $T_{(k)}$ of the original tensor $T$. In result the $S_k$ matrices are obtained.

The problem – how to avoid data copying for different modes of $T_{(k)}$ in HOSVD?

Can be parallelized (Cyganek, 2013)
A class hierarchy for efficient tensor representation

```
TLimageFor
# fData : pixel type = val
+ GetPixel( matrix_index ) : PixelType
+ SetPixel( matrix_index ) : PixelType

TFlatTensorFor
# fTensorMode : int
# fIndexVector : vector
+ GetElement( TensorIndex ) : EType
+ SetElement( TensorIndex, EType )
# Offset_ForwardCyclic( TensorIndex, MatrixIndex, Mode )
# Offset_BackwardCyclic( TensorIndex, MatrixIndex, Mode )

TFlatTensorProxyFor
  - fMotherTensor : TFlatTensorFor &
+ GetElement( TensorIndex ) : EType
+ SetElement( TensorIndex, EType )
+ GetPixel( matrix_index ) : PixelType
+ SetPixel( matrix_index ) : PixelType
# Offset_ForwardCyclic( MatrixIndex, TensorIndex, Mode )
# Offset_BackwardCyclic( MatrixIndex, TensorIndex, Mode )
```

Proxy Design Pattern
Software framework for tensor representation and decomposition:

DeRecLib available on the Internet for free ...
Software framework for tensor representation and decomposition:

DeRecLib available on the Internet for free ...

Object-oriented design (templates, C++)
http://home.agh.edu.pl/~cyganek/cyganekobject.htm

This was a single tensor classifier but let’s build an ensemble of such to run parallel...
Tensor processing usually results in high computational and memory demands. The former can be alleviated by parallel implementation of the specifically chosen parts of the system. In this approach we exploit both strategies for parallel decompositions:

- Data decomposition.
- Functional decomposition.

However, in many aspects parallel operation of some of the software blocks leads to higher memory demands. Therefore both aspects need to be considered together.
Parallel Implementation of the **Ensemble of Tensor Classifiers**

We have already seen a general scheme…
Parallel Implementation of the **Ensemble of Tensor Classifiers**

Architecture of the ensemble composed of the multi-class HOSVD classifiers.

Each member multi-class classifier is trained with its **partition** of training data from each class.

Data partitions are obtained due to the **bagging** process.
Some results…
Experimental results:

USPS dataset contains selected and preprocessed scans of the handwritten digits from envelopes by the U.S. Postal Service.

The database is divided into the **7291 training** and 2007 testing partitions.

The system was implemented in C++ and run on the computer with the 8 GB RAM and Pentium Core Q 820 @ 1.73GHz.
Experimental results:

Each test and train pattern originally is in a form of a $16 \times 16$ gray level image. Since this dataset is perceived as a relatively difficult for machine classification (reported human error is 2.5%), it has been used for comparison of different classifiers.

Each experimental setup was run number of times (from 3 to 10, depending on computational complexity) and an average answer is reported. In all cases the Gaussian noise was added to the input image at level of 10%,
Experimental results:

Influence of the number of data points used in bagging process on accuracy of the ensembles with different number of members

Accuracy vs. number of classifiers in the ensemble for 3 sizes of data samples in the bagging (192, 256, 560). Input images of size 16×16. In number of components \( H=16 \)
Experimental results:

Performance of the ensembles of different number of members in respect to the two different sizes of the input images: $16\times16$ vs. $32\times32$ pixels

Accuracy vs. number of classifiers in the ensemble for two different sizes of input images: original ($16\times16$) vs. enlarged by image warping ($32\times32$). In both cases 192 data samples (images) were used in bagging. Components $H=16$. 
Experimental results:

Accuracy vs. number of components $H$. Input images were warped to 32×32, number of classifiers in ensemble set to 15, bagging partitions of 192 images used.
Experimental results:

Accuracies and parameters for each digit separately:

Experimental setup: 15 members in the ensemble, input data transformed to 32×32 resolution, 192 training images from bagging, $H=16$ components.

<table>
<thead>
<tr>
<th>Digit</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Train</td>
<td>1194</td>
<td>1005</td>
<td>731</td>
<td>658</td>
<td>652</td>
<td>556</td>
<td>664</td>
<td>645</td>
<td>542</td>
<td>644</td>
</tr>
<tr>
<td>#Test</td>
<td>359</td>
<td>264</td>
<td>198</td>
<td>166</td>
<td>200</td>
<td>160</td>
<td>170</td>
<td>147</td>
<td>166</td>
<td>177</td>
</tr>
<tr>
<td>Accuracy</td>
<td>0.986</td>
<td>0.981</td>
<td>0.899</td>
<td>0.880</td>
<td>0.930</td>
<td>0.925</td>
<td>0.982</td>
<td>0.959</td>
<td>0.910</td>
<td>0.960</td>
</tr>
</tbody>
</table>
OpenMP again…

System training speed-up ratio of the serial and parallel implementations for different number of member classifiers \( E \) in the ensemble and different chunks of data from bagging. 64 chunks.
OpenMP again…

System training speed-up ratio of the serial and parallel implementations for different number of member classifiers E in the ensemble and different chunks of data from bagging. 128 chunks (b).
Conclusions
What we have gained with parallel approach?

Our **DESIGN** into a parallel system lead us to an **ENSEMBLE** of classifiers.

1. We gain **speed** improvement (on different levels and thanks to **OpenMP**)  
2. We obtained **better accuracy** (due to the ensemble of classifiers)  
3. We made **easier training** (due to boosting – smaller data partitions)

So, when starting/improving a classification system

1. Think of splitting data
2. Think of many (simple) classifiers
3. Try to optimize even a single classifier in the ensemble

Future…?
Literature

Application Programming Interface Version 4.5 OpenMP Architecture Review Board November 2015  
http://www.openmp.org

Application Programming Interface Examples Version 4.0.2 – OpenMP Architecture Review Board March 2015  
http://www.openmp.org

Thank you!

