Parallel Implementations of SVM for Earth Observation

Jordi Muñoz-Marí [a], Antonio J. Plaza [b], J. Anthony Gualtieri [c], Gustavo Camps-Valls [a]

[a] *Department of Electronics Engineering, University of Valencia, Spain, {jordi,gcamps}@uv.es, http://www.uv.es/{jordi,gcamps}*

[b] *Department of Computer Science, University of Extremadura, Cáceres, Spain. aplaza@unex.es http://www.umbc.edu/rssipl/people/aplaza*

[c] *NASA's Goddard Space Flight Center, Greenbelt, Maryland, USA, Anthony.Gualtieri@gst.com*

Abstract. Imaging spectroscopy, also known as hyperspectral remote sensing, is concerned with the measurement, analysis, and interpretation of spectra acquired from a given scene (or specific object) at a short, medium or long distance by an airborne or satellite sensor. Analysis in a timely manner of the acquired multi-dimensional images allows to develop applications with high social impact, such as urban growing monitoring, crop fields identification, target detection for military and defense/security deployment, wildland fire detection and monitoring, biological threat detection, biophysical parameter estimation, or monitoring of oil spills and other types of chemical contamination.

In this context, support vector machines (SVM) [1, 2, 3] have become one of the state-of-the-art machine learning tools for hyperspectral image classification. However, its high computational cost for large scale applications makes the use of SVM limited to off-line processing scenarios. Certainly, with the recent explosion in the amount and complexity of hyperspectral data, parallel processing has soon become a requirement in many remote sensing missions, especially with the advent of low-cost systems such as commodity clusters and distributed networks of computers In order to address this relevant issue, this chapter explores the development of two parallel versions of SVMs for remote sensing image classification.

Sequential minimal optimization is a very popular algorithm for training SVMs, but it still requires a large amount of computation time for solving large size problems. In this work, we evaluate the performance of a parallel implementation of the SVM based on the parallelization of the incomplete Cholesky factorization and present novel parallel implementations that balance the load across the available processors through standard Master-Worker decompositions. Both methodologies are theoretically analyzed in terms of scalability, computational efficiency and time response. The impact of the multi-class scheme is also analyzed. Results on real multispectral and hyperspectral datasets illustrate the performance of the methods. We finally discuss the possibility of obtaining processing results quickly enough for practical use via the Marenostrum supercomputer available at Barcelona Supercomputing Center in Spain, and other massively parallel facilities at NASA's Goddard Space Flight Center in Maryland.

Keywords. Parallel processing, image classification, remote sensing, support vector machine.

1. Introduction

Materials in a scene reflect, absorb, and emit electromagnetic radiation in a different way depending of their molecular composition and shape. Remote sensing exploits this physical fact and deals with the acquisition of information about a scene (or specific object) at a short, medium or long distance. The radiation acquired by an (airborne or satellite) sensor is measured at different wavelengths and the resulting spectral signature (or *spectrum*) is used to identify a given material. The field of *spectroscopy* is concerned with the measurement, analysis, and interpretation of such spectra [4]. Figure 1 shows a schematic example of the application of imaging spectroscopy to perform satellite remote sensing.

Earlier sensor developements (often called *multispectral*) considered a few number of bands, which readily demonstrated to be a limitation for detecting similar materials. Thus, a new class of imaging spectroscopy sensors, which acquire hundreds of contiguous narrow bands or channels appeared, and are called hyperspectral (imaging) sensors. Hyperspectral sensors sample the reflective portion of the electromagnetic spectrum ranging from the visible region (0.4-0.7μm) through the near-infrared (about 2.4 μm) in hundreds of N narrow contiguous bands about 10 nm wide[1]. Hyperspectral sensors represent an evolution in technology from earlier multispectral sensors, which typically collect spectral information in only a few discrete, non-contiguous bands.

The high spectral resolution characteristic of hyperspectral sensors preserves important aspects of the spectrum (e.g., shape of narrow absorption bands), and makes differentiation of different materials on the ground possible. The spatially and spectrally sampled information can be described as a data cube (colloquially referred to as "the hypercube"), which includes two spatial coordinates and the spectral one (or wavelength). As a consequence, each image pixel is defined in a (potentially very high) dimensional space where each dimension corresponds to a given wavelength interval in the spectrum.

The use of hyperspectral images for Earth Observation is a consolidated technology since the (increasing) high number of spectral bands contained in the acquired image allows excellent characterization, identification, and classification of the land-covers [4]. However, many classifiers are affected by input sample dimension, tend to overfit data in the presence of noise, or the computational cost poorly scales with the number of samples. All these problems are related theoretically through the well-known *curse of dimensionality* by which developing a classifier with low number of high-dimensional samples runs the risk of overfitting the training data, and then showing poor generalization performance [5, 6].

In the last few years, the use of support vector machines (SVMs) [7, 8, 9, 3] for hyperspectral image classification has been paid attention basically because the method integrates in the same classification procedure: (i) a *feature extraction* step, as samples are mapped to a higher dimensional space where a simpler (linear) classification is performed, becoming non-linear in the input space; (ii) a *regularization* procedure by which model's complexity is efficiently controlled; and (iii) the minimization of an upper bound of the generalization error, thus following the structural risk minimization principle. These theoretical properties make the SVM very attractive in the context of hyper-

[1] Other types of hyperspectral sensors exploit the emissive properties of objects by collecting data in the mid-wave and long-wave infrared (MWIR and LWIR) regions of the spectrum.

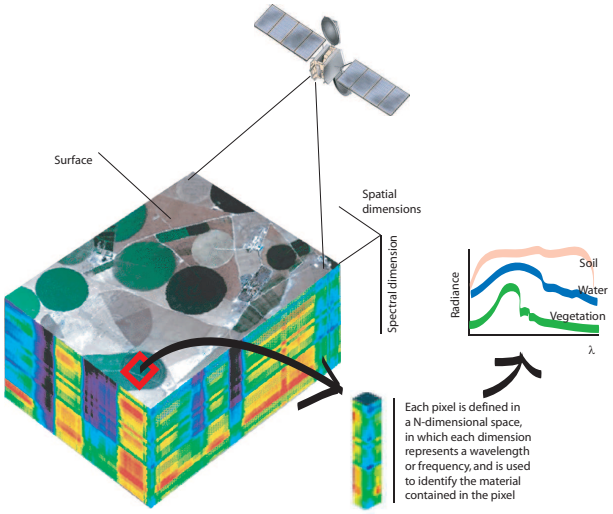

Figure 1. Principle of imaging spectroscopy.

spectral image classification where the classifier commonly deals with a low number of high dimensional training samples [10, 1, 9, 3].

However, SVMs have the problem of scaling badly with the number of training samples. Also, the problem of finding a good set of free parameters is an important one in the SVM framework. Specifically, the choice of the kernel and its associated free parameter is a crucial (and so far unsolved) problem, making the training process very heuristic and computationally very demanding [9]. With the recent explosion in the amount and complexity of hyperspectral data, and with the increasing availability of very high resolution images, the number of labeled samples to train the classifiers is becoming a critical problem. In this scenario, parallel processing has become a requirement in many remote sensing missions, especially with the advent of low-cost systems such as commodity clusters and distributed networks of computers [11, 12, 13]. In order to address this relevant issue, this chapter explores the development of parallel versions of SVMs for hyperspectral image classification. In particular, we focus on two different approaches: a boss-worker approach and a parallelization based on the direct decomposition of the kernel matrix via Cholesky decomposition.

The rest of the chapter is outlined as follows. Section 2 presents related work in the context of other parallel and grid-enabled applications for the considered problem. Section 3 revises the fundamentals of kernel methods and SVM. Noting the critical problem of computing, storing and operating with huge kernel matrices, Section 4 revises an efficient parallel formulation of the algorithm, recently presented in [14]. Section 5 pays attention to the experimental results obtained in both hyperspectral and multispectral images, and analyzes the scalability and accuracy issues. Section 6 concludes with some remarks and further work.

2. Related Work

Several efforts have been presented in the recent literature regarding the parallelization of algorithms for remotely sensed hyperspectral image analysis (including kernel methods) using high performance computing architectures [15, 16, 17]. The idea of using COTS (commercial off the shelf) computer equipment, clustered together to work as a computational "team" has been a very attractive solution (often referred to as Beowulf-class cluster computing) which has already offered access to greatly increased computational power, but at a low cost (commensurate with falling commercial PC costs) in a number of hyperspectral imaging applications [13, 18]. On the other hand, although most parallel techniques and systems for hyperspectral image information processing have traditionally been homogeneous in nature (i.e., made up of identical processing units), a recent trend in the design of high performance computing systems for data-intensive problems in remote sensing is to utilize highly heterogeneous computing resources [19, 20]. This heterogeneity is seldom planned, arising mainly as a result of technology evolution over time and computer market sales and trends. In this regard, networks of heterogeneous COTS resources can realize a very high level of aggregate performance in hyperspectral imaging applications, and the pervasive availability of these resources is now allowing the application of grid computing practices to remote sensing and hyperspectral imaging problems [15, 21]. It is expected that grid-based high performance computing systems will soon represent a tool of choice for the scientific community devoted to very high-dimensional data analysis in remote sensing, as it has been the case with other fields.

Finally, although remote sensing data processing algorithms generally map quite nicely to parallel systems made up of commodity CPUs, these systems are generally expensive and difficult to adapt to onboard remote sensing data processing scenarios, in which low-weight and low-power integrated components are essential to reduce mission payload and obtain analysis results in real time, i.e., at the same time as the data are collected by the sensor. In this regard, an exciting new development in the field of commodity computing is the incorporation of programmable hardware devices into onboard remote sensing applications, including field programmable gate arrays (FPGAs) [18] and graphic processing units (GPUs) [22, 23, 24, 25], which can bridge the gap towards onboard and real-time analysis of remote sensing data. In this work, however, we have focused on evaluating the capabilities of CPU-based clusters in the context of remote sensing image classification with SVM. Further work will analyze the GPU-based clusters in this matter.

3. Kernel Methods and the Support Vector Machine

This section reviews the main ideas under the framework of learning problems with kernel methods, and analyzes the standard formulation of the SVM. Also, it points out the main problems involved from the computational complexity viewpoint.

3.1. Fundamentals on Kernel Methods

When using linear algorithms, a well-established theory and efficient methods are often available. Kernel methods exploit this fact by embedding the data set S defined over

the input or attribute space $\mathcal{X}$ ($S \subseteq \mathcal{X}$) into a higher (possibly infinite) dimensional Hilbert space $\mathcal{H}$, or *feature space*, and then they build a linear algorithm therein, resulting in an algorithm which is nonlinear with respect to the input data space. The mapping function is denoted as $\boldsymbol{\phi} : \mathcal{X} \rightarrow \mathcal{H}$. Though linear algorithms will benefit from this mapping because of the higher dimensionality of the feature space, the computational load would dramatically increase because we should compute sample coordinates in that high dimensional space. This computation is avoided through the use of the kernel trick by which, if an algorithm can be expressed with dot products in the input space, its (non-linear) kernel version only needs the dot products among mapped samples. Kernel methods compute the similarity between training samples $S = \{\mathbf{x}_i\}_{i=1}^n$ using pair-wise inner products between mapped samples, and thus the so-called kernel matrix $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) = \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j) \rangle$ contains all the necessary information to perform many classical linear algorithms in the feature space.

3.2. Support Vector Machines

One of the most widely used kernel methods is the support vector machine (SVM). The classification methodology of SVMs attempts to separate samples belonging to different classes by tracing maximum margin hyperplanes in the kernel space where samples are mapped to (see Fig. 2). Maximizing the distance of samples to the optimal decision hyperplane is equivalent to minimizing the norm of $\mathbf{w}$, and thus this becomes the first term in the minimizing functional. For better manipulation of this functional, the quadratic norm of the weights is preferred. Therefore, following previous notation, the SVM method solves the following primal problem:

$$\min_{\mathbf{w}, \xi_i, b} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \xi_i \right\} \tag{1}$$

constrained to:

$$y_i(\langle \boldsymbol{\phi}(\mathbf{x}_i), \mathbf{w} \rangle + b) \geq 1 - \xi_i \qquad \forall i = 1, \ldots, n \tag{2}$$

$$\xi_i \geq 0 \qquad \forall i = 1, \ldots, n \tag{3}$$

where $\mathbf{w}$ is the normal to the optimal decision hyperplane defined as $\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle + b = 0$, and b represents the bias or closest distance to the origin of the coordinate system. These parameters define a linear classifier in the kernel space $\mathcal{H}$:

$$\hat{y}_* = f(\mathbf{x}_*) = \operatorname{sgn}(\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_*) \rangle + b) \tag{4}$$

The non-linear function $\boldsymbol{\phi}$ maps samples to a higher dimensional space, which in accordance with Cover's theorem [26], guarantees that the transformed samples are more likely to be linearly separable. The regularization parameter C controls the generalization capabilities of the classifier, and ξ_i are positive slack variables allowing to deal with permitted errors.

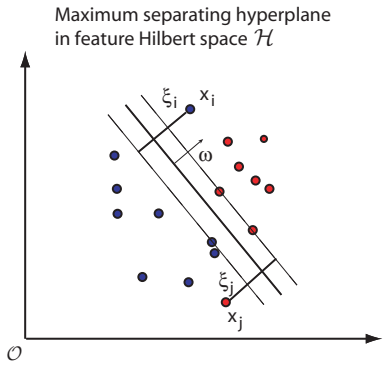

Figure 2. SVM: Linear decision hyperplanes in a non-linearly transformed space, where *slack* variables ξ_i are included to deal with errors.

3.2.1. SVM and the Kernel Trick

The above problem is solved by introducing Lagrange multipliers ($\xi_i \geq 0, \mu_i \geq 0$) for each constraint:

$$\min_{\mathbf{w},\boldsymbol{\xi},b} \left\{ \|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i [y_i(\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_i)\rangle + b) + \xi_i - 1] - \sum_{i=1}^{n} \mu_i \xi_i \right\} \quad (5)$$

Now, by making zero the gradient of this primal-dual functional $\mathcal{L}_{pd}$ with respect to the primal variables ($\mathbf{w}$, b, ξ_i), we obtain the following conditions:

$$\frac{\partial \mathcal{L}_{pd}}{\partial \mathbf{w}} = 0 \rightarrow \mathbf{w} = \sum_i y_i \alpha_i \boldsymbol{\phi}(\mathbf{x}_i) \quad (6)$$

$$\frac{\partial \mathcal{L}_{pd}}{\partial b} = 0 \rightarrow \sum_i y_i \alpha_i = 0 \quad (7)$$

$$\frac{\partial \mathcal{L}_{pd}}{\partial \xi_i} = 0 \rightarrow C = \alpha_i + \mu_i, \;\; i = 1, \ldots, n \quad (8)$$

If constraints (6)-(8) are included in the Lagrange functional $\mathcal{L}_{pd}$ (Eq. (5)) in order to remove the primal variables, the dual problem $\mathcal{L}_d$ to be solved is obtained:

$$\max_{\boldsymbol{\alpha}} \left\{ \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j)\rangle \right\}, \quad (9)$$

constrained to $0 \leq \alpha_i \leq C$ and $\sum_i \alpha_i y_i = 0, \forall i = 1, \ldots, n$. In this way, one gets rid of the explicit usage of very high dimensional vectors, $\mathbf{w}$, and optimizes (9) with respect to variables α_i instead.

It is worth noting that all $\boldsymbol{\phi}$ mappings used in the SVM learning occur in the form of inner products. This allows us to define a kernel function K:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j)\rangle, \quad (10)$$

and then, without considering the mapping $\boldsymbol{\phi}$ explicitly, a non-linear SVM can be defined. Note that the pair $\{\mathcal{H}, \boldsymbol{\phi}\}$ will only exist if the kernel function K fullfils Mercer's conditions[2].

3.2.2. Solving the QP problem

The problem in (9) constitutes a quadratic programming (QP) problem with linear constraints. Let us recall that QP problems are a special kind of mathematical optimization problem in which the goal is to optimize (minimize or maximize) a quadratic functional of several variables subject to linear constraints on these variables. Assuming a variable $\mathbf{x} \in \mathbb{R}^n$, an $n \times n$ matrix $\mathbf{Q}$ is symmetric, and $\mathbf{c}$ is any $n \times 1$ vector, a QP problem is formulated as:

$$\min_{\mathbf{x}} \left\{ \frac{1}{2}\mathbf{x}^\top Q\mathbf{x} + \mathbf{c}^\top \mathbf{x} \right\}, \tag{11}$$

subject to one (or more) constraints of the form:

$$\mathbf{A}\mathbf{x} \leqslant \mathbf{b} \text{ (inequality constraint)} \tag{12}$$

$$\mathbf{E}\mathbf{x} = \mathbf{d} \text{ (equality constraint)} \tag{13}$$

where $\mathbf{v}^T$ indicates the vector transpose of $\mathbf{v}$. The notation $\mathbf{A}\mathbf{x} \leqslant \mathbf{b}$ means that every entry of the vector $\mathbf{A}\mathbf{x}$ is less than or equal to the corresponding entry of the vector $\mathbf{b}$.

Comparing (9) and (11), one can identify the following terms: minimizing variable $\mathbf{x} = \boldsymbol{\alpha}$, matrix $\mathbf{Q} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$, $\mathbf{E} = \mathbf{Y}$ is a column vector containing the actual labels, $\mathbf{A} = \mathbf{I}$ is a column unitary vector, $\mathbf{d} = \mathbf{0}$, and $\mathbf{b} = \mathbf{C}$ is a column vector containing the value of the regularization (penalization) parameter C. This QP problem can be solved optimally with many available software packages (Matlab, LaPACK, etc), but as will be analyzed in detail in the next section, alternative more efficient ways are available.

3.2.3. Predicting with SVM

After solving the QP problem with a given kernel function, one obtains the optimal Lagrange multipliers, α_i, which reflect the relative relevance of each sample for classification. The key issue is that we have worked implicitely in the higher dimensional feature space, and retrieve a weight vector in the original input space. Note that, by solving the optimization problem, the feature-space model parameters $\mathbf{w}$ are expressed as a linear expansion of the mapped samples $\boldsymbol{\phi}(\mathbf{x}_i)$ through the dual parameters α_i, i.e. $\mathbf{w} = \sum_{i=1}^{n} y_i \alpha_i \boldsymbol{\phi}(\mathbf{x}_i)$ (Eq. (6)).

By plugging (6) into (4), the decision function for any test vector $\mathbf{x}_*$ is given by:

$$\hat{y}_* = f(\mathbf{x}_*) = \operatorname{sgn}\left(\sum_{i=1}^{n} y_i \alpha_i K(\mathbf{x}_i, \mathbf{x}_*) + b \right), \tag{14}$$

[2] According to Hilbert-Schmidt theory, $K(\mathbf{x}_i, \mathbf{x}_j)$ can be any symmetric function satisfying Mercer's conditions. This was firstly stated by [27]. The same idea was used by [28] for the analysis of the convergence properties of the method of potential functions, and happened at the same time as the method of the optimal hyperplane was developed by [29]. Full details on the Mercer's conditions can be obtained from [30].

where b is calculated using the primal-dual relationship, and where only samples with non-zero Lagrange multipliers α_i account for the solution. This leads to the very important concept of *sparsity*, i.e. the solution is expressed as a function only of the most critical training samples in the distribution, namely *support vectors* (SV). For deeper analysis and application of SVMs in remote sensing image classification refer to [31, 32, 10, 1].

3.3. Large Scale SVM Implementations

Support Vector Machines (SVMs) suffer from a widely known scalability problem in both memory use and computational time. Solving a SVM implies solving a quadratic programming (QP) problem.

Note that, in the case of the SVM, the QP problem is directly expressed as a function of the training kernel matrix K, which contains the similarity (distance) among all training samples. Thus, solving this problem requires storing the matrix and making operations with it. At present, the most effective interior point method (IPM) for solving the QP problem with linear constraints is the primal-dual IPM [33], which essentially tries to remove inequality constraints using a bounding function, and then exploit the iterative Newton's method to solve the Karush-Kuhn-Tucker (KKT) conditions related to the Hessian matrix $\mathbf{Q}$. This is very computationally demanding as it grows as $\mathcal{O}(n^3)$ in time and $\mathcal{O}(n^2)$ in space, where n is the number of training samples

Some alternative algorithms have appeared to solve the problem in reasonable time and amount of memory for several thousands of training samples. A powerful approach to scale up SVM training is by using decomposition methods [34], which break a large QP problem into a series of manageable QP subproblems. The most popular decomposition method is the *sequential minimal optimization* (SMO) [35]. In the SMO, the smaller QP problems are solved analytically, which avoids using a time-consuming numerical QP optimization as an inner loop. The amount of memory required for SMO is linear in the training set size, which allows SMO to handle very large training sets. Without kernel caching, SMO scales somewhere between linear and quadratic in the training set size for various test problems, while a standard projected conjugate gradient (PCG) chunking algorithm scales somewhere between linear and cubic in the training set size. SMO's computation time is dominated by SVM evaluation, hence SMO is fastest for linear SVMs and sparse data sets. Despite these advantages, the SMO algorithm still requires a large amount of computation time for solving large scale problems, and also the kernel storage which is of the size of the square of training points.

In recent years, other scale-up strategies have been proposed. The SimpleSVM algorithm [36] makes use of a greedy working set selection strategy to identify the most relevant samples (support vectors) for incremental updating of the kernel submatrix. In [37], authors propose an online SVM learning together with active example selection. In [38], special data structures for fast computations of the kernel in their chunking algorithm are exploited. Besides, instead of maximizing the dual problem as is usually done, in [39] propose to directly minimize the primal problem. Some other powerful algorithms for solving SVM have recently appeared, such as the Core SVM or the Ball Vector Machine (BVM). However, large scale problems involving $n > 10^5$ samples require more efficient, parallel versions of SVM to solve the problem in reasonable time.

In this work, we focus on parallel implementations of the SVM. The first attempts on parallelization of SVM were presented in [40, 41]. An alternative approach was pro-

posed in [42] based on cascade schemes. A different motivation based on randomized sample selection techniques was introduced in [43]. Also very recently, other approaches have been presented with improved performance, such hybrid [44], or cost effective [45] strategies. In [46], a parallel implementation of the SMO was presented. The parallel SMO was developed using message passing interface (MPI) by first partitioning the entire training set into smaller subsets and then simultaneously run multiple CPU processors to deal with each of the partitioned data sets. Experiments showed a considerable speedup. Nevertheless, the algorithm had to work with the samples, not with the kernel, which may eventually offer interesting options, such as the development of new classifiers or the combination of multi-source and multi-temporal image features [47, 48, 3].

An interesting alternative to the parallelization of the SMO has been presented in [14]. This parallel SVM algorithm (PSVM) is able to reduce the memory use and to parallelize both data loading and computation, and at the same time works directly with the precomputed kernel matrix. Given n training instances each with d dimensions, the PSVM first loads the training data in a round-robin fashion onto m machines[3]. Next, PSVM performs a parallel row-based Incomplete Cholesky Factorization (ICF) on the loaded data [49]. At the end of parallel ICF, each machine stores only a fraction of the factorized matrix. PSVM then performs parallel IPM to solve the QP optimization problem, while the computation and the memory demands are improved w.r.t. other decomposition-based algorithm, such as the SVMLight [50], libSVM [51], SMO [35], or SimpleSVM [36].

4. Parallel Implementations of SVM

The increasing sophistication of hyperspectral image processing algorithms, along with the high dimensionality and volume of the data, has made parallel processing a requirement in many remote sensing applications, especially with the advent of low-cost parallel systems such as commodity clusters [52]. In this section, we revise the parallel versions of SVM used in this chapter. Before describing the parallel algorithms used, we first provide an overview of the common parameters to evaluate and design a parallel processing algorithm.

4.1. Parameters for Parallel Algorithm Design

There are multiple, sometimes conflicting, design goals to be considered when developing a parallel algorithm. One could maximize static runtime characteristics such as *speedup* – sequential execution runtime divided by parallel execution time, or *efficiency* – speedup divided by the number of processors employed. Another goal could be maximizing *scalability* – the ability to maintain constant efficiency when both the problem size and the number of processors are increased i.e., in an attempt to increase the longevity of a solution in the face of continuously improving computation power. Yet another dimension could be to maximize *productivity* – the usefulness of a solution as a function of time divided by its costs – or in particular *development time productivity* which is defined as speedup divided by relative effort – the effort needed to develop a serial version

[3]Note that this strategy is feasible for homogenous parallel machines only, but not for heterogeneous environments.

divided by the effort needed to develop a parallel solution [53]. Note that, in this context, the use of skeletons has played an important role [54].

The two proposed implementations, though conceptually different, seek the same optimality criterion: to maximize speedup while also considering productivity. The first parallel SVM considered in this work is based on the efficient solution of the IPM problem [14]. So far, we only used this method for binary problems. This is certainly a limitation in remote sensing scenarios where there are typically several land cover classes of interest in the scene. Also, in this strategy, no attention is paid to the spatial variability of the spectral signature, since the algorithm works at a pixel level. These are shortcomings to be addressed in the future. The second approach remedies these problems by adopting a master-worker strategy and a smart image treatment.

4.2. Parallel SVM via Cholesky Factorization

The PSVM method originally introduced in [14] is aimed at reducing memory use through performing a row-based, approximate matrix factorization. The key step of PSVM is parallel ICF (PICF). Traditional column-based ICF [55, 56] can reduce computational cost, but the initial memory requirement is not efficient for very large datasets. Alternatively, the PSVM performs parallel row-based ICF as its initial step, which loads training instances onto parallel machines and performs factorization simultaneously on these machines. Once PICF has loaded n training data distributedly on m machines, and reduced the size of the kernel matrix through factorization, IPM can be solved on parallel machines simultaneously.

4.2.1. Implementation of the Algorithm

Notationally, ICF approximates the Hessian matrix $\mathbf{Q}$ (of size $n \times n$) by a smaller matrix $\mathbf{H}$ (of size $p \times n$), i.e., $\mathbf{Q} = \mathbf{H}\mathbf{H}^\top$. ICF, together with the exploitation of the Sherman-Morrison-Woodbury formula[4], can greatly reduce the computational complexity in solving an $n \times n$ linear system. The work of [55] provides a theoretical analysis of how ICF influences the optimization problem in (9).

The PSVM in [14] iterates until the approximation of $\mathbf{Q}$ by $\mathbf{H}_k\mathbf{H}_k^\top$ (measured by $trace(\mathbf{Q} - \mathbf{H}_k\mathbf{H}_k^\top)$) is satisfactory, or the predefined maximum iterations (or say, the desired rank of the ICF matrix) p is reached. As suggested in [49], a parallelized ICF algorithm can be obtained by constraining the parallelized Cholesky Factorization algorithm, iterating at most p times. However, in the proposed algorithm [49], matrix $\mathbf{H}$ is distributed by columns in a round-robin way on m machines (hence it is called 'column-based parallelized ICF'). Such column-based approach is optimal for the single-machine setting, but cannot gain full benefit from parallelization because of the (i) large memory requirement, as each machine must be able to store a local copy of the training data, and (ii) limited parallelizable computation, since the summation of local inner product result and the vector update must be performed on one single machine. Therefore, rather than performing column-wise, a row-based approach starts by initializing variables and loading training data onto m machines in a round-robin fashion. The algorithm then performs

[4]The Sherman-Morrison-Woodbury formula from linear algebra states that $(\mathbf{C} + \mathbf{A}\mathbf{B})^{-1} = \mathbf{C}^{-1} - \mathbf{C}^{-1}\mathbf{A}(\mathbf{I} + \mathbf{B}\mathbf{C}^{-1}\mathbf{A})^{-1}\mathbf{B}\mathbf{C}^{-1}$, where $\mathbf{C}$ is an invertible $n \times n$ matrix, $\mathbf{A} \in \mathbb{R}^{n\times m}$ and $\mathbf{B} \in \mathbb{R}^{m\times n}$.

the ICF main loop until the termination criteria are satisfied (e.g., the rank of matrix $\mathbf{H}$ reaches p).

At the end of the algorithm, $\mathbf{H}$ is stored distributedly on m machines, ready for parallel IPM [14]. PICF enjoys three advantages: (i) parallel memory use ($\mathcal{O}(np/m)$), (ii) parallel computation ($\mathcal{O}(p^2n/m)$), and (iii) low communication overhead ($\mathcal{O}(p^2log(m))$). Particularly on the communication overhead, its fraction of the entire computation time shrinks as the problem size grows. We will verify this in the experimental section. This pattern permits a larger problem to be solved on more machines to take advantage of parallel memory use and computation. More details can be found in [14].

4.2.2. Efficiency of the Algorithm

The method loads only essential data to each machine to perform parallel computation. Let n denote the number of training instances, p the reduced matrix dimension after factorization (p is significantly smaller than n), and m the number of machines. PSVM reduces the memory requirement from $\mathcal{O}(n^2)$ to $\mathcal{O}(np/m)$, and improves computation time to $\mathcal{O}(np^2/m)$.

4.3. Parallel SVM via Master-Worker Architecture

The design of a parallel SVM for processing hyperspectral images is driven by the desire to maximize speedup while also considering productivity, with the ultimate goal of reusing most of the code for the sequential algorithms in the parallel implementations. For addressing speedup, or more specifically, to balance the load across the available processors, we used a standard master-slave decomposition framework. In this approach, a particular processor of the cluster is designated as the *master*, whose job is to decompose the problem and build a list of tasks that will be assigned to the *workers*. The master processor then waits to receive back the completed tasks from each worker processor, gathers the partial results from them and provides the final result. Under the above framework, the best situation arises when a problem can be decomposed into a non-trivial number of independent sub-problems. In the jargon of parallel computation, this is called an *embarrassingly parallel* problem [57].

4.3.1. Implementation of the Algorithm

Although theoretically there are many alternatives, in practice, two common domain decomposition strategies are most often used in hyperspectral processing due to their straightforward partitioning of the problem space:

1. *Spatial-domain decomposition* subdivides the image into multiple blocks made up of entire pixel vectors, and assigns one or more blocks to each processing element
2. *Spectral-domain decomposition* subdivides the whole multi-band data into blocks made up of contiguous spectral bands (sub-volumes), and assigns one or more sub-volumes to each processing element.

It should be noted that spectral-domain decomposition techniques break the spectral identity of the data because each pixel vector is split amongst several processing elements. As a result, spatial-domain data decomposition is generally preferred in parallelization of hyperspectral image processing algorithms [13]. There are several reasons

that justify the above decision. First, spatial-domain decomposition preserves the information carried out by the entire spectral signature of each hyperspectral image pixel. Second, it provides a natural approach for low level image processing [58], as many operations require the same function to be applied to a small set of elements around each data element present in the image data structure. A final important consideration is that in spectral-domain decomposition, the calculations made for each hyperspectral pixel may require input originating from neighboring processing elements, and thus require intensive inter-processor communication.

4.3.2. Multiclass Implementation

We give in this section a parallel version of supervised classification using SVMs for multiple-class problems [12]. A standard approach in kernel methods is to decompose the multiple class problem into multiple two-class problems. In order to develop our parallel version, we use *pairwise classification* to create $\frac{S(S-1)}{2}$ separate pairs of classifiers. Then, we use a a voting strategy which is based on building S pairs of classifiers, each of which with the pair of classes consisting of class i, and the combined class of $1, 2, \ldots, i-1, i+1, \ldots S$ [59].

A rough argument can be given for choosing *pairwise classification* instead of *one versus the rest* for computational efficiency. Assume that the time complexity of using the SVM algorithm for two classes scales as $O(l^2)$, where l is the total number of pair training vectors. The scaling above is based on the complexity of quadratic optimization for finding the support vectors in the training vectors. If we assume that each training class is of the same size, i.e., $l/2$, then the time complexity for the two approaches is given by Table 1, which shows that *pairwise classification* is more efficient. The downside of the above approach is that the number of pairwise SVM's grows as S^2 for *pairwise classification*. For $S = 16$, the number of pairs is 120. However, we can decompose the problem into $\frac{S(S-1)}{2}$ independent pairs of computations. Thus, we can efficiently use a parallel cluster machine having only processor-to-processor communication between K compute nodes, without shared memory.

A simple approach, which in hindsight is naive, would be to run in groups of K pairs in lockstep, distributed across the processors. However, typically the number of training vectors for each pair is not the same, resulting in the fact that the processor handling the largest number training vectors will always be the last to finish, leaving other processors idle. To address this issue, our parallel algorithm uses a master-slave approach (i.e., one processor is used as the master and the rest are used as workers). The master first builds a list of tasks, given by the set of all $\frac{S(S-1)}{2}$ pairs of classes, and then assigns a particular processor to each of the first $K-1$ tasks. When a worker processor completes its calculations, the master sends that worker the next task on the list. Using this simple approach, the worker processors are always busy.

4.3.3. Efficiency of the Algorithm

In this subsection, we provide a summary of the computations executed by the workers for each pair of classes during the parallel computation:

1. *Training*. From the given training vectors for each of the two classes, find the support vectors for that pair (also called 'the model').

Table 1. Time complexity on a single processor for performing SVM training for a classifier of S classes, each class containing $l/2$ exemplars.

Pairwise classification	*One-versus-the-rest*
$\mathcal{O}(S^2 l^2)$	$\mathcal{O}(S^3 l^2)$
$S(S-1)/2\mathcal{O}(l^2)$	$S * \mathcal{O}(((S-1)\frac{l}{2} + \frac{l}{2})^2)$

2. *Prediction*. From the obtained model, construct the pair classifier and apply it to the full hyperspectral data cube.
3. *Classification*. Using a voting strategy at each pixel, find the class with the largest number of votes from the $\frac{S(S-1)}{2}$ pairs of classifiers and compare the results on the test set pixels of the hyperspectral cube.

5. Experimental Results

This section presents the results obtained by two parallel SVM approaches, in both multispectral and hyperspectral image classification. According to our previous works [12], the gain of the boss-worker approach is more noticeable when working with high dimensional (e.g. hyperspectral) samples, and when exploiting the spatial information, as it works in an image region-basis. In the second case, the PSVM via Cholesky decomposition is general enough to work with any kind of data, but it is more appropriate for moderate pixel dimensionality, as the bottleneck is the kernel computation with Cholesky decomposition. In addition, including textural or spatial information must be done through specific kernel tricks [47], which not always lead to smarter (faster) implementations.

5.1. Multispectral Image Classification

In this section we present the results obtained with the parallel implementation of SVM via Cholesky factorization in a complex multispectral image classification scenario.

5.1.1. The Mare Nostrum facility

MareNostrum comprises 2560 JS21 compute nodes (blades) and 42 p615 servers. Every blade has two processors at 2.3 GHz running Linux operating system with 8 GB of memory RAM and 36 GB local disk storage. All the servers provide a total of 280 TB of disk storage accessible from every blade through GPFS (Global Parallel File System). The networks that interconnect the MareNostrum are: (1) *Myrinet Network*: High bandwidth network used by parallel applications communications; and (2) *Gigabit Network:* Ethernet network used by the blades to mount remotely their root file system from the servers and the network over which GPFS works. More information is available at `http://www.bsc.es/`.

5.1.2. Data collection

In this experiment, we try to detect cloud presence in multispectral images. Experiments were carried out using two MERIS Level 1b (L1b) images taken over Barrax (Spain),

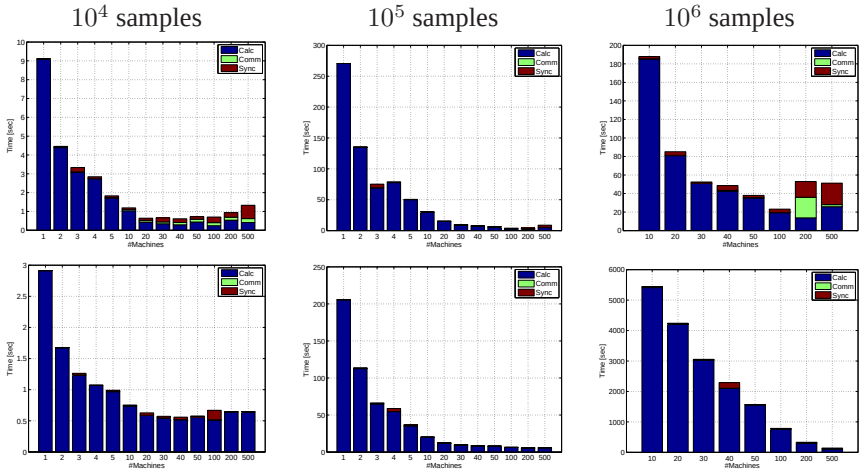

Figure 3. Cost a function of the number of training and test samples (10^4, 10^5, 10^6) and detailed in 'synchronization', 'communmication' and 'computation' for a number of processors between 1 and 500 used.

which are part of the data acquired in the framework of the SPARC 2003 and 2004 ESA campaigns (ESA-SPARC Project). These images were acquired on July 14th of two consecutive years (2003-07-14 and 2004-07-14). For our experiments, we used as input 13 spectral bands (MERIS bands 11 and 15 were removed since they are affected by atmospheric absorptions) and 6 physically-inspired features extracted from MERIS bands in a previous work [60]: cloud brightness and whiteness in the visible (VIS) and near-infrared (NIR) spectral ranges, along with atmospheric oxygen and water vapour absorption features. Cloud presence is considered as the target class. Cloud screening is specially well suited to semi-supervised approaches since cloud features change to a great extent depending on the cloud type, thickness, transparency, height, and background (being extremely difficult to obtain a representative training set); and cloud screening must be carried out before atmospheric correction (being the input data affected by the atmospheric conditions). In the selected image, the presence of snow can be easily confused with clouds, which increases the difficulty of the problem.

5.1.3. Scalability, speed up and overheads

Scalability experiments were run with three large datasets obtained from randomly subsampling the MERIS image ($\{10^4, 10^5, 10^6\}$ samples). Note that, for the case $n = 10^6$, a single machine cannot store the factorized matrix $\mathbf{H}$ in local memory, so we show results for the cases $m > 10$. The running time consists of three parts: computation ('Comp'), communication ('Comm'), and synchronization ('Sync'). Figure 3 show the scores for different number of machines and the three data sizes. The PSVM provides excellent performance and achieves a steady state in computation cost for $m > 20$. For more than 100 machines, the performance is deteriorated. In this case, specially important for $m > 200$, smaller problems *per* machine have to be solved, which results in an overall increase of support vectors and thus both training and test processing time increases.

Figure 4 shows the speedup curves for different data sizes, along with the linear speedup line. This parallel version of the SVM cannot achieve linear speedup beyond a limit which depends on the number of machines and the size of the datast. This result

was already reported in [14]. The fact that the problem is split in many machines also increases the time needed for communication and synchronization overheads. Communication time is incurred when message passing takes place between machines. Synchronization overhead is incurred when the master machine waits for task completion on the slowest machine. Note that, in homogeneous schemes, there are not obviously 'slower' machines but harder tasks to be performed, as in our case. The computation speedup becomes sublinear when adding machines beyond a threshold (around 50 machines). This is due to the fact that the algorithm computes the inverse of a matrix whose size depends on the number of machines m, but fortunately, the larger the dataset is, the smaller is this (unparallelizable) time fraction. Therefore, more machines (larger m) can be employed for larger datasets (larger n) to gain speedup. Finally, we should note that the highest impact on speed up is the communication overhead, rather than the synchronization.

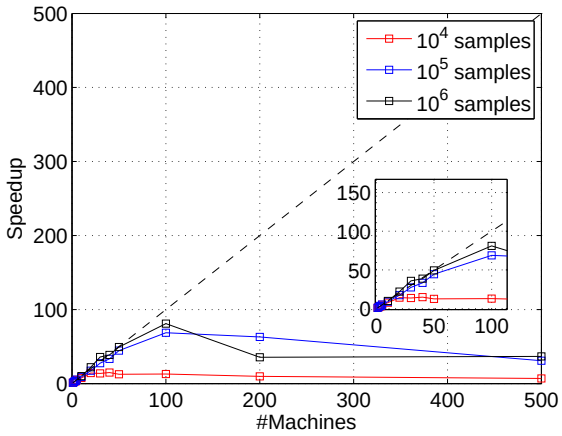

Figure 4. Speedup for different data sizes.

5.2. Hyperspectral Image Classification

In this section we present the results obtained with the master-worker parallel implementation of SVM in a standard hyperspectral image classification scenario.

5.2.1. The Medusa Cluster

The Medusa cluster at NASA's Goddard Space Flight Center is is a 64-node, 128-processor, 1.2 GHz AMD Athlon cluster with 1 GB of memory per node and 2.3 TB of total disk storage. Each node is connected to the others with dual-port Myrinet. Along with the 128-processor compute core, Medusa has more than a dozen workstations throughout a building at GSFC attached to the core with 2 Ghz optical fibre Myrinet. The experiments reported in this work were run from one of those The *master* nodes, i.e. a Linux PC with a single 1.2 GHz AMD Athlon processor and 1.5GB memory, which is connected to the Medusa cluster via fiber Gigabit Ethernet. Typically, the user logs into one of such nodes and performs all calculations from there. The operating system is Fedora Core.

Table 2. Processing time in seconds on the Medusa cluster for training (finding the support vectors), loading the support vectors for prediction, prediction (applying the support vectors from every pair to hyperspectral data to get a pair prediction vote), classification (totaling the votes), total time, and speedup for various numbers of processors using the whole AVIRIS Indian Pines scene.

♯CPUs	2	4	6	8	10	12	14	16	18	20	30	40	50
Time for training	104	57	50	35	35	34	34	35	35	35	43	53	56
Time load predict	20	21	25	15	15	16	16	17	19	20	29	37	45
Time predict	156	65	48	40	36	36	34	33	32	31	31	32	32
Time classify	1	1	<1	<1	1	<1	<1	<1	1	1	<1	1	1
Time total	281	144	123	90	87	86	84	85	87	87	103	123	134
Speedup	1.0	2.0	2.3	3.1	3.2	3.3	3.3	3.3	3.2	3.2	2.7	2.3	2.1

5.2.2. Data collection

The Indian Pines scene was gathered by the AVIRIS instrument in 1992. It consists of 145×145 pixels and 16 ground-truth classes, ranging from 20 to 2468 pixels in size. It was acquired over a mixed agricultural/forested region in NW Indiana. The data set represents a very challenging land-cover classification scenario, in which the primary crops of the area (mainly corn and soybeans) were very early in their growth cycle, with only about 5% canopy cover. Discriminating among the major crops under this circumstances can be very difficult (in particular, given the moderate spatial resolution of 20 meters). This fact has made the scene a challenging benchmark to validate classification accuracy of hyperspectral imaging algorithms. The data is available online from http://dynamo.ecn.purdue.edu/̃biehl/MultiSpec. We removed 20 noisy bands covering the region of water absorption, and worked with 200 spectral bands.

5.2.3. Accuracy

Table 2 reports parallel performance results of the proposed parallel SVM approach on the Medusa cluster. As shown by the table, the time for data movement between processors can be substantial. Specifically, the most significant times reported in the table correspond to the following operations:

1. Moving all the training vectors from the master to every worker at the beginning of the training phase, which results in a constant time of around 19 seconds (see row *Time for training* in Table 2).
2. Moving all the support vectors and the hyperspectral cube from the master to each worker at the beginning of the prediction phase, which varies in time from 20–45 seconds (see row *Time load predict* in Table 2).

The times for moving the support vectors found in the training phase (known as 'the model') from the workers back to the master, is less than one second, and the time for moving the resulting pair classification from the workers to the master is on the order of one or two seconds. The final classification phase of gathering the votes from each of the pair results takes on the order of a second for all 120 pairs results and is performed sequentially on a single processor.

As shown in Table 2, the speedup (performance gain with regards to using one processor) is not linear, but grows and then levels out at around 14 processors with a value of 3.3, and then declines for more than 16 processors. The decline is due to the limited communication bandwidth among the processors, i.e., as the number of processors in-

creases there will be more data collision and thereby delays. The saturation at 3.3 is due to there being a wide distribution of processing times in the training phase which depends on the number of training vectors for each pair classification, and a wide distribution of processing times for the prediction phase depending on the number of support vectors.

5.2.4. Remarks

The slowest pairs come to dominate the time to completion and adding more processors does not offer further speedup. If the number of processors is less than the number of pairs to be computed, then careful arranging of which pairs go to which processors can offer speedup, albeit at the cost of having to plan in advance on which processors particular pairs are to be computed. We believe that significant improvements in the proposed parallel implementation can also be obtained by requiring data movement of only the data needed by the workers, rather than making all the data available to all the workers. Also, by combining the training and prediction phases into a single process for each worker, we could eliminate extra data movement.

6. Conclusions and future work

In this chapter we evaluated the performance of two parallel versions of the popular SVM in the context of remote sensing image classification. The first parallel SVM considered in this work is based on the efficient solution of the IPM problem. The second approach adopts a master-worker strategy and a smart image treatment. In both cases, the same optimality criterion is pursued: to maximize speedup while also considering productivity. Our experimental results, conducted using a well-known hyperspectral data set with accurate ground-truth, provide insightful remarks about parallelization of SVM-based problems in two types of massively parallel platforms: the MareNostrum supercomputer (one of the most powerful cluster computers in Europe) and the Medusa cluster at NASA's Goddard Space Flight Center in Maryland. For instance, our study reveals that simple master-worker models can be effective for parallel implementation of SVM problems, not only due to the *embarrasingly parallel* nature of such algorithms, but also because they can reduce sequential computations at the master node and involve only minimal communication between the parallel tasks, namely, at the beginning and ending of such tasks. However, further work is required in order to improve the scalability of the proposed methods to a very large number of processors, as demonstrated by the experimental results reported in this paper. This, along with a better characterization of the spatial variability of spectral signatures in the data, will be the focus of our future research developments in this field.

Acknowledgements

The authors would like to acknowledge the personnel at the 'Barcelona Supercomputing Center' for supporting this research activity through the project 'High Performance Computing for Earth Observation-Based Hyperspectral Imaging Applications'.

This paper has been partially supported by the Spanish Ministry for Education and Science under projects CICYT AYA2008-05965-C04-03 and CONSOLIDER CSD2007-

00018. The authors wish also to thank L. Gómez-chova for pre-processing the cloud images and make them available for this work.

References

[1] G. Camps-Valls and L. Bruzzone. Kernel-based methods for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 43(6):1351–1362, June 2005.

[2] J. Muñoz Marí, L. Bruzzone, and G. Camps-Valls. A support vector domain description approach to supervised classification of remote sensing images. *IEEE Transactions on Geoscience and Remote Sensing*, 45(8):2683–2692, 2007.

[3] G. Camps-Valls and L. Bruzzone, editors. *Kernel methods for Remote Sensing Data Analysis*. Wiley & Sons, UK, Dec 2009.

[4] J. A. Richards and Xiuping Jia. *Remote Sensing Digital Image Analysis. An Introduction*. Springer-Verlag, Berlin, Heidenberg, 3rd edition, 1999.

[5] G. F. Hughes. On the mean accuracy of statistical pattern recognizers. *IEEE Transactions on Information Theory*, 14(1):55–63, 1968.

[6] K. Fukunaga and R. R. Hayes. Effects of sample size in classifier design. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(8):873–885, 1989.

[7] V. N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, New York, 1998.

[8] B. Schölkopf and A. Smola. *Learning with Kernels – Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press Series, 2002.

[9] G. Camps-Valls, J. L. Rojo-Álvarez, and M. Martínez-Ramón, editors. *Kernel Methods in Bioengineering, Signal and Image Processing*. Idea Group Publishing, Hershey, PA (USA), Nov. 2007. ISBN: 1-559904-042-5.

[10] G. Camps-Valls, L. Gómez-Chova, J. Calpe, E. Soria, J. D. Martín, L. Alonso, and J. Moreno. Robust support vector method for hyperspectral data classification and knowledge discovery. *IEEE Transactions on Geoscience and Remote Sensing*, 42(7):1530–1542, July 2004.

[11] J. Brazile, M. E. Schaepman, D. Schläpfer, J. W. Kaiser, J. Nieke, and K. I. Itten. Cluster versus grid for large-volume hyperspectral image preprocessing. In H.-L. A. Huang and H. J. Bloom, editors, *Atmospheric and Environmental Remote Sensing Data Processing and Utilization: an End-to-End System Perspective. Edited by Huang, Hung-Lung A.; Bloom, Hal J. Proceedings of the SPIE, Volume 5548, pp. 48-58 (2004).*, volume 5548 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pages 48–58, October 2004.

[12] J. A. Gualtieri. A parallel processing algorithm for remote sensing classification. Technical report, Summaries of the Airborne Earth Science Workshop, Pasadena, USA, 2004. Available: http://aviris.jpl.nasa.gov/html/aviris/documents.html.

[13] Antonio Plaza, David Valencia, Javier Plaza, and Pablo Martínez. Commodity cluster-based parallel processing of hyperspectral imagery. *J. Parallel Distrib. Comput.*, 66(3):345–358, 2006.

[14] Edward Chang, Kaihua Zhu, Hao Wang, Hongjie Bai, Jian Li, Zhihuan Qiu, and Hang Cui. Parallelizing support vector machines on distributed computers. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 257–264. MIT Press, Cambridge, MA, 2008.

[15] A. Plaza and C.-I Chang. *High Performance Computing in Remote Sensing*. Chapman & Hall/CRC Press, Boca Raton, FL, 2007.

[16] A. Plaza and C.-I Chang. Special issue on high performance computing for hyperspectral imaging. *International Journal of High Performance Computing Applications*, 22, 2008.

[17] Cheng T. Chu, Sang K. Kim, Yi A. Lin, Yuanyuan Yu, Gary R. Bradski, Andrew Y. Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. In Bernhard Schölkopf, John C. Platt, and Thomas Hoffman, editors, *NIPS*, pages 281–288. MIT Press, 2006.

[18] A. Plaza and C.-I Chang. Clusters versus FPGA for parallel processing of hyperspectral imagery. *International Journal of High Performance Computing Applications*, 22:366–385, 2008.

[19] A. Plaza, D. Valencia, and J. Plaza. An experimental comparison of parallel algorithms for hyperspectral analysis using homogeneous and heterogeneous networks of workstations. *Parallel Computing*, 34:92–114, 2008.

[20] A. Plaza. Parallel techniques for information extraction from hyperspectral imagery using heterogeneous networks of workstations. *Journal of Parallel and Distributed Computing*, 68:93–111, 2008.

[21] J. Brazile, R. Richter, D. Schläpfer, M. E. Schaepmand, and K. I. Itten. Cluster versus grid for operational generation of ATCOR MODTRAN-based look up tables. *Parallel Computing*, 34:32–46, 2008.

[22] J. Setoain, M. Prieto, C. Tenllado, and F. Tirado. GPU for parallel on-board hyperspectral image processing. *International Journal of High Performance Computing Applications*, 22:424–437, 2008.

[23] B. Catanzaro, N. Sundaram, and K. Keutzer. Fast support vector machine training and classification on graphics processors. In *ACM International Conference Proceeding Series; Vol. 307 archive Proceedings of the 25th international conference on Machine learning*, 2008.

[24] Bryan Catanzaro, Narayanan Sundaram, and Kurt Keutzer. Fast support vector machine training and classification on graphics processors. Technical Report 11, EECS Department, University of California, Berkeley, February 2008.

[25] T-N Do, V-H Nguyen, and F. Poulet. A fast parallel SVM algorithm for massive classification tasks. In *Modelling, Computation and Optimization in Information Systems and Management Sciences. Second International Conference MCO 2008*, Metz, France, Sep 2008.

[26] T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with application in pattern recognition. *IEEE Transactions on Electronic Computers*, 14:326–334, June 1965.

[27] R. Courant and D. Hilbert. *Methods of Mathematical Physics*. Interscience Publications. John Wiley, New York, USA, 1953.

[28] A. Aizerman, E. M. Braverman, and L. I. Rozoner. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[29] V. Vapnik and A. Chervonenkis. A note on one class of perceptrons. *Automation and Remote Control*, 25, 1964.

[30] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, USA, 2nd edition, 2000.

[31] J. A. Gualtieri and R. F. Cromp. Support vector machines for hyperspectral remote sensing classification. In *Proceedings of the SPIE, 27th AIPR Workshop*, pages 221–232, February 1998.

[32] J. A. Gualtieri, S. R. Chettri, R. F. Cromp, and L. F. Johnson. Support vector machine classifiers as applied to AVIRIS data. In *Proceedings of The 1999 Airborne Geoscience Workshop*, February 1999.

[33] Sanjay Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.

[34] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In J. Principe, L. Gile, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII – Proceedings of the 1997 IEEE Workshop*, pages 276–285, New York, 1997. IEEE.

[35] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208, Cambridge, MA, 1999. MIT Press.

[36] S.V.M. Vishwanathan and M. Narasimha Murty. SSVM: a simple SVM algorithm. In *Proceedings of the 2002 International Joint Conference on Neural Networks, 2002. IJCNN '02.*, pages 2393–2398, 2002.

[37] Antoine Bordes, Seyda Ertekin, Jason Weston, and Léon Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, September 2005.

[38] Sören Sonnenburg, Gunnar Rätsch, Christin Schäfer, and Bernhard Schölkopf. Large scale multiple kernel learning. *Journal of Machine Learning Research*, 7:1531–1565, July 2006.

[39] Olivier Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19(5):1155–1178, 2007.

[40] L. Qian and T. Hung. Parallel SVM for large data-set mining. In *Fourth International Conference on Data Mining*, pages 661–670, COPPE, Federal University of Rio de Janeiro, Brazil, 2003. WIT Press.

[41] Jian-Pei Zhang, Zhong-Wei Li, and Jing Yang. A parallel SVM training algorithm on large-scale classification problems. In *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, volume 3, pages 1637–1641, 2005.

[42] Hans P. Graf, Eric Cosatto, Léon Bottou, Igor Dourdanovic, and Vladimir Vapnik. Parallel support vector machines: The cascade SVM. In *NIPS*, 2004.

[43] Yumao Lu and Vwani Roychowdhury. Parallel randomized support vector machine. In *Advances in knowledge discovery and data mining*, volume 3918, pages 205–214, Singapore, 2006. Springer Berlin.

[44] Tatjana Eitrich, Wolfgang Frings, and Bruno Lang. HyParSVM – a new hybrid parallel software for support vector machine learning on SMP clusters. In *Euro-Par 2006 Parallel Processing*, pages 350–359. Springer Berlin, 2006.

[45] Tatjana Eitrich and Bruno Lang. Parallel cost-sensitive support vector machine software for classification. In *Neural Information Processing Systems Workshop*, volume 34, pages 141–144, Cambridge, MA, 2006. MIT Press.

[46] L.J. Cao, S.S. Keerthi, Chong-Jin Ong, J.Q. Zhang, U. Periyathamby, Xiu Ju Fu, and H.P. Lee. Parallel sequential minimal optimization for the training of support vector machines. *Neural Networks, IEEE Transactions on*, 17:1039–1049, 2006.

[47] G. Camps-Valls, L. Gómez-Chova, J. Muñoz-Marí, J. Vila-Francés, and J. Calpe-Maravilla. Composite kernels for hyperspectral image classification. *IEEE Geoscience and Remote Sensing Letters*, 3(1):93–97, Jan 2006.

[48] G. Camps-Valls, L. Gómez-Chova, M. Muñoz Marí, J.and Martínez-Ramón, and J. L. Rojo-Álvarez. Kernel-based framework for multi-temporal and multi-source remote sensing data classification and change detection. *IEEE Transactions on Geoscience and Remote Sensing*, 46(6):1822–1835, Jun 2008.

[49] Gene H. Golub and Charles F. Van Loan. *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)*. The Johns Hopkins University Press, October 1996.

[50] T. Joachims. *Making large-scale support vector machine learning practical*, (http://svmlight.joachims.org/), pages 169–184. MIT Press, Cambridge, MA, 1998.

[51] Chih C. Chang and Chih J. Lin. *LIBSVM: a library for support vector machines*, (http://www.csie.ntu.edu.tw/~cjlin/libsvm/), 2001.

[52] Ron Brightwell, Lee Ann Fisk, David S. Greenberg, Trammell Hudson, Mike Levenhagen, Arthur B. Maccabe, and Rolf Riesen. Massively parallel computing using commodity components. *Parallel Computing*, 26(2-3):243–266, 2000.

[53] Andrew Funk, Victor Basili, Lorin Hochstein, and Jeremy Kepner. Application of a development time productivity metric to parallel software development. In *SE-HPCS '05: Proceedings of the second international workshop on Software engineering for high performance computing system applications*, pages 8–12, New York, NY, USA, 2005. ACM.

[54] E. Alba, F. Almeida, M. Blesa, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, J. GonzÃ¡lez, C. León, L. Moreno, J. Petit, J. Roda, A. Rojas, and F. Xhafa. Efficient parallel LAN/WAN algorithms for optimization. the MALLBA project. *Parallel Computing*, 32(5-6):415–440, Jun 2006.

[55] Shai Fine, Katya Scheinberg, Nello Cristianini, John Shawe-taylor, and Bob Williamson. Efficient svm training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2001.

[56] F. Bach and M. Jordan. Predictive low-rank decomposition for kernel methods. In *Proceedings of the Twenty-second International Conference on Machine Learning (ICML), 2005.*, 2005.

[57] B. Wilkinson and A. Michael. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Upper Saddle River, NJ: Prentice Hall, second edition, 2005.

[58] Frank Seinstra, Dennis Koelma, and Jan-Mark Geusebroek. A software architecture for user transparent parallel image processing on MIMD computers. *Lecture Notes in Computer Science*, 2150:653, 2001.

[59] C. W. Hsu and C. J. Lin. A comparison of methods for multiclass support vector machines. *International Journal of Neural Networks*, 13:415–425, March 2002.

[60] L. Gómez-Chova, G. Camps-Valls, J. Calpe, L. Guanter, and J. Moreno. Cloud screening algorithm for ENVISAT/MERIS multispectral images. *IEEE Transactions on Geoscience and Remote Sensing*, 45(12), Dec 2007.