

Some Improvements to a Parallel Decomposition Technique for Training Support Vector Machines

Thomas Serafini¹, Luca Zanni¹, and Gaetano Zanghirati²

¹ Department of Mathematics, University of Modena and Reggio Emilia

² Department of Mathematics, University of Ferrara

Abstract. We consider a parallel decomposition technique for solving the large quadratic programs arising in training the learning methodology Support Vector Machine. At each iteration of the technique a subset of the variables is optimized through the solution of a quadratic programming subproblem. This inner subproblem is solved in parallel by a special gradient projection method. In this paper we consider some improvements to the inner solver: a new algorithm for the projection onto the feasible region of the optimization subproblem and new linesearch and steplength selection strategies for the gradient projection scheme. The effectiveness of the proposed improvements is evaluated, both in terms of execution time and relative speedup, by solving large-scale benchmark problems on a parallel architecture.

Keywords: support vector machines, quadratic programs, decomposition techniques, gradient projection methods, parallel computation

1 Introduction

Support Vector Machines (SVMs) are an effective learning technique [13] which received increasing attention in the last years. Given a training set of labelled examples

$$D = \{(\mathbf{z}_i, y_i), i = 1, \dots, n, \quad \mathbf{z}_i \in \mathbb{R}^m, y_i \in \{-1, 1\}\},$$

the SVM learning methodology performs classification of new examples $\mathbf{z} \in \mathbb{R}^m$ by using a decision function $F : \mathbb{R}^m \rightarrow \{-1, 1\}$, of the form

$$F(\mathbf{z}) = \text{sign} \left(\sum_{i=1}^n x_i^* y_i K(\mathbf{z}, \mathbf{z}_i) + b^* \right), \quad (1)$$

where $K : \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}$ denotes a special kernel function (linear, polynomial, Gaussian, ...) and $\mathbf{x}^* = (x_1^*, \dots, x_n^*)^T$ is the solution of the convex quadratic programming (QP) problem

$$\begin{aligned} \min f(\mathbf{x}) &= \frac{1}{2} \mathbf{x}^T G \mathbf{x} - \mathbf{x}^T \mathbf{1} \\ \text{sub. to } \mathbf{y}^T \mathbf{x} &= 0, \quad 0 \leq x_j \leq C, \quad j = 1, \dots, n, \end{aligned} \quad (2)$$

ALGORITHM PGPD (Parallel SVM Decomposition Technique)

1. Let $\mathbf{x}^{(0)}$ be a feasible point for (2), let n_{sp} and n_c be two integer values such that $n \geq n_{sp} \geq n_c \geq 2$, n_c even, and set $i = 0$. Arbitrarily split the indices $\{1, \dots, n\}$ into the set B of *basic* variables, with $\#B = n_{sp}$, and the set $N = \{1, \dots, n\} \setminus B$ of *nonbasic* variables. Arrange the arrays $\mathbf{x}^{(i)}$, \mathbf{y} and G with respect to B and N :

$$\mathbf{x}^{(i)} = \begin{bmatrix} \mathbf{x}_B^{(i)} \\ \mathbf{x}_N^{(i)} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} \mathbf{y}_B \\ \mathbf{y}_N \end{bmatrix}, \quad G = \begin{bmatrix} G_{BB} & G_{BN} \\ G_{NB} & G_{NN} \end{bmatrix}.$$

2. **Compute in parallel** the Hessian matrix G_{BB} of the subproblem

$$\min_{\mathbf{x}_B \in \Omega_B} f_B(\mathbf{x}_B) = \frac{1}{2} \mathbf{x}_B^T G_{BB} \mathbf{x}_B - \mathbf{x}_B^T (\mathbf{1} - G_{BN} \mathbf{x}_N^{(i)}) \quad (3)$$

where $\Omega_B = \{\mathbf{x}_B \in \mathbb{R}^{n_{sp}} \mid \mathbf{y}_B^T \mathbf{x}_B = -\mathbf{y}_N^T \mathbf{x}_N^{(i)}, \mathbf{0} \leq \mathbf{x}_B \leq C\mathbf{1}\}$ and **compute in parallel** the solution $\mathbf{x}_B^{(i+1)}$ of the above problem. Set $\mathbf{x}^{(i+1)} = \begin{bmatrix} \mathbf{x}_B^{(i+1)T} & \mathbf{x}_N^{(i)T} \end{bmatrix}^T$.

3. **Update in parallel** the gradient $\nabla f(\mathbf{x}^{(i+1)}) = \nabla f(\mathbf{x}^{(i)}) + [G_{BB} \ G_{BN}]^T (\mathbf{x}_B^{(i+1)} - \mathbf{x}_B^{(i)})$ and terminate if $\mathbf{x}^{(i+1)}$ satisfies the KKT conditions for problem (2).
 4. Update B by changing at most n_c elements through the strategy described in [12]. Set $i \leftarrow i + 1$ and go to step 2.
-

where G has entries $G_{ij} = y_i y_j K(\mathbf{z}_i, \mathbf{z}_j)$, $i, j = 1, 2, \dots, n$, $\mathbf{1} = (1, \dots, 1)^T$ and C is a parameter of the SVM algorithm. Once the vector \mathbf{x}^* is computed, $b^* \in \mathbb{R}$ in (1) is easily derived. The matrix G is generally dense and in many real-world applications its size is very large ($n \gg 10^4$). Thus, strategies suited to exploit the special features of the problem become a need, since standard QP solvers based on explicit storage of G cannot be used. Among these strategies, decomposition techniques are certainly the most investigated (see for instance [1, 2, 4, 5] and the references therein). They consist in splitting the original problem into a sequence of QP subproblems sized $n_{sp} \ll n$ that can fit into the available memory. An effective parallel decomposition technique is proposed in [14] (see also [11]). It is based on the Joachims' decomposition idea [4] and on a special variable projection method [7, 8] as QP solver for the inner subproblems. In contrast with other decomposition algorithms, that are tailored for very small-size subproblems (typically less than 10^2), the proposed technique is appropriately designed to be effective with subproblems large enough (typically more than 10^3) to produce few decomposition steps. Due to the effectiveness of the inner QP solver, this method is well comparable with the most effective decomposition approaches on scalar architectures. However, its straightforward parallelization is an innovative feature. In fact, the expensive tasks (kernel evaluations and QP subproblems solution) of the few decomposition steps can be efficiently performed in parallel and promising computational results are reported in [14]. The new version of this approach, equipped with the gradient projection method GVPM introduced in [10] as inner QP solver and with the acceleration strategies suggested in [12], is

called Parallel Gradient Projection-based Decomposition Technique (PGPDT) and its main steps are summarized in Algorithm PGPDT .

In this paper we examine two improvements to the PGPDT. The improvements are suggested by the recent work [3] where a new algorithm for computing the projection of a vector onto the feasible region of the SVM QP problem and a special nonmonotone gradient projection method for (3) are proposed. We will show that these strategies give rise to an efficient inner QP solver for the decomposition techniques and imply better performance for PGPDT, both in terms of execution time and relative speedup.

2 About Gradient Projection-type Inner Solvers

In order to explain the PGPDT improvements we are going to introduce, we need to briefly discuss about the numerical behaviour of the parallel decomposition technique. To this end, we show the PGPDT performance on two well-known benchmark problems: the QP problem sized $n = 60000$ arising when a classifier for digit “8” is trained through a Gaussian SVM ($K(\mathbf{z}_i, \mathbf{z}_j) = e^{-\|\mathbf{z}_i - \mathbf{z}_j\|^2 / (2\sigma^2)}$, $\sigma = 1800$, $C = 10$) on the MNIST database of handwritten digits¹ and the QP problem sized $n = 49749$ arising in training Gaussian SVM ($\sigma = \sqrt{10}$, $C = 5$) on the Web data set². We solve these problems by PGPDT on a IBM CLX/768, a Linux Cluster equipped with Intel Xeon 3GHz processors and 1GB of memory per processor. Table 1 shows the results obtained for different numbers of processing elements (PEs): the *time* column shows the overall training time, sp_r is the relative speedup and *it* is the number of decomposition iterations. t_{prep} , t_{solv} and t_{grad} are respectively the time for computing the Hessian G_{BB} of the subproblem (step 2.), to solve the subproblem (step 2.) and to update the gradient (step 3.). Finally, it_{in} shows the total number of iterations of the inner QP solver and t_{fix} is the execution time of the non-parallelized code, which is obtained by $t_{\text{fix}} = \text{time} - t_{\text{prep}} - t_{\text{solv}} - t_{\text{grad}}$, i.e. by subtracting the execution time of the parallelized parts from the total execution time. It can be observed that t_{fix} is very small compared to the total execution time, and this shows that steps 2 and 3 are the core computational tasks. This table shows satisfactory speedups on the MNIST data set; for 2 and 4 processing elements we even have a superlinear behaviour due to the increased amount of memory available for caching the elements of G . For the Web data set, the speedup sp_r is not as good as for the MNIST case. This can be explained as follows: the gradient updating (t_{grad}) has always a good speedup in both the data sets, while the inner solver is not able to achieve the same good speedup. In the MNIST-like data sets, where $t_{\text{grad}} \gg t_{\text{solv}}$, the suboptimal speedup of the inner solver is compensated by the good speedup of the gradient updating, and the overall behaviour is good. On the other hand, when $t_{\text{grad}} < t_{\text{solv}}$, as for the Web data set, the speedup of the inner solver becomes the main bottleneck.

¹ Available at <http://yann.lecun.com/exdb/mnist>.

² Available at <http://research.microsoft.com/~jplatt/smo.html>.

Table 1. PGPDT performance scaling on MNIST ($n = 60000$) and Web ($n = 49749$) data sets

<i>PEs</i>	MNIST set, $n_{sp} = 2000, n_c = 600$								Web set, $n_{sp} = 1500, n_c = 750$							
	<i>time</i>	<i>sp_r</i>	<i>it</i>	<i>t_{prep}</i>	<i>t_{solv}</i>	<i>t_{grad}</i>	<i>t_{fix}</i>	<i>it_{in}</i>	<i>time</i>	<i>sp_r</i>	<i>it</i>	<i>t_{prep}</i>	<i>t_{solv}</i>	<i>t_{grad}</i>	<i>t_{fix}</i>	<i>it_{in}</i>
1	598.2	15	14.6	81.9	501.4	0.3	5961	242.7	23	3.2	182.6	56.6	0.3	17955		
2	242.2	2.5	15	13.6	49.0	179.3	0.3	6091	125.3	1.9	26	3.9	90.1	30.8	0.5	18601
4	129.6	4.6	15	10.0	28.8	90.2	0.6	6203	73.2	3.3	25	3.0	53.6	16.1	0.5	18759
8	75.5	7.9	15	7.9	21.3	45.7	0.6	5731	46.4	5.2	22	2.2	35.7	7.9	0.6	17408
16	43.8	13.7	17	5.5	13.7	24.2	0.4	5955	32.8	7.4	25	1.5	25.9	5.0	0.4	17003

In the following we will introduce an improved inner solver able to reduce the above drawbacks. The inner QP subproblems (3) have the following general form:

$$\min_{\mathbf{w} \in \Omega} \bar{f}(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T A \mathbf{w} + \mathbf{b}^T \mathbf{w} \quad (4)$$

where the matrix $A \in \mathbb{R}^{n_{sp} \times n_{sp}}$ is symmetric and positive semidefinite, $\mathbf{w}, \mathbf{b} \in \mathbb{R}^{n_{sp}}$ and the feasible region Ω is defined by

$$\Omega = \{\mathbf{w} \in \mathbb{R}^{n_{sp}}, \quad \mathbf{0} \leq \mathbf{w} \leq C\mathbf{1}, \quad \mathbf{c}^T \mathbf{w} = d, \quad d \in \mathbb{R}\}. \quad (5)$$

We recall that now the size n_{sp} allows the matrix A to be stored in memory. The special gradient projection method (GVPM) used by PGPDT [10] combines a monotone linesearch strategy with an adaptive steplength selection based on the Barzilai–Borwein rules. Gradient projection methods are appealing approaches for problems (4) since they consist in a sequence of projections onto the feasible region, that are nonexpensive due to the special constraints (5), as it will be described in the next section. As a consequence, the main task of each iteration remains a matrix-vector product for computing $\nabla \bar{f}(\mathbf{w})$ that can be straightforwardly parallelized by a row block-wise distribution of the entries of A .

Recently, Dai and Fletcher [3] have proposed a new gradient projection method for singly linearly constrained QP problems subject to lower and upper bounds. In the computational experiments reported in [3], this method has shown better convergence rate in comparison to GVPM on some medium-scale SVM test problems.

Here we are interested in evaluating the PGPDT performance improvements due to this new inner solver. We recall our implementation of the Dai and Fletcher method in Algorithm DF.

For this method the same considerations given for GVPM about the computational cost per iteration and the parallelization still hold true. Nevertheless, the linesearch step and the steplength selection rule are very different.

The algorithm DF uses an adaptive nonmonotone linesearch in order to allow the objective function value $\bar{f}(\mathbf{w}^{(k)})$ to increase on some iterations. Its main feature is the adaptive updating of the reference function value f_{ref} . The purpose of the updating rule is to cut down the number of times the linesearch is brought

ALGORITHM DF Gradient Projection Method

1. *Initialization.* Let $\mathbf{w}^{(0)} \in \Omega$, $0 < \alpha_{\min} < \alpha_{\max}$, $\alpha_0 \in [\alpha_{\min}, \alpha_{\max}]$, $L = 2$;
set $f_{\text{ref}} = \infty$, $f_{\text{best}} = f_c = \bar{f}(\mathbf{w}^{(0)})$, $\ell = 0$, $k = 0$, $\mathbf{s}^{(k-1)} = \mathbf{y}^{(k-1)} = \mathbf{0}$.
2. *Projection.* Terminate if $\mathbf{w}^{(k)}$ satisfies a stopping criterion; otherwise compute the descent direction

$$\mathbf{d}^{(k)} = P_{\Omega}(\mathbf{w}^{(k)} - \alpha_k(A\mathbf{w}^{(k)} + \mathbf{b})) - \mathbf{w}^{(k)}.$$

3. *Linesearch.*
If ($k = 0$ and $\bar{f}(\mathbf{w}^{(k)} + \mathbf{d}^{(k)}) \geq \bar{f}(\mathbf{w}^{(k)})$) or ($k > 0$ and $\bar{f}(\mathbf{w}^{(k)} + \mathbf{d}^{(k)}) \geq f_{\text{ref}}$)
then

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \lambda_k \mathbf{d}^{(k)}, \quad \text{with} \quad \lambda_k = \arg \min_{\lambda \in [0,1]} \bar{f}(\mathbf{w}^{(k)} + \lambda \mathbf{d}^{(k)});$$

else

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \mathbf{d}^{(k)};$$

end.

4. *Update.* Compute $\mathbf{s}^{(k)} = \mathbf{w}^{(k+1)} - \mathbf{w}^{(k)}$; $\mathbf{y}^{(k)} = A(\mathbf{w}^{(k+1)} - \mathbf{w}^{(k)})$.

If $\mathbf{s}^{(k)T} \mathbf{y}^{(k)} \leq 0$ then

set $\alpha_{k+1} = \alpha_{\max}$;

else

If $\mathbf{s}^{(k-1)T} \mathbf{y}^{(k-1)} \leq 0$ then

$$\text{set } \alpha_{k+1} = \min \left\{ \alpha_{\max}, \max \left\{ \alpha_{\min}, \frac{\mathbf{s}^{(k)T} \mathbf{s}^{(k)}}{\mathbf{s}^{(k)T} \mathbf{y}^{(k)}} \right\} \right\};$$

else

$$\text{set } \alpha_{k+1} = \min \left\{ \alpha_{\max}, \max \left\{ \alpha_{\min}, \frac{\mathbf{s}^{(k)T} \mathbf{s}^{(k)} + \mathbf{s}^{(k-1)T} \mathbf{s}^{(k-1)}}{\mathbf{s}^{(k)T} \mathbf{y}^{(k)} + \mathbf{s}^{(k-1)T} \mathbf{y}^{(k-1)}} \right\} \right\};$$

end.

end.

If $\bar{f}(\mathbf{w}^{(k+1)}) < f_{\text{best}}$ then

set $f_{\text{best}} = \bar{f}(\mathbf{w}^{(k+1)})$, $f_c = \bar{f}(\mathbf{w}^{(k+1)})$, $\ell = 0$;

else

set $f_c = \max \left\{ f_c, \bar{f}(\mathbf{w}^{(k+1)}) \right\}$, $\ell = \ell + 1$;

If $\ell = L$ then

set $f_{\text{ref}} = f_c$, $f_c = \bar{f}(\mathbf{w}^{(k+1)})$, $\ell = 0$;

end.

end.

Set $k \leftarrow k + 1$, and go to step 2.

into play and, consequently, to frequently accept the iterate $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \mathbf{d}^{(k)}$ obtained through an appropriate steplength α_k .

For the steplength updating in DF, the rule

$$\alpha_{k+1} = \frac{\sum_{i=0}^{m-1} \mathbf{s}^{(k-i)T} \mathbf{s}^{(k-i)}}{\sum_{i=0}^{m-1} \mathbf{s}^{(k-i)T} \mathbf{y}^{(k-i)}}, \quad m \geq 1,$$

Table 2. PGPDT performance scaling with DF as inner solver.

<i>PEs</i>	MNIST set, $n_{sp} = 2000, n_c = 600$							Web set, $n_{sp} = 1500, n_c = 750$						
	<i>time</i>	<i>sp_r</i>	<i>it</i>	<i>t_{prep}</i>	<i>t_{solv}</i>	<i>t_{grad}</i>	<i>it_{in}</i>	<i>time</i>	<i>sp_r</i>	<i>it</i>	<i>t_{prep}</i>	<i>t_{solv}</i>	<i>t_{grad}</i>	<i>it_{in}</i>
1	591.1		15	14.8	73.0	503.0	5254	206.5		24	3.2	145.9	57.0	12247
2	232.4	2.5	15	13.7	38.9	178.2	4937	91.2	2.3	23	3.7	59.0	28.1	12068
4	124.9	4.7	15	10.3	23.8	90.5	5061	59.9	3.4	29	3.3	39.6	16.6	13516
8	70.1	8.4	15	7.6	16.2	45.7	5044	40.6	5.1	26	2.4	29.1	8.8	13554
16	41.4	14.3	15	5.1	11.9	24.0	5062	27.3	7.6	23	1.5	20.7	4.7	12933

is used with the choice $m = 2$ because it is observed to be the best one for the SVM QP problems. We conclude the introduction to DF by recalling that its global convergence can be proved by proceeding as in [9].

In order to analyze the behaviour of this new solver within PGPDT, we can refer to Table 2 which shows the same test of Table 1 but using DF as the inner solver, in place of GVPM. Looking at the it_{in} column, one can observe a great reduction of the overall inner iterations; considering that the cost per iteration of the DF method is almost the same as for GVPM, this implies a more efficient solution of the inner subproblem. This fact is confirmed by comparing the t_{solv} columns, which report the total execution time of the inner solvers. As a result, the overall execution time of the PGPDT and its speedup sp_r are improved by the introduction of this new inner solver.

3 About the Projection Onto the Feasible Region

The gradient projection algorithms used as inner solvers in the PGPDT require at each iteration to project a vector onto a feasible region Ω of the form (5). This feasible region has a special structure defined by a single linear equality constraint and box constraints. This section is about methods for computing efficiently a projection onto such a feasible region.

The orthogonal projection of a vector \mathbf{z} onto Ω is the solution $P_\Omega(\mathbf{z})$ of the following separable strictly convex quadratic program:

$$\begin{aligned} \min \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} - \mathbf{z}^T \mathbf{w} \\ \text{sub. to} \quad & \mathbf{0} \leq \mathbf{w} \leq C\mathbf{1}, \quad \mathbf{c}^T \mathbf{w} = d. \end{aligned} \tag{6}$$

By exploiting the KKT conditions of (6) it is possible to prove [3, 6] that $P_\Omega(\mathbf{z})$ can be derived from the solution of the piecewise linear monotone non-decreasing equation in one variable

$$r(\lambda) = \mathbf{c}^T \mathbf{w}(\lambda) - d = 0, \tag{7}$$

where

$$\mathbf{w}(\lambda) = \text{mid}(\mathbf{0}, (\mathbf{z} + \lambda \mathbf{c}), C\mathbf{1})$$

in which $\text{mid}(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$ is the componentwise operation that gives the median of its three arguments. Once the solution λ^* of (7) is computed, we obtain $P_\Omega(\mathbf{z})$ by setting $P_\Omega(\mathbf{z}) = \mathbf{w}(\lambda^*)$.

Table 3. PGPDT performance scaling with DF and secant-based projector.

<i>PEs</i>	MNIST set, $n_{sp} = 2000, n_c = 600$							Web set, $n_{sp} = 1500, n_c = 750$						
	<i>time</i>	<i>sp_r</i>	<i>it</i>	<i>t_{prep}</i>	<i>t_{solv}</i>	<i>t_{grad}</i>	<i>it_{in}</i>	<i>time</i>	<i>sp_r</i>	<i>it</i>	<i>t_{prep}</i>	<i>t_{solv}</i>	<i>t_{grad}</i>	<i>it_{in}</i>
1	590.0		15	14.9	67.4	507.5	5107	200.8		23	3.0	143.4	54.1	12824
2	234.1	2.5	16	14.1	39.4	179.2	5432	99.1	2.0	26	4.0	63.7	30.5	14090
4	121.0	4.9	15	10.1	20.1	89.7	4913	46.6	4.3	23	3.0	28.0	15.2	11477
8	67.7	8.7	16	8.1	13.2	46.1	5004	35.5	5.7	25	2.0	24.6	8.3	14500
16	39.2	15.1	15	5.2	9.6	23.9	5476	22.1	9.1	24	1.5	15.1	5.1	13200

Thus, the main computational task for computing $P_\Omega(\mathbf{z})$ consists in solving the equation (7). The gradient projection methods tested in the previous section solve this root finding problem by the $O(n)$ bisection-like algorithm proposed in [6]. Here, we consider an alternative approach introduced in [3] based on a $O(n)$ secant-type method. In particular, we are interested in evaluating how the use of this new projection strategy within the DF gradient projection method can improve the PGPDT performance. Our implementation of the secant-type algorithm for (7) follows the one proposed in [3] and, since the projection is required at each iteration of the DF scheme, we use to hot-start the algorithm by providing the optimal λ of the previous projection as the initial approximation. Finally, we stop the secant-type algorithm if one of the following conditions is satisfied: $|r(\lambda_i)| < 10^{-10}\sqrt{Cn}$ or $|\Delta\lambda_i| < 10^{-11}(1+|\lambda_i|)$, where λ_i is the current approximation and $\Delta\lambda_i = \lambda_i - \lambda_{i-1}$.

Table 3 shows the performance results of the PGPDT equipped with the DF gradient projection method as inner QP solver and the secant-type algorithm [3] for computing the projections. By comparing Tables 3 and 2, we can evaluate the different behaviour due to the new secant-based projector with respect to the bisection-based projector previously used. First of all, by considering the total number of the inner solver iterations (it_{in}) we may observe a very similar behaviour in terms of convergence rate of the DF gradient projection method. This confirms that the two projectors works with a well comparable accuracy. Furthermore, by comparing the total scalar time, slightly better results are obtained with the new secant-based projector. In spite of these similarities, since the projector is not parallelized and, consequently, it gives rise to a fixed time independent on the number of PEs, the time saving due to the new more efficient projector implies a significantly better speedup of the inner solver and of the overall PGPDT, especially when many PEs are used.

By comparing Tables 3 and 1 it is possible to evaluate the overall improvements of the new strategies presented in this paper which, especially in the case of Web-like data sets, consist in a promising reduction of solution times and an increase of the speedup when many processors are used.

Table 4. Performance of the improved PGPDT on a large-scale data set.

KDDCUP-99, $n_{sp} = 600, n_c = 200$							
<i>PEs</i>	<i>time</i>	<i>sp_r</i>	<i>it</i>	t_{prep}	t_{solv}	t_{grad}	<i>MKer</i>
1	46301		1042	62.0	526.0	45711	170469
2	19352	2.4	1031	56.0	300.0	18994	97674
4	8394	5.5	1003	40.1	177.7	8175	74478
8	4821	9.6	1043	30.4	150.1	4639	77465
16	2675	17.3	1016	27.8	113.6	2532	75423

4 Test on a large data set

In this section we briefly present the performance of the improved PGPDT on a large-size data set: the KDDCUP-99.

The KDDCUP-99 is the Intrusion Detection data set³, used for the Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99. The training set consists in binary TCP dump data from seven weeks of network traffic. Each original pattern has 34 continuous features and 7 symbolic features. We normalize each continuous feature to the range $[0, 1]$, and transform each symbolic feature to integer data and then normalize it to the range $[0, 1]$. The original training set consists in 4898431 examples containing repeated examples. We removed the duplicate data and extracted the first 200000 and the last 200000 examples, thus obtaining a training set of size 400000. We used a Gaussian kernel with parameters $\sigma^2 = 0.625, C = 0.5$ and 512MB of caching area.

Table 4 shows the training results using PGPDT algorithm on the IBM Linux cluster. The column *MKer* counts the millions of total Kernel evaluations, which is the total sum of the kernel evaluations over all the processing elements.

A superlinear speedup can be observed also for the largest number of processors. As we mentioned, this is mainly due to the PGPDT ability to efficiently exploit the larger amount of memory for caching the G 's entries, which yields a significant kernel evaluations reduction. The good scalability shown in these experiments confirms that the proposed improved PGPDT is very suitable to face in parallel large and even huge data sets.

5 Conclusions

In this paper we presented some improvements to the PGPDT decomposition algorithm for training large-scale SVMs. The changes concern the solution of the inner QP subproblems, which is a crucial point for the performance of the PGPDT. We experimentally show that a new gradient projection scheme, based on the adaptive nonmonotone linesearch in combination with an averaged Barzilai-Borwein steplength rule, improves the performance of both the sequential and the parallel version of the code over the monotone gradient projection

³ Available at <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>

solver previously used by PGPDT. Besides, a secant-based projector gives a further improvement with respect to the bisection-based projector currently available in the PGPDT software. The combination of these new strategies gives rise to an improved PGPDT version for large-scale SVM problems, which is available for download at <http://dm.unife.it/gpdt>.

References

1. C.C. Chang, C.J. Lin (2002), LIBSVM: a Library for Support Vector Machines, available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
2. R. Collobert, S. Benjo (2001), SVM Torch: Support Vector Machines for Large-Scale Regression Problems, *Journal of Machine Learning Research* **1**, 143–160.
3. Y.H. Dai, R. Fletcher (2003), New Algorithms for Singly Linearly Constrained Quadratic Programs Subject to Lower and Upper Bounds, Research Report NA/216, Department of Mathematics, University of Dundee.
4. T. Joachims (1998), Making Large-Scale SVM Learning Practical, *Advances in Kernel Methods*, B. Schölkopf *et al.*, eds., MIT Press, Cambridge, MA.
5. C.J. Lin (2001), On the Convergence of the Decomposition Method for Support Vector Machines, *IEEE Transactions on Neural Networks* **12**(6), 1288–1298.
6. P.M. Pardalos, N. Kover (1990), An Algorithm for a Singly Constrained Class of Quadratic Programs Subject to Upper and Lower Bounds, *Math. Programming* **46**, 321–328.
7. V. Ruggiero, L. Zanni (2000), A Modified Projection Algorithm for Large Strictly Convex Quadratic Programs, *J. Optim. Theory Appl.* **104**(2), 281–299.
8. V. Ruggiero, L. Zanni (2000), Variable Projection Methods for Large Convex Quadratic Programs, *Recent Trends in Numerical Analysis*, D. Trigiante, ed., Advances in the Theory of Computational Mathematics 3, Nova Science Publ., 299–313.
9. T. Serafini (2005), Gradient Projection Methods for Quadratic Programs and Applications in Training Support Vector Machines, Ph.D. Thesis, Dept. of Mathematics, University of Modena and Reggio Emilia.
10. T. Serafini, G. Zanghirati, L. Zanni (2005), Gradient Projection Methods for Large Quadratic Programs and Applications in Training Support Vector Machines, *Optim. Meth. and Soft.* **20**, 353–378.
11. T. Serafini, G. Zanghirati, L. Zanni (2004), Parallel Decomposition Approaches for Training Support Vector Machines, *Parallel Computing: Software Technology, Algorithms, Architectures and Applications*, G.R. Joubert, W.E. Nagel, F.J. Peters and W.V. Walter, ed., Advances in Parallel Computing **13**, Elsevier, Amsterdam, The Netherlands, 259–266.
12. T. Serafini, L. Zanni (2005), On the working set selection in Gradient Projection-based Decomposition Techniques for Support Vector Machines, *Optim. Meth. and Soft.*, to appear. (<http://cdm.unimo.it/home/matematica/zanni.luca/>).
13. V.N. Vapnik (1998), *Statistical Learning Theory*, John Wiley and Sons, New York.
14. G. Zanghirati, L. Zanni (2003), A Parallel Solver for Large Quadratic Programs in Training Support Vector Machines, *Parallel Computing* **29**, 535–551.