New parallel support vector regression for predicting building energy consumption

H. X. Zhao, F. Magoules

New Parallel Support Vector Regression for Predicting Building Energy Consumption

Hai-xiang Zhao, Frédéric Magoulès
Applied Mathematics and Systems Laboratory
Ecole Centrale Paris
Châtenay Malabry, France
Email: haixiang.zhao@ecp.fr, frederic.magoules@hotmail.com

***Abstract*—One challenge of predicting building energy consumption is to accelerate model training when the dataset is very large. This paper proposes an efficient parallel implementation of support vector regression based on decomposition method for solving such problems. The parallelization is performed on the most time-consuming work of training, i.e., to update the gradient vector f. The inner problems are dealt by sequential minimal optimization solver. The underlying parallelism is conducted by the shared memory version of Map-Reduce paradigm, making the system particularly suitable to be applied to multi-core and multiprocessor systems. Experimental results show that our implementation offers a high speed increase compared to Libsvm, and it is superior to the state-of-the-art MPI implementation Pisvm in both speed and storage requirement.**

Index Terms*—Support Vector Regression (SVR); Building energy consumption; Parallel computing; Multi-core; Map-Reduce

I. Introduction

Since proposed in 1990s by V.N. Vapnik, support vector machine (SVM) has been a popular supervised learning method of solving classification and regression problems [1]. The support vector regression (SVR) has shown robust generalization ability in the application of predicting building energy consumption [2]–[5]. The essential computation of SVR is to solve a quadratic problem (QP) which is both time and memory costly, causing it a challenge to solve large scale problems. Despite of several optimizing or heuristic methods such as shrinking, chunking [6], kernel caching [7], approximation of kernel matrix [8], sequential minimal optimization (SMO) [9], primal estimated sub-gradient solver [10], a more sophisticated and satisfactory resolution is always expected for this challenging problem. As stated in [5], the building's energy system is extremely complex involving large number of influence factors, making it a common situation that we tackle large scale datasets.

With the development of chip technologies, computers with multi-core or multiprocessor are becoming more available and affordable in modern market. This paper therefore attempts to investigate and demonstrate how SVR can benefit from this modern platform when solving the problem of predicting building energy consumption. A new parallel SVR that is particularly suitable to this platform is proposed. Decomposition method and inner SMO solver compose the main procedure of training. A shared cache is designed to store the kernel columns. For the purpose of achieving easy implementation without sacrificing performance, the new parallel programming framework Map-Reduce is chosen to perform the underlying parallelism. The proposed system is therefore named as MRPsvm (abbreviation of "Map-Reduce parallel SVM"). Comparative experiments are conducted on three simulated energy consumption datasets, showing significant performance improvement in our system compared to Libsvm [11] and Pisvm [12].

The following sections are organized as follows. Section II introduces the principle of SVR training and the decomposition method. Section III explains how MRPsvm is implemented. Section IV states the related work. Section V presents how to prepare the energy consumption datasets and the numerical experiments. Conclusions are drawn in section VI.

II. Support vector regression

A. Principle of ϵ-SVR

Let's present the training data as $(x_1, z_1), ..., (x_l, z_l)$,where vector x_i is the ith sample, z_i is the ith target value corresponding to x_i, l is the number of samples. The goal of *ϵ-SVR* is to find a decision function $g(x)$ that makes the deviation between the calculated targets and the actual targets is at most ϵ [1]. The dual form of the SVR can be written in the following quadratic form:

$$\min_{\alpha} \quad \frac{1}{2}\alpha^T Q\alpha - \sum_{i=1}^{2l} p_i\alpha_i \tag{1}$$

subject to

$$y^T\alpha = 0 \tag{2}$$

$$0 \leq \alpha_i \leq C \quad \forall i = 1, ..., 2l \tag{3}$$

where α is a vector of $2l$ variables, Q is a $2l$ by $2l$ positive semidefinite matrix. Each element of Q has the following form:

$$Q_{ij} = K(x_m, x_n)$$

$$m = \begin{cases} i & \text{if} \quad i \leq l \\ i-l & \text{if} \quad i > l \end{cases}$$
$$n = \begin{cases} j & \text{if} \quad j \leq l \\ j-l & \text{if} \quad j > l \end{cases}$$
$$i, j = 1, ..., 2l$$

where $K(x_i, x_j)$ is called kernel function which can be substituted by any Mercer kernel, the most common ones are radial basis function (RBF), polynomial function, sigmoid function and linear function. The parameter p in (1) is defined as:

$$p_i = \begin{cases} \epsilon + z_i & \text{if} \quad i = 1, ..., l \\ \epsilon - z_i & \text{if} \quad i = l+1, ..., 2l \end{cases} \tag{4}$$

y in the constraint (2) is defined as:

$$y_i = \begin{cases} 1 & \text{if} \quad i = 1, ..., l \\ -1 & \text{if} \quad i = l+1, ..., 2l \end{cases} \tag{5}$$

In the constraint (3), C is the upper bound used to trade off between model performance on training data and its generalization ability. The objective of the problem is to find the solution of α which makes (1) minimized and the constraints (2) (3) fulfilled. After the optimum α is found, the decision function can be formulated as:

$$g(x) = \sum_{i=1}^{l} (-\alpha_i + \alpha_{i+l}) K(x_i, x) + b$$

where b is a constant value which can be easily calculated in the training step. The training samples that satisfy $(-\alpha_i + \alpha_{i+l} \neq 0)$ are called support vectors. It is obvious that only support vectors have contribution to the decision function, this is the reason why the algorithm is called support vector regression.

B. Decomposition Approach

In building energy application, large amounts of datasets are highly available for model analysis. Since the size of the kernel matrix Q is $2l * 2l$, it is difficult to store the whole matrix in memory when l is very large. For the example of 20 buildings which will be described in section V, to store the whole Q for this dataset we need more than 74GB memory which is not affordable by ordinary users. And for the case of 50 buildings, the memory requirement is even larger. Osuna et al. [6] proposed a method to decompose the problem into smaller tasks. In each task, a working set which contains certain parts of α is chosen to be optimized, while the rest of α remains in constant value. The program repeats the select-optimize process iteratively until global optimality conditions are satisfied. In each iteration, only the involved partition of kernel matrix needs to stay in the memory. Let B denote the working set which has n variables and N denote the non-working set which has $(2l - n)$ variables. Then, α, y, Q and p are correspondingly written as:

$$\alpha = \begin{vmatrix} \alpha_B \\ \alpha_N \end{vmatrix}, \quad y = \begin{vmatrix} y_B \\ y_N \end{vmatrix}, \quad Q = \begin{vmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN} \end{vmatrix}, \quad p = \begin{vmatrix} p_B \\ p_N \end{vmatrix}$$

Accordingly, the small task in this case can be written as:

$$\begin{aligned} \min \quad & \frac{1}{2}\alpha_B^T Q_{BB}\alpha_B - \alpha_B^T(p_B - Q_{BN}\alpha_N) \\ & + \frac{1}{2}\alpha_N^T Q_{NN}\alpha_N - \alpha_N^T p_N \end{aligned} \tag{6}$$

subject to

$$\alpha_B^T y_B + \alpha_N^T y_N = 0 \tag{7}$$

$$0 \leq \alpha_B \leq C \tag{8}$$

Since the last term $(\frac{1}{2}\alpha_N^T Q_{NN}\alpha_N - \alpha_N^T p_N)$ of (6) remains constant in each iteration, it can be omitted while calculating, so that function (6) has the same form as function (1). One of the advantages of this decomposition method is that the newly generated task is small enough to be solved by most off-the-shelf methods. In our work, we choose SMO as the inner small task solver due to its relative simplicity, yet high performance characteristics. In fact, SMO is itself an extreme case of the decomposition method where the working set contains only two variables. This kind of binary sub-problem can be easily solved analytically [9] [11]. As stated in [7], the solution of (6) is strictly feasible towards the optimum solution of global problem (1). This feature guarantees the global convergence of the decomposition method.

The Karush-Kuhn-Tucker (KKT) optimality condition is verified through evaluating the gradient of (1), i.e., $f_i = \sum_{j=1}^{2l} \alpha_j Q_{ij} + p_i$ for all $i = 1, ..., 2l$. This procedure can be summarized as follows. First, we classify the training samples into two categories

$$I_{up}(\alpha) = \{i | \alpha_i < C, y_i = 1 \text{ or } \alpha_i > 0, y_i = -1\}$$
$$I_{low}(\alpha) = \{i | \alpha_i < C, y_i = -1 \text{ or } \alpha_i > 0, y_i = 1\}$$

Then we search two extreme values $m(\alpha)$ and $M(\alpha)$:

$$m(\alpha) = \max_{i \in I_{up}(\alpha)} -y_i f_i \tag{9}$$

$$M(\alpha) = \min_{i \in I_{low}(\alpha)} -y_i f_i \tag{10}$$

And then, we define the stop condition as:

$$m(\alpha) - M(\alpha) \leq \epsilon \tag{11}$$

The selection of working set directly influences the speed of convergence. For inner SMO solver, maximal violating pair is selected to be the binary working set according to the second order information [13]. We do not state here how inner SMO solver works since it has been discussed in detail in [9] and [11]. For the selection of working set B, we simply consider the first order information and select, in some sense, the maximal violating pairs as proposed by [14]. Suppose the required size of B is n, we choose q $(q < n)$ variables from α by sequentially selecting pair of variables which satisfy (9) and (10). The remaining $(n - q)$ variables are chosen as those who entered B in the last iteration but not yet selected in current B. The selection of these $(n - q)$ variables follows

the sequence: free variables ($0 < \alpha_i < C$), lower bound variables ($\alpha_i = 0$), upper bound variables ($\alpha_i = C$). The reason for putting restraint on the number of new variables entering the working set is to avoid frequent entering-leaving of certain variables. Otherwise, the speed of convergence would considerably slow down [14].

After the working set is optimized, f is updated by the newly optimized $\alpha_j, \forall j \in B$. This procedure is crucial as it prepares f for the next iteration to do optimality condition evaluation and working set selection. In fact, this is the most computational expensive step in SVR training due to the heavy work of computing Q_{ij}. The updating procedure can be written as follows:

$$f_i^* = f_i + \sum_{j \in B} \Delta\alpha_j Q_{ij} \qquad i = 1, ..., 2l \tag{12}$$

where $\Delta\alpha_j$ is the newly optimized α_j minus the old α_j. The whole decomposition method is summarized in algorithm 1.

Algorithm 1 Decomposition solver of SVR
Input: data set $(x_i, z_i), \forall i \in 1, ..., l$
Initialize: $\alpha_i = 0$, p_i by (4), y_i by (5),
$f_i = p_i$, $\forall i \in 1, ..., 2l$
Calculate: I_{up}, I_{low}, $m(\alpha)$, $M(\alpha)$
Repeat
select working set B until $\mid B \mid = n$
update α_i by SMO solver, $\forall i \in B$
update f_i, $\forall i \in 1, ..., 2l$
calculate I_{up}, I_{low}, $m(\alpha)$, $M(\alpha)$
Until $m(\alpha) - M(\alpha) \leq \epsilon$

III. System implementation

Next, we will introduce some key points of our system implementation, including why and how Map-Reduce is used to do the parallelism, shared caching technique and the difference of our system compared to Pisvm.

A. Parallelizing the QP Solver

Map-Reduce is a new parallel programming framework originally proposed in [15]. It allows users to write code in a functional style: map computations on separated data, generate intermediate key-value pairs and then reduce the summation of intermediate values assigned to the same key. A runtime system is designed to automatically handle low-level mapping, scheduling, parallel processing and fault tolerance. It is a simple, yet very useful framework. It can help people extract parallelism of computations on large datasets by taking advantage of distributed systems.

Problem (12) can be regarded as a summation of several computational expensive terms as shown in the top right corner in Figure 1. Therefore, Map-Reduce is naturally suitable to deal with this problem. The working set B is uniformly decomposed into several small pieces, the calculation of f^* is also divided into several parts in the same manner as for B. Each part is then assigned to a mapper. After the parallel calculations of these mappers, final f^* is summed up by the reducer. Here, $(j_k, k = 1, ..., n)$ is the variable index of working set in kernel matrix, which gives the kth variable in B with its index in Q as j_k. In practice, since some of $\Delta\alpha_i$ are so marginal that they can be omitted, it is not necessary to update f on all of the n variables.

In some recent work, Map-Reduce is proved to be an effective parallel computing framework on multi-core systems. Chu et al. [16] have developed Map-Reduce as a general programming framework on multi-core systems for machine learning applications. Phoenix designed in [17] implements a common API for Map-Reduce. It allows users to easily parallelize their applications without conducting concurrency management. The Map-Reduce tasks are performed in threads on multi-core systems. An efficient integrated runtime system is supposed to handle the parallelization, resource management and fault recovery by itself. This system is adopted as the underlying Map-Reduce handler in our MRPsvm. We show the parallel architecture of one iteration in Figure 1. The small tasks are uniformly distributed to mappers which have the same number of processors. Phoenix serves as the role of creating and managing mappers and reducers, making the system is easy to be implemented.

B. Kernel Matrix Caching

Since the calculation of kernel elements dominates the training work, it is an effective consideration to cache the kernel elements in memory as much as possible. MRPsvm maintains a fix-sized cache which stores recently accessed or generated kernel columns. The cache replacement policy is a simple least-recent-use strategy, the same as that of Libsvm. Only the column currently needed but not hit in the cache will be calculated. All parallel mappers share an unique copy of cache in the shared memory. As a result, the operation of inserting a new column into the cache should be synchronized.

For inner SMO solver, since the kernel matrix size is small enough and is dependent on the size of working set B which is normally set to 1024 according to the knowledge of experience, it is practical to cache the full version of this small matrix.

To reduce the storage requirements, the sample vectors x_i are stored by sparse representation. When calculating a kernel column $(Q_{ij}, i = 1, ..., 2l)$, we need to unroll the jth sample vector to dense format and then calculate the dot products of this vector with all $2l$ sample vectors. There is only one copy of the whole dataset (x, z) in the shared memory.

C. Comparison of MRPsvm with Pisvm

Pisvm also uses decomposition method to train SVM in parallel. It is an efficient tool to analyze multiple buildings' energy behaviors as stated in our previous work [5]. However its implementation is different from our new implementation MRPsvm in many aspects. First, Pisvm is based on MPI implementation and aims at extracting parallelism from distributed memory systems, while our parallel algorithm is conducted

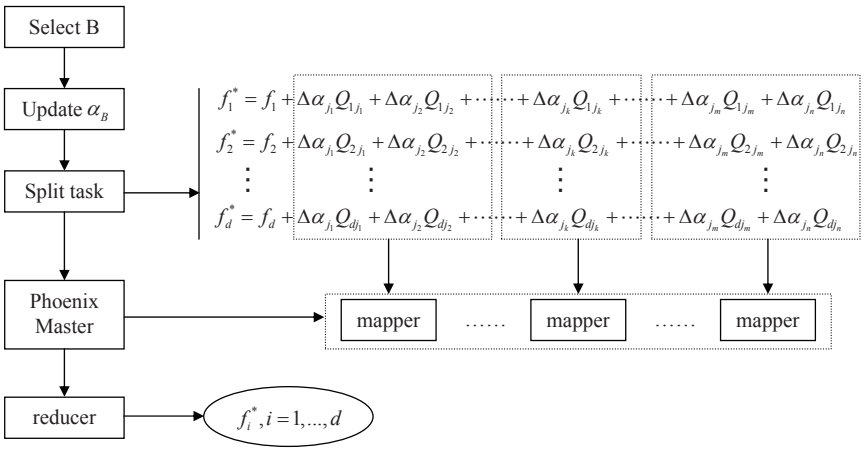

Fig. 1: Architecture of the parallelization in one iteration ($d = 2l$).

by Map-Reduce threads on shared memory system. The two implementations are based on totally different models. Second, in Pisvm, each process stores a copy of data samples, while in the contrary, MRPsvm stores only one copy in the shared memory. This means MRPsvm can save large amount of storage when the dataset is huge. The saved space can be used to cache more kernel matrix in order to further improve training speed. Third, Pisvm adopts a distributed cache strategy in order to share the saved kernel elements among all of the processes. Each process stores locally a piece of the cache. Consequently, the work for updating gradients is divided and assigned globally to proper processors according to the cache locality. In contrast, MRPsvm has only one copy of the cache, and each processor accesses the cache equally, so that the overhead of global assignment is avoided. However, we have to note that synchronization on cache write is required.

In section V, we will compare the performance of Pisvm with that of MRPsvm in the application of building energy prediction, providing direct evidence that our system is more efficient and suitable than the MPI implementation on multi-core systems.

IV. Related work

Several approaches have been proposed to parallelize SVM, mostly for solving classification problem. They can be classified into several categories according to the type of QP solver. Based on stochastic gradient descent method, P-packSVM optimizes SVM training directly on the primal form of SVM for arbitrary kernels [18]. Very high efficiency and competitive accuracy have been achieved. Psvm proposed in [8] is based on interior point QP solver. It approximates the kernel matrix by incomplete Cholesky Factorization. Memory requirement is reduced and scalable performance has been achieved. Bickson et al. [19] solve the problem by Gaussian belief propagation which is a method from complex system domain. The parallel solver brings competitive speedup on large scale problems. The decomposition method attracts more attention than the above solvers. Graf et al. [20] train several SVMs on small data partitions, then they aggregate support vectors from two pair SVMs to form new training samples on which another training is performed. The aggregation is repeated until only one SVM remains. The similar idea is adopted by Dong et al. [21], in their work, sub-SVMs are performed on block diagonal matrices which are regarded as the approximation to the original kernel matrix. Consequently, nonsupport vectors are removed when dealing these sub-problems. Zanni et al. [14] parallelize SVM-light with improved working set selection and inner QP solver. Hazan et al. [22] propose a parallel decomposition solver using Fenchel Duality. Lu et al. [23] parallelize randomized sampling algorithms for SVM and SVR.

Cao et al. [24] and Catanzaro et al. [25] parallelize SMO solver for training SVM for classification. Both work mainly focuses on updating gradient for KKT condition evaluation and the working set selection. The difference between them lies on the implementation details and the programming models. Specifically speaking, the first work is conducted by using MPI on clusters while the second one by Map-Reduce threads on modern GPU platform. In our work, we also adopt SMO algorithm. But we use it as the inner QP solver without any parallel computation, in fact, we perform the parallelization on external decomposition procedure. The main advantage of our coarse-grained parallelism is that it can significantly reduce the burden of overheads since the number of iterations in global decomposition procedure (where $n \gg 2$) is extremely smaller than that of pure SMO algorithm (where $n = 2$). Although both GPU SVM [25] and our system are implemented in threads, we will not compare them in experiments since they are designed for different platforms and GPU SVM is specially used to solve classification problems.

V. Predicting building energy consumption

In this part, we present the application of SVR on predicting building energy consumption. The proposed parallel SVR is compared with Libsvm and Pisvm on three datasets to demonstrate how much performance improvement can be achieved by the new implementation. Libsvm is a widely used sequential implementation while Pisvm is the state-of-the-art parallel

implementation. Although Pisvm is not especially designed for multi-core architecture, we still have good reasons for doing this comparison. Firstly, as the best knowledge as we know, there is no existing parallel implementation of SVR that is specially developed for multi-core systems. Therefore there is a strong need to verify if our system could outperform the parallel implementation designed for distributed memory. Secondly, most of the systems surveyed in section IV are not available to the public, while Pisvm, as a typical parallel implementation of SVM, is easy to obtain. Thirdly, the QP solver of Pisvm is the same as MRPsvm, hence, if we compare our system with Pisvm, the advantage of Map-Reduce framework is more convincing.

A. Energy Consumption Datasets

Three datasets are prepared for the model training. They denote the historical energy consumption of one building, 20 buildings and 50 buildings respectively. The datasets are simulated in Energyplus which is a widely applied, state-of-the-art and effective building energy simulation tool [26].

The buildings are designed as follows. All of them are for office use and located in urban area. We evenly distribute them into five typical cities of France which are Paris-Orly, Marseilles, Strasbourg, Bordeaux and Lyon. As outlined in Figure 2, the five cities vary remarkably in ambient dry bulb temperatures, making the datasets represent energy requirements under five typical weather conditions. Each building has similar structures, i.e., single-story, mass-built, one rectangle room with attic roof and four windows without shading. Electrical equipments including lighting system, fans, water heaters, are scheduled as common office use. In winter season (from November 1st to March 31st), district heating is applied in order to keep the room temperature at a constant level. Ventilation is adopted for indoor thermal comfort. The number of occupants depends on the housing space and people density, with the average of 0.2 people per zone floor area. During the simulation, some input variables such as size, orientation, window area, scheduling, are set differently to achieve diversity among multiple buildings.

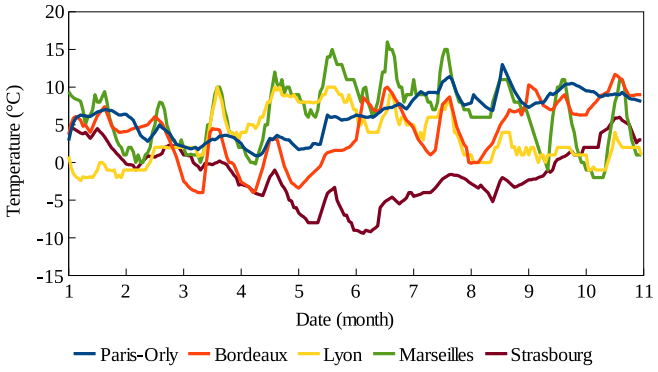

Fig. 2: Dry bulb temperature in the first 11 days of January.

The dataset of one building is hourly energy dynamics in a period of one year. We record 8 important features [27] as shown in Table I. For other two datasets, the recording period is from November to March which is winter season in France, and we record 4 more features which generate the building diversity, i.e., height, length, width and window/wall area ratio.

TABLE I: The features of one building's consumption data.

Features	Unit
Outdoor Dry Bulb	C
Outdoor Air Density	kg/m^3
Water Mains Temperature	C
Number of Occupants	-
Lights Total Heat Gain	J
Electric Equipment Total Heat Gain	J
Zone Mean Air Temperature	C
Zone Infiltration Volume	m^3

Since people usually do not work in weekends or holidays, the energy requirement in these days is quite small compared to normal working days. This means weekends and holidays have totally different energy behaviors from working days. Therefore, to simplify the model in our practice, we only use the consumption data of working days in the experiments. One more building is simulated for model evaluation purpose. The attributes of the three datasets are shown in Table II.

B. Experiments and Results

We perform the experiments on two shared memory computers which represent two different hardware architectures. As outlined in Table III, the first computer has 2 cores while the second one has 4. Both of them have a shared L2 cache and memory and running 64 bit Linux system (kernel version 2.6.27-7).

TABLE III: The physical features of the experimental environment.

Features	Computer-I	Computer-II
# of CPUs	1	1
# of cores	2	4
Frequency	3.4GHz*2	1.6GHz*4
Memory	2G	2G
L2 cache	2M	4M

We train all SVRs with Gaussian kernel, i.e., $K(x_i, x_j) = exp\{-\gamma\|x_i - x_j\|^2\}$. The parameters C and γ are evaluated by 5-fold cross validation and ϵ is set to 0.01, as shown in the last three columns of Table II. Since the caching technique is crucial for performance, for a reliable comparison, we set the cache size to be the same for all three SVR implementations. Furthermore, we restrict the cache size to be far smaller than the memory size in order to minimize page faults in runtime. Here we have to emphasize that the following reported performance might not be the best for all three implementations, only serving for comparison purpose.

TABLE II: Description of the three datasets and the three parameters of SVR on each dataset.

Dataset	# training samples	# testing samples	# Dimensions	C	γ	ϵ
One building	5064	1008	8	32	0.594	0.01
20 buildings	49940	2498	12	16	0.4193	0.01
50 buildings	124850	2498	12	16	0.4357	0.01

Two performance evaluation methods are applied in this work. One is the mean squared error (MSE) which gives the average deviation of the predicted values to the real ones. Another is the squared correlation coefficient (SCC) which lies in $[0, 1]$ and gives the ratio of successfully predicted number of target values on total number of target values.

On each dataset, we train the sequential implementation Libsvm, and the parallel implementations Pisvm and MRPsvm on both computer-I and computer-II. Table IV shows the results of the three implementations performed on dual-core processor, including the number of support vectors (nSVs), MSE, SCC and training time. We show the training time on quad-core system in Table V. Since nSVs, MSE and SCC in quad-core case are the same as those in dual-core case, we omit them in Table V. Note that the time columns represent the whole training time, i.e., from reading the problem to writing the outputs.

The results demonstrate that MRPsvm has successfully parallelized SVR training. For all three datasets, the accuracy and the nSVs of MRPsvm are quite close to that of Libsvm and Pisvm. Much more time is saved when running MRPsvm than Libsvm and Pisvm in all tests. Here we use speedup to demonstrate how many times faster is the parallel implementation than sequential implementation:

$$Speedup = \frac{Training\ time\ of\ Libsvm}{Training\ time\ of\ parallel\ implementation}$$

We show the speedup of MRPsvm and that of Pisvm comparatively in Figures 3, 4, 5. For the case of one building, the speed of MRPsvm is twice higher than that of Libsvm. For the case of 20 and 50 buildings, MRPsvm performs more than 16 times faster than Libsvm. On both computers and on all three datasets, MRPsvm achieves remarkable higher speedup than Pisvm, indicating that MRPsvm is more suitable than Pisvm on multi-core systems. The speed improvement by MRPsvm is particularly obvious in multiple buildings cases, indicating that MRPsvm performs better than Pisvm on large scale datasets. However, the better performance is not guaranteed on even larger problems due to locality and overheads of reduction. In each mapper, the updating of f requires the access of the whole data samples and several temporal vectors which have the size close to $2l$. Therefore, for large datasets, it is difficult to guarantee the locality for using L2 cache which is shared among several threads. Since we partition the global problem by columns, each mapper generates $2l$ intermediate f_i, the reduction is costly when l goes to a large scale.

It is not surprising that the parallel performance on quad-core only slightly outperforms that on dual-core. One reason is that the two computers have different architectures, e.g., the single processor frequency of computer-II is half slower than that of computer-I. The other reason is that we set the cache size to be the same for these two architectures and did not make full use of memory. This means far more time could be saved if we increase the cache size with caution to the maximum. In this optimum case, the cache size of MRPsvm and Libsvm is larger than that of Pisvm, since the former two systems generally require less storage space. It is implied that more improvements can be achieved for MRPsvm than Pisvm.

We note that the accuracy of MRPsvm is almost the

TABLE V: The training time of the three predictors performed on computer-II. Time unit is second.

Dataset	Libsvm	Pisvm	MRPsvm
1 building	18.0	7.3	6.9
20 buildings	2532.1	214.1	133.5
50 buildings	32952.2	2325.5	1699.0

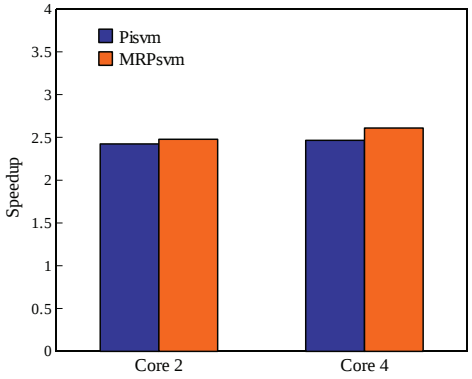

Fig. 3: Speedup of Pisvm and MRPsvm on one building's data.

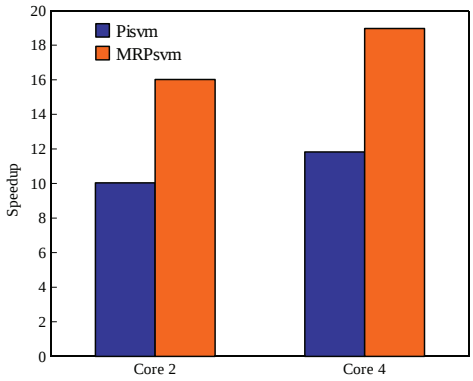

Fig. 4: Speedup of Pisvm and MRPsvm on 20 buildings' data.

TABLE IV: The training time and performance of the three predictors on three datasets performed on computer-I. bd: building, nSVs: number of support vectors, MSE: mean squared error, SCC: squared correlation coefficient. The unit of time is second.

Data	Libsvm				Pisvm				MRPsvm			
	nSVs	***MSE***	***SCC***	***Time***	***nSVs***	***MSE***	***SCC***	***Time***	***nSVs***	***MSE***	***SCC***	***Time***
1 bd	2150	6.16e-4	0.97	22.3	2162	6.10e-4	0.97	9.2	2168	6.14e-4	0.97	9.0
20 bd	9014	2.11e-3	0.96	3407.0	8967	2.12e-3	0.96	339.5	8970	2.08e-3	0.96	212.7
50 bd	22826	3.73e-4	0.97	44212.5	22823	3.74e-4	0.97	4179.8	22799	3.73e-4	0.97	2745.8

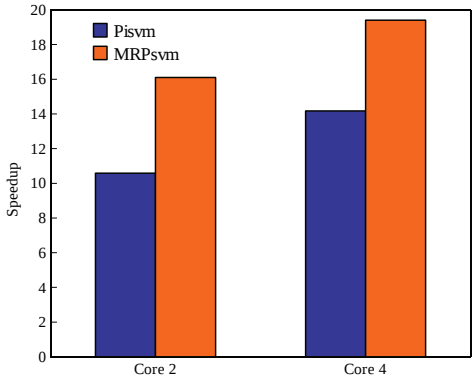

Fig. 5: Speedup of Pisvm and MRPsvm on 50 buildings' data.

same as that of Libsvm. In fact, these three implementations have essentially the same mechanism in QP solving, i.e., to iteratively optimize one pair of variables until achieving global optimization. Their difference comes from the selection of variable pair which may induce totally different results in a sub-task. But in the global point of view, as long as convergence is reached, the influence of internal difference is marginal.

VI. Conclusion

We focus in parallel SVR for solving large scale problems since it is suitable to the building energy analysis. The proposed MRPsvm implements decomposition method with SMO as the sub-problem solver. The simple, but pragmatic programming framework Map-Reduce is adopted to conduct parallel updating of the gradient vector f. The system is relatively easy to be implemented and very high performance is achieved.

Experimental results show that the new system provides the same accuracy as Libsvm does, yet performs far more efficiently than the sequential implementation in solving stated problems. On the smallest dataset, MRPsvm achieves a more-than-twice speed times up than Libsvm while on the largest dataset, the speed times up can reach higher than 16-fold and 19-fold on dual-core system and on quad-core system, respectively. The proposed implementation is superior to the state-of-the-art Pisvm in all tests in the sense of both speed and memory requirement. Since the multi-core system is dominating the trend of processor development and is highly available in modern market, MRPsvm is potentially very practical and feasible in solving large scale regression problems. Furthermore, the success of MRPsvm indicates that Map-Reduce is a possible option to parallelize machine learning algorithms.

However, the proposed system is not yet mature, there are still several aspects worth considering for further improvements, for instance, shrinking, finding the best granularity of parallel work for a particular dataset.

References

[1] V. N. Vapnik, *The nature of statistical learning theory*. Springer-Verlag New York, Inc., 1995.

[2] B. Dong, C. Cao, and S. E. Lee, "Applying support vector machines to predict building energy consumption in tropical region," *Energy and Buildings*, vol. 37, no. 5, pp. 545–553, 2005.

[3] Q. Li, Q. Meng, J. Cai, Y. Hiroshi, and M. Akashi, "Applying support vector machine to predict hourly cooling load in the building," *Applied Energy*, vol. 86, no. 10, pp. 2249–2256, 2009.

[4] F. Lai, F. Magoulès, and F. Lherminier, "Vapnik's learning theory applied to energy consumption forecasts in residential buildings," *International Journal of Computer Mathematics*, vol. 85, no. 10, pp. 1563–1588, 2008.

[5] H. X. Zhao and F. Magoulès, "Parallel support vector machines applied to the prediction of multiple buildings energy consumption," *Journal of Algorithms & Computational Technology*, vol. 4, no. 2, pp. 231–249, 2010.

[6] E. Osuna, R. Freund, and F. Girosi, "Training support vector machines: an application to face detection," in *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition*, 1997, pp. 130–136.

[7] T. Joachims, "Making large-scale support vector machine learning practical," *Advances in kernel methods: support vector learning*, pp. 169–184, 1999.

[8] E. Y. Chang, K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, and H. Cui, "Psvm: Parallelizing support vector machines on distributed computers," in *NIPS*, vol. 20, 2007.

[9] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization," *Advances in kernel methods: support vector learning*, pp. 185–208, 1999.

[10] S. Shalev-Shwartz, Y. Singer, and N. Srebro, "Pegasos: Primal estimated sub-gradient solver for svm," in *Proceedings of the 24th International Conference on Machine Learning*, 2007, pp. 807–814.

[11] C. C. Chang and C. J. Lin, *LIBSVM: a library for support vector machines*, 2001, available online at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[12] D. Brugger, "Parallel support vector machines," in *Proceedings of the IFIP International Conference on Very Large Scale Integration of System on Chip*, 2007.

[13] R. E. Fan, P. H. Chen, and C. J. Lin, "Working set selection using second order information for training support vector machines," *Journal of Machine Learning Research*, vol. 6, pp. 1889–1918, 2005.

[14] L. Zanni, T. Serafini, and G. Zanghirati, "Parallel software for training large scale support vector machines on multiprocessor systems," *Journal of Machine Learning Research*, vol. 7, pp. 1467–1492, 2006.

[15] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[16] C. T. Chu, S. K. Kim, Y. A. Lin, Y. Yu, G. R. Bradski, A. Y. Ng, and K. Olukotun, "Map-reduce for machine learning on multicore," in *NIPS*, 2006, pp. 281–288.

[17] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, "Evaluating mapreduce for multi-core and multiprocessor systems," in *Proceedings of the IEEE 13th International Symposium on High Performance Computer Architecture*, 2007, pp. 13–24.

[18] Z. A. Zhu, W. Z. Chen, G. Wang, C. G. Zhu, and Z. Chen, "P-packsvm: Parallel primal gradient descent kernel svm," in *Proceedings of the 9th IEEE International Conference on Data Mining*, 2009, pp. 677–686.

[19] D. Bickson, E. Yom-tov, and D. Dolev, "A gaussian belief propagation solver for large scale support vector machines," in *Proceedings of the 5th European Conference on Complex Systems*, 2008.

[20] H. P. Graf, E. Cosatto, L. Bottou, I. Durdanovic, and V. Vapnik, "Parallel support vector machines: The cascade svm," in *In Advances in Neural Information Processing Systems*, vol. 17, 2005, pp. 521–528.

[21] J. X. Dong, A. Krzyzak, and C. Y. Suen, "Fast svm training algorithm with decomposition on very large data sets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 4, pp. 603–618, 2005.

[22] T. Hazan, A. Man, and A. Shashua, "A parallel decomposition solver for svm: Distributed dual ascend using fenchel duality," in *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2008, pp. 1–8.

[23] Y. Lu and V. Roychowdhury, "Parallel randomized sampling for support vector machine (svm) and support vector regression (svr)," *Knowledge and Information Systems*, vol. 14, pp. 233–247, 2008.

[24] L. J. Cao, S. S. Keerthi, C. J. Ong, J. Q. Zhang, U. Periyathamby, J. F. Xiu, and H. P. Lee, "Parallel sequential minimal optimization for the training of support vector machines," *IEEE Transactions on Nueral Networks*, vol. 17, no. 4, pp. 1039–1049, 2006.

[25] B. Catanzaro, N. Sundaram, and K. Keutzer, "Fast support vector machine training and classification on graphics processors," in *Proceedings of the 25th International Conference on Machine Learning*, 2008, pp. 104–111.

[26] EnergyPlus, 2010, available online at: http://www.EnergyPlus.gov.

[27] H. X. Zhao and F. Magoulès, "Feature selection for predicting building energy consumption based on statistical learning method," *Journal of Algorithms & Computational Technology*, 2011 (in press).