PaperReading

MIC-SVM:Designing a highly efficient support vector machine for advanced modern multi-core and many-core architectures
Y. You et al., IPDPS 2014

Bo Peng

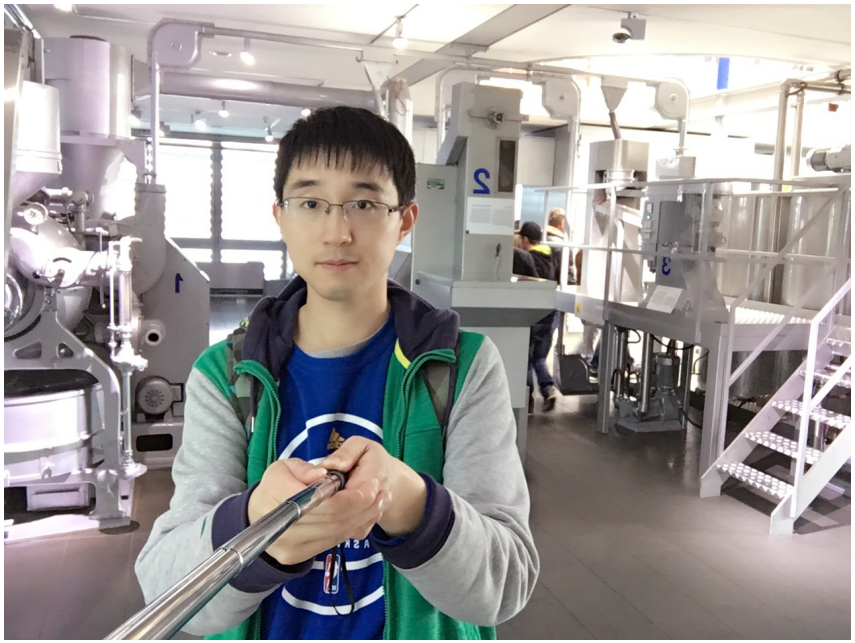
Digital Science Center
Indiana University

pengb@indiana.edu

April 13, 2017

Outline

the Authors



- Yang You, PhD student at UC Berkeley CS, Master 2015 Tsinghua
- research interests include Parallel Computing, Distributed Systems, and Machine Learning
- CA-SVM: Communication-Avoiding Support Vector Machines on Distributed Systems. Best Paper IPDPS 2015

the Authors

- Prof. James Demmel
  Computer Science Division Chair, and EECS Associate Chair,Professor of Mathematics and Computer Science
- Research Projects:
  Communication-Avoiding Algorithms
  The Complexity of Accurate Floating Point Computation
  LAPACK, scaLAPACK, main contributor

the Authors: BeBOP Group

BeBOP(Berkeley Benchmarking and OPtimization Group)
The BeBOP group is broadly interested in understanding software performance tuning issues, and the interaction or implications for hardware design. Among our general interests are

- the interaction between application software, compilers, and hardware managing trade-offs among the various measures of performance, such as speed, accuracy, power, storage, ...
- automating the performance tuning process, starting with the computational kernels which dominate application performance in scientific computing and information retrieval
- performance modeling and evaluation of future computer architectures

Outline

Motivations

Mic-svm: Designing a highly efficient support vector machine for advanced modern multi-core and many-core architectures

Motivation

- svm is important...widely used...recently being used in HPC for performance and power... prediction
- challenge of scaling performance over tens or even hundreds of cores within a single chip
- there has been no efficient SVM tool designed for advanced x86 based multi- and many-core architectures

Research Problem

Problems to solve(...unwritten...)

- What's the problem? –svm train phase inefficient in x-core system
- What's the state-of-art solutions? – libsvm, gpusvm
- What's the deficiency of those solutions? –
- What's the methodology to investigate these performance issues? – bottleneck, profiling, bound analysis, ...
- What's the suitable optimization techniques?
- What's a good mapping from input data pattern to specific architecture and optimization decisions?

Contributions

Contributions

- Designing and implementing MIC-SVM a highly efficient parallel...
- Proposing novel analysis methods and optimization techniques (i.e. adaptive support for input patterns and data parallelism) to fully utilize the multi-level parallelism provided by the studied architectures.
- Exploring and improving the deficiencies of the current tool...
- Providing insights on how to select the most suitable architectures for specific algorithms and input data patterns to achieve best performance.

Outline

Classification Problems

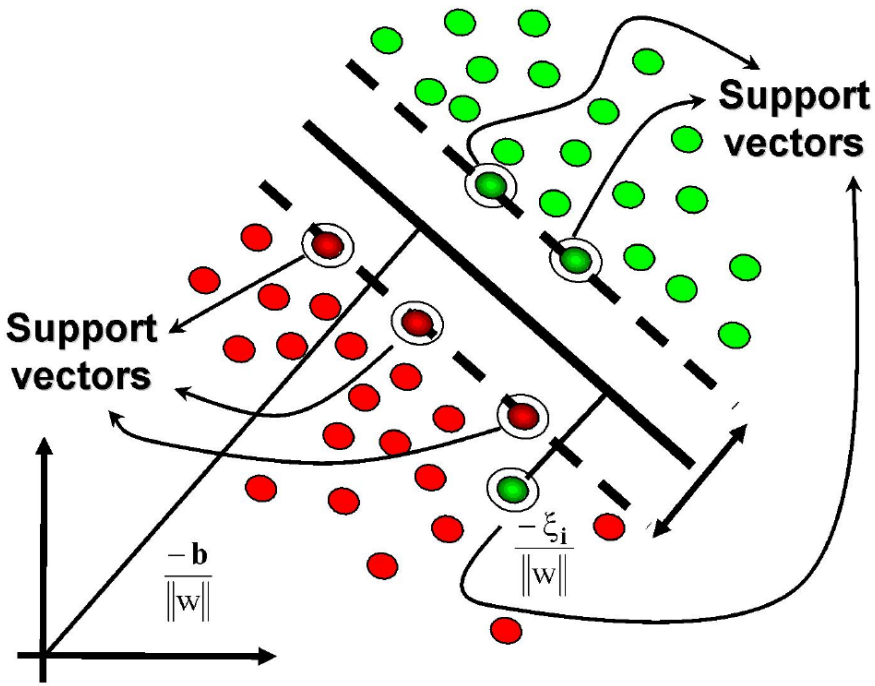
IRIS Dataset

s.no	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

s.no	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
145	6.7	3.3	5.7	2.5	virginica
146	6.7	3.0	5.2	2.3	virginica
147	6.3	2.5	5.0	1.9	virginica
148	6.5	3.0	5.2	2.0	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3.0	5.1	1.8	virginica

- features: {sepal.length, sepal.width, petal.length, petal.width}.
- lables: {setosa, virginica}
- task: predict label by observed data with the fetures

SVM(Support Vector Machine)

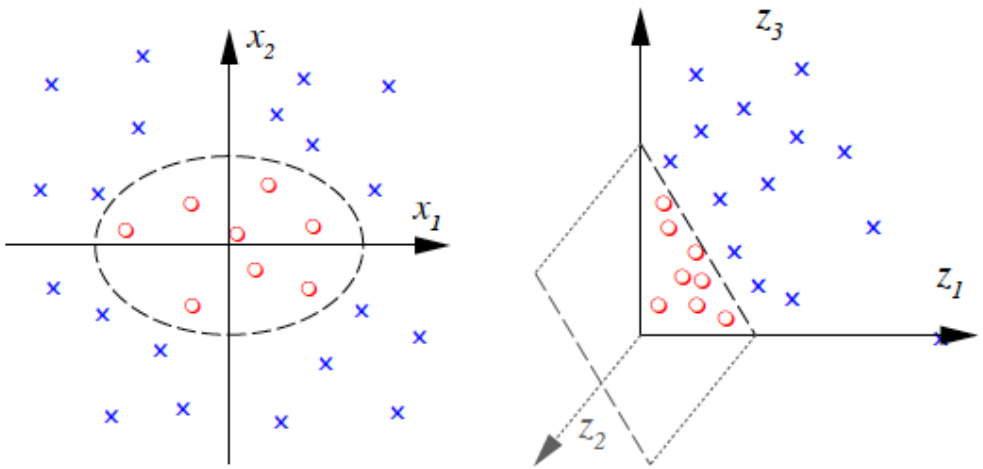


- separation plane: $y = \vec{w} \cdot \vec{x} - b$
- separation with maximum margin
- finally, we will get: $\vec{w} = \sum_{i=1}^{n} \alpha_i y_i \vec{x_i}$ $\vec{w}$ can be written as a linear combination of the support vectors.

- a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.
- $\arg\min_{\vec{w},b} \left\{ \frac{1}{2} \|\vec{w}\|^2 \right\}$ subject to $y_i(\vec{w} \cdot \vec{x_i} - b) \geq 1, \forall i$
- $\arg\min_{\vec{w},b,\xi} \left\{ \frac{1}{2} \|\vec{w}\|^2 + C \sum_{i=1}^{n} \xi_i \right\}$ subject to $y_i(\vec{w} \cdot \vec{x_i} - b) \geq 1 - \xi_i, \forall i$

SVM: Kernel Trick

$$\Phi : R^2 \to R^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{(2)}x_1x_2, x_2^2)$$



$$\begin{aligned} & k(\mathbf{x}, \mathbf{x}') \\ &= \langle \mathbf{x}, \mathbf{x}' \rangle^2 \\ &= (x_1 x'_1 + x_2 x'_2)^2 \\ &= x_1^2 x'^2_1 + \sqrt{2} x_1 x_2 \sqrt{2} x'_1 x'_2 \\ &+ x_2^2 x'^2_2 \\ &= \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle \end{aligned}$$

- $\hat{y} = \vec{w} \cdot \vec{x} - b = \sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}') - b$, for $k(\mathbf{x}, \mathbf{x}') = \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle_{\mathcal{V}}$
- use of kernel functions, which enable them to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between the images of all pairs of data in the feature space.

Numerical Optimization Problem

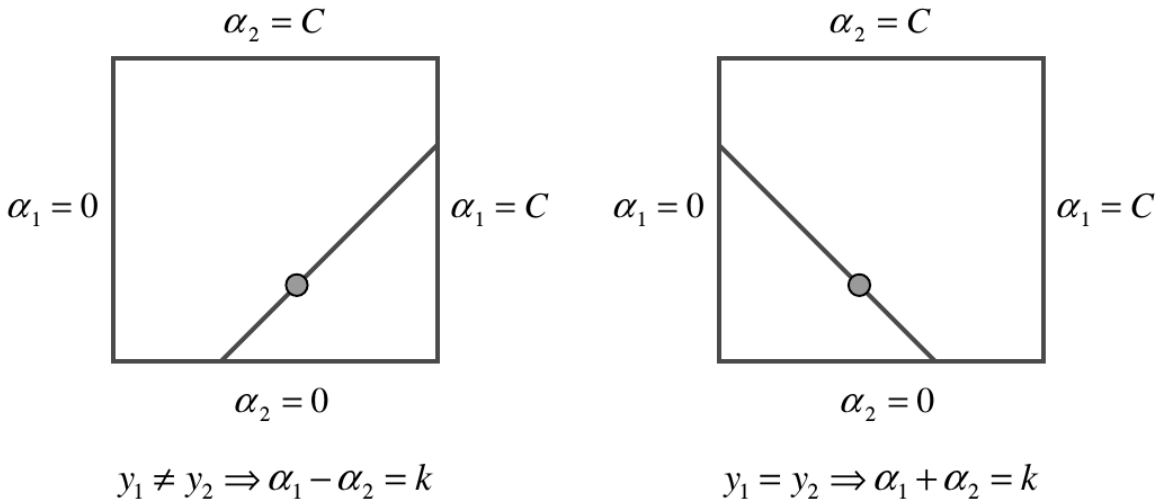
.....transformed into the dual form

- QP(Quadratic programming) problem
- training phase
  Maximize: $F(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j Kernel(X_i, X_j)$
  subject to: $\sum_{i=1}^{n} \alpha_i y_i = 0$ and $0 \leq \alpha_i \leq C, \forall i$
- prediction phase
  predict: $\hat{y}_i = u_i = \sum_{j=1}^{n} \alpha_j y_j K(\vec{x_j}, \vec{x_i}) - b$

SMO(Sequential Minimal Optimization) Algorithm

- At every step, SMO chooses two Lagrange multipliers to jointly optimize, finds the optimal values for these multipliers, and updates the SVM to reflect the new optimal values

subject to:
$\sum_{i=1}^{n} \alpha_i y_i = 0$
and
$0 \leq \alpha_i \leq C, \forall i$

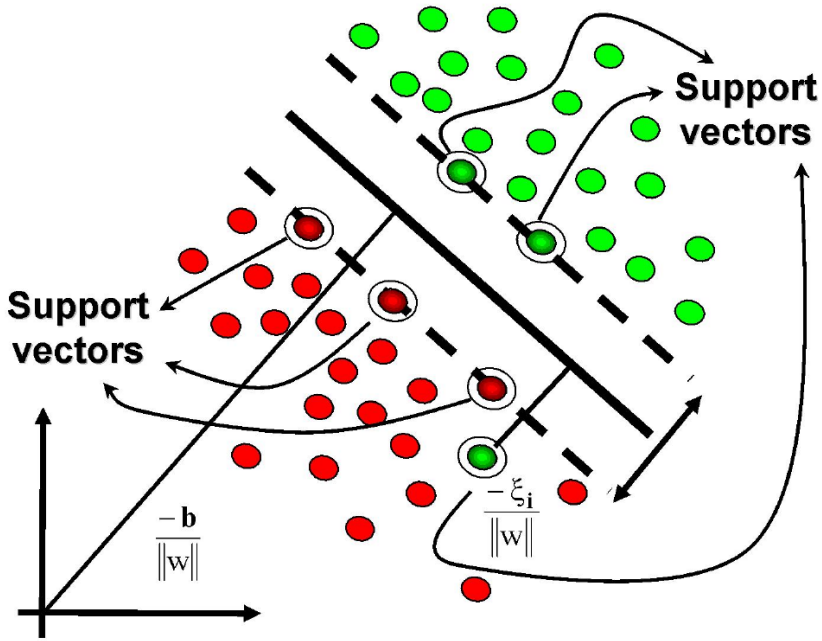


- The inequality constraints cause the Lagrange multipliers to lie in the box. The linear equality constraint causes them to lie on a diagonal line. Therefore, one step of SMO must find an optimum of the objective function on a diagonal line segment.

SMO: Update Equation

- The second derivative of the objective function along the diagonal line can be expressed as: $\eta = K(\vec{x_1}, \vec{x_1}) + K(\vec{x_2}, \vec{x_2}) - 2K(\vec{x_1}, \vec{x_2})$
- $\alpha_2^{new} = \alpha_2 + \frac{y_2(E_1 - E_2)}{\eta}$ where $E_i = u_i - y_i$ is the error on ith training example
- $\alpha_1^{new} = \alpha_1 + y_1 y_2(\alpha_2 - \alpha_2^{new})$
- $E_i^{new} = E_i + (\alpha_1^{new} - \alpha_1) y_1 K(\vec{X_1}, \vec{X_i}) + (\alpha_2^{new} - \alpha_2) y_2 K(\vec{X_2}, \vec{X_i})$
- more deatils refer to [Platt 1998]

SMO:selection of the two lagrange multipliers



The Karush-Kuhn-Tucker (KKT) conditions are necessary and sufficient conditions for an optimal point of a positive definite QP problem.

set: $u_i = \sum_{j=1}^{n} \alpha_j y_j K(\vec{x_j}, \vec{x_i}) - b$,

then

$\alpha_i = 0 \Leftrightarrow y_i u_i \geq 1$,

$0 < \alpha_i < C \Leftrightarrow y_i u_i = 1$,

$\alpha_i = C \Leftrightarrow y_i u_i \leq 1$,

- chooses the first one by iterating over the entire training set, determining whether each example violates the KKT conditions.
- chooses the second one with maximized step size which is approximated by the absolute value of error $|E_1 - E_2|$.

SMO:selection equations

- (KKT) conditions
  set: $u_i = \sum_{j=1}^{n} \alpha_j y_j K(\vec{x_j}, \vec{x_i}) - b$,
  $\alpha_i = 0 \Leftrightarrow y_i u_i \geq 1$,
  $0 < \alpha_i < C \Leftrightarrow y_i u_i = 1$,
  $\alpha_i = C \Leftrightarrow y_i u_i \leq 1$,
- group the examples into two sets:
  $I_1 = \{i : 0 < \alpha_i < C\} \cup \{i : y_i > 0, \alpha_i = 0\} \cup \{i : y_i < 0, \alpha_i = C\}$
  where $E_i = u_i - y_i \geq 0$ otherwise voilate KKT cond.
  $I_2 = \{i : 0 < \alpha_i < C\} \cup \{i : y_i < 0, \alpha_i = 0\} \cup \{i : y_i > 0, \alpha_i = C\}$
  where $E_i = u_i - y_i \leq 0$ otherwise voilate KKT cond.
- selection equations:
  $i_1 = \arg\min_i \{E_i : i \in I_1\}$
  $i_2 = \arg\max_i \{E_i : i \in I_2\}$

SMO:Algorithm PseudoCode

Algorithm 1 The Original SMO Algorithm

0: Start
1: Input the training samples X_i and pattern labels y_i, $\forall i \in \{1, 2, ..., n\}$.
2: Initializations, $\alpha_i = 0$, $f_i = -y_i$, $\forall i \in \{1, 2, ..., n\}$.
3: Initializations, $b_{high} = -1$, $i_{high} = \min\{i : y_i = -1\}$, $b_{low} = 1$, $i_{low} = \max\{i : y_i = 1\}$.
4: Update α_{high} and α_{low} according to Equation (4) and (5).
5: If $Kernel(X_{high}, X_i)$ is not in memory, then compute $Kernel(X_{high}, X_i)$ and cache $Kernel(X_{high}, X_i)$ in memory using the LRU strategy.
6: If $Kernel(X_{low}, X_i)$ is not in memory, then compute $Kernel(X_{low}, X_i)$ and cache $Kernel(X_{low}, X_i)$ in memory through LRU strategy.
7: Update f_i according to Equation (6), $\forall i \in \{1, 2, ..., n\}$
8: Compute i_{high}, i_{low}, b_{high}, and b_{low} according to Equation (9) and (10).
9: Update α_{high} and α_{low} according to Equation (4) and (5).
10: If iterations meet certain number, then do shrinking.
11: If $b_{low} > b_{high} + 2 \times tolerance$, then go to Step 5.
12: End

SMO:Summary

Input Data		Model	Cached
X1	y1	α1	E1
X2	y2	α2	E2
X3	y3	α3	E3
X4	y4	α4	E4
...	...	...	...
Xi	yi	αi	Ei
...	...	...	...
Xn	yn	αn	En

- QP overall time complexity at least $O(n^2)$, space $O(n^2)$
- SMO type algorithm, empirically, it is known that the number of iterations may be higher than linear to the number of training data.
- $E_i^{new} = E_i + (\alpha_1^{new} - \alpha_1) y_1 K(\vec{X_1}, \vec{X_i}) + (\alpha_2^{new} - \alpha_2) y_2 K(\vec{X_2}, \vec{X_i})$
- The update of all E_i at each step requires access to all the training samples, which costs more than 90% of the total time in the entire SMO algorithm.

Outline

Methodology to parallelizing and optimizing

- In this section, we will briefly describe several important design methods and optimization details for our proposed MIC-SVM on parallelizing and accelerating SVM method on Intel Ivy Bridge CPUs and Intel Xeon Phi coprocessor (MIC).

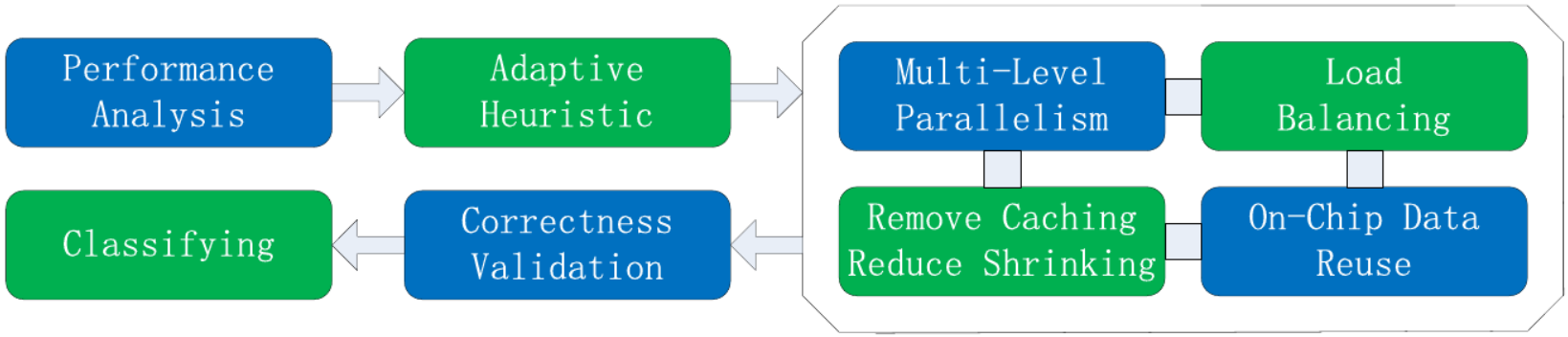


Fig. 2. General flow for parallelizing and optimizing MIC-SVM.

A: Analysis of the Algorithm and Architecture

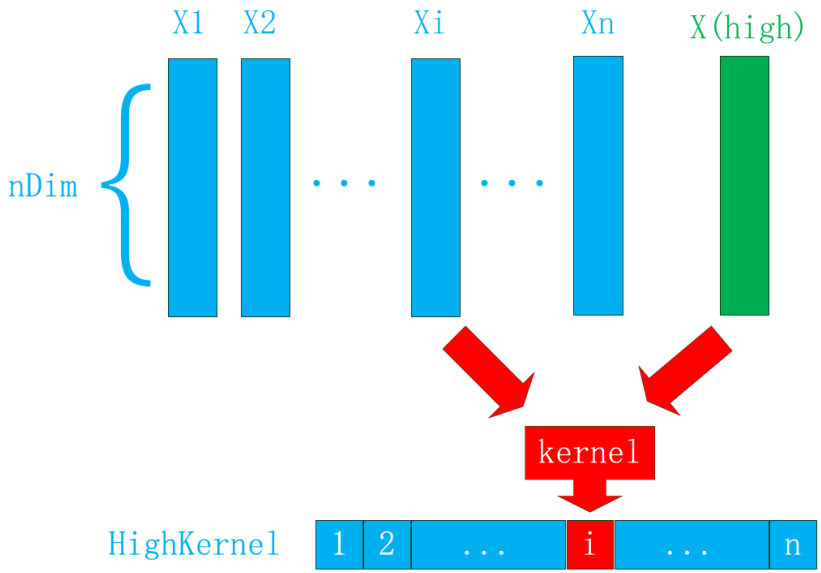


Fig. 3. This figure corresponds to the computation of $HighKernel$ (used in Equation (11)). $HighKernel$ is a vector, and $HighKernel(i) = Kernel(X_{high}, X_i)$. From this figure we can see that the evaluation of $HighKernel$ need to access all the training samples. The computation of $LowKernel$ can be accomplished in the same way.

- $E_i^{new} = E_i + (\alpha_1^{new} - \alpha_1) y_1 K(\vec{X_1}, \vec{X_i}) + (\alpha_2^{new} - \alpha_2) y_2 K(\vec{X_2}, \vec{X_i})$
- The update of all E_i is the bottleneck.
- what's the bound?

A: Analysis of the Algorithm and Architecture

TABLE II
RCMB VALUES FOR EVALUATED ARCHITECTURES IN THIS PAPER

Architecture	Ivy Bridge	MIC	Fermi	Kepler
Frequency (GHz)	2.20	1.09	1.50	0.73
Peak Performance Double (Gflops)	422	1010	515	1320
Peak Performance Single (Gflops)	844	2020	1030	3950
Theoretical Memory bandwidth (GB/s)	128	352	192	250
RCMB Single (Gflops/GB)	6.59	5.74	5.36	15.8
RCMB Double (Gflops/GB)	3.30	2.87	2.68	5.28

TABLE I
STANDARD KERNEL FUNCTIONS

Linear	$Kernel(X_i, X_j) = X_i \dot{X}_j$
Polynomial	$Kernel(X_i, X_j) = (aX_i \dot{X}_j + r)^d$
Gaussian	$Kernel(X_i, X_j) = -\gamma \|\|X_i - X_j\|\|^2$
Sigmoid	$Kernel(X_i, X_j) = tanh(aX_i \dot{X}_j + r)$

- Define the Ratio of Computation to Memory Access (RCMA) to describe SMOs algorithmic feature.
  $RCMA = \frac{number\_of\_comp\_flops}{number\_of\_memory\_access\_bytes}$
- Define Ratio of Peak Computation to Peak Memory Bandwidth (RCMB) to describe the theoretical architectural bound.
  $RCMB = \frac{theoretical\_peak\_performance}{theoretical\_bandwidth}$

C: Two-Level Parallelism

arch	**task parallelism**	**data parallelism**
Ivy Bridge	utilizing multiple hardware threads	SIMD(AVX256)
MIC	utilizing multiple hardware threads	on-core VPU (Vector Processing Unit) and SIMD(AVX512)
Fermi,Kepler	independent warps	different CUDA cores

C: Two-Level Parallelism

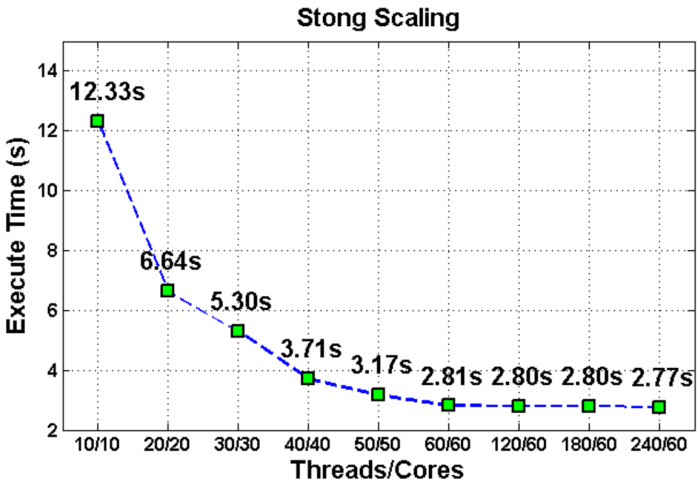


Fig. 6. Results for strong scaling test of MIC-SVM on Intel Xeon Phi using different number of threads and cores. threads/core=number of threads/number of cores.

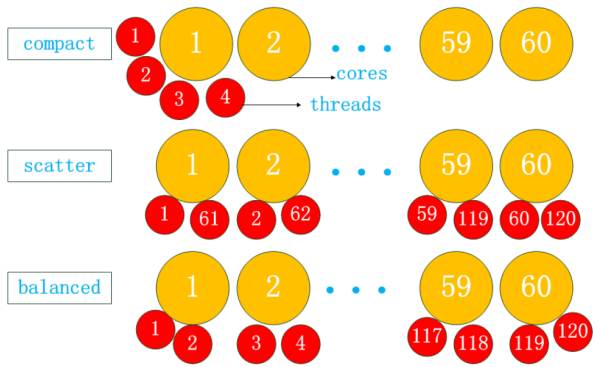


Fig. 7. Three affinity models for load balancing. 1) compact mode: the new

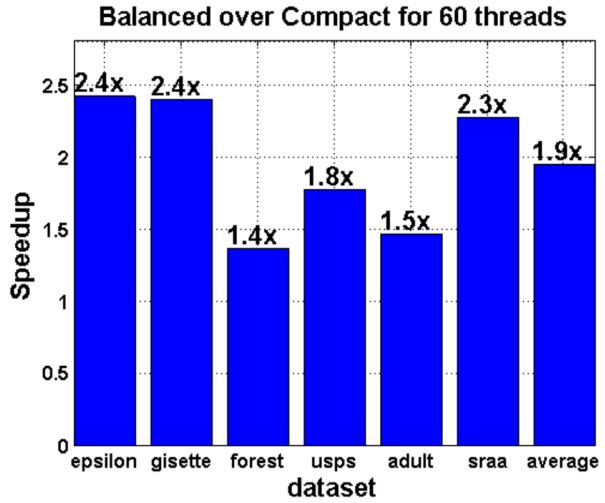


Fig. 8. Performance speedup of using "balanced mode" over "compact mode" when running MIC-SVM on Intel Xeon Phi with various types of datasets.

- The proper number of threads: number of physical resources in device
- Load balancing: affinity models
- use the Cilk array notation to achieve the data parallelism explicitly rather than applying the implicit compiler model.

D: Removing Caching and Reduce Shrinking Frequency

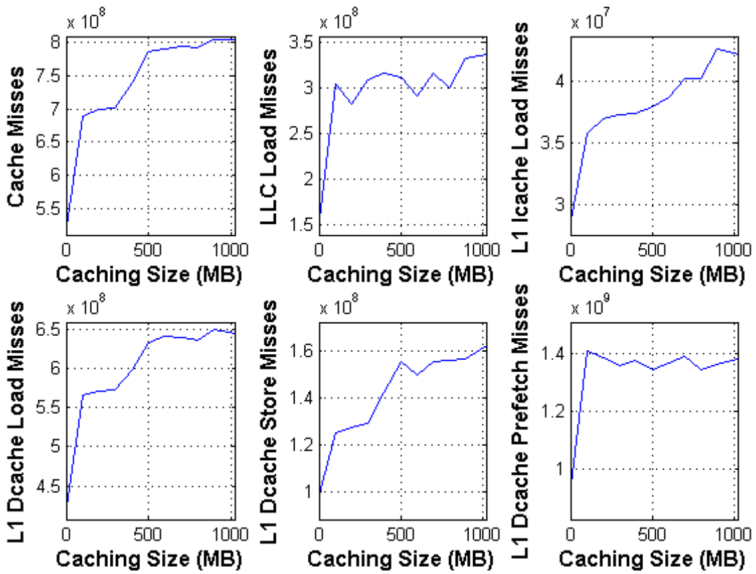


Fig. 9. Memory activities of running MIC-SVM on Intel Xeon Phi architecture while increasing the caching size .

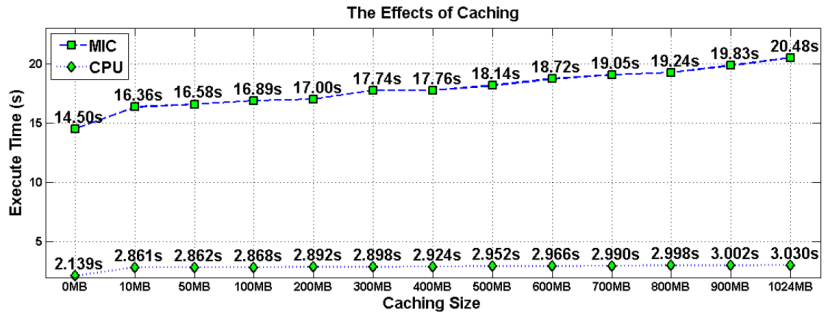


Fig. 10. Caching effects on overall execution time when running MIC-SVM on Intel Xeon Phi (MIC) and Ivy Bridge CPUs under increasing caching size.

- caching strategy can effectively reduce the number of kernels being evaluated. However, it also requires more memory access, may affect the performance negatively due to high memory access latency, memory contention from multithreading, and limited bandwidth.

E. Reducing the Gap Between RCMA and RCMB

- Together with load balancing, we employ the OpenMP SCHEDULE technique (static type, default chunk size) to distribute the training samples evenly to all hardware threads.
- exploring the proper granularity of parallelism by configuring the data sizes for two-level parallelism,
- minimizing the threads creation and destroy to reduce synchronization overhead
- maximizing the TLB page size to 2MB to obtain significantly more memory coverage when the datasets require more than 16MB memory.

Parallel SMO Algorithm

Algorithm 2 Parallel SMO Algorithm

0: Start

1: Input the training samples X_i and pattern labels y_i, $\forall i \in \{1, 2, ..., n\}$. Applying adaptive heuristic support to decide how to process input datasets (sparse or dense) and whether to vectorize the kernel or not.

2: Map all the X_i, y_i, and α_i ($\forall i \in \{1, 2, ..., n\}$) to all the hardware threads evenly through static SCHEDULE.

3: Initializations, for all hardware threads concurrently, $\overrightarrow{\alpha}_t = 0$, $\overrightarrow{F}_t = -\overrightarrow{Y}_t$, $\forall t \in \{1, 2, ..., T\}$.

4: Initializations, $b_{high} = -1$, $i_{high} = \min\{i : y_i = -1\}$, $b_{low} = 1$, $i_{low} = \max\{i : y_i = 1\}$.

5: Check the thread Affinity to keep load balancing.

6: Update α_{high} and α_{low} according to Equation (4) and (5).

7: If kernel is vectorized, then do each kernel evaluation in each hardware thread through vectorization. Update $\overrightarrow{F}_t$ according to Equation (6) through task parallelism, $\forall t \in \{1, 2, ..., T\}$

8: If kernel is un-vectorized, then do each kernel evaluation in each hardware serially. Update $\overrightarrow{F}_t$ according to Equation (6) through the combination of task parallelism and data parallelism, $\forall t \in \{1, 2, ..., T\}$

9: Local reduce to obtain i_{high}, i_{low}, b_{high}, and b_{low} according to Equation (9) and (10) in each hardware thread.

10: Global reduce to obtain i_{high}, i_{low}, b_{high}, and b_{low} according to Equation (9) and (10).

11: Update α_{high} and α_{low} according to Equation (4) and (5).

12: If $b_{low} > b_{high} + 2 \times tolerance$, then go to Step 5.

13: End

Outline

A. Experimental Setup and Datasets

TABLE III
RELATED PARAMETERS OF ARCHITECTURES

Architecture	Ivy Bridge	KNC	Fermi	Kepler
Cores	24	61	448	2496
L1 cache (KB)	64/core	64/core	64/SM	64/SM
L2 cache (KB)	256/core	512/core	768/card	1536/card
L3 cache (MB)	30/socket	0	0	0
Coherent cache	L3	L2	L2	L2
Memory type	DDR3	GDDR5	GDDR5	GDDR5
Memory size (GB)	64	8	6	6
Measured Memory bandwidth (GB/s)	68	159	97	188

TABLE IV
THE TEST DATASETS

Dataset	n	$nDim$	Density	Cost	Gamma
epsilon [25]	27,000	2,000	1.0000	1.0	0.08000
gisette [26]	6,000	5,000	0.9910	1.0	0.00020
forest [27]	12,000	54	0.2386	10000	0.00010
usps [28]	266,079	675	0.1497	1.0	0.03125
adult [18]	32,561	123	0.1128	0.1	0.06250
sraa [21]	72,309	20,958	0.0024	1.0	0.03125

C. Correctness Validation for MIC-SVM

TABLE V
THE CLASSIFICATION ACCURACY, THE VALUE OF B AND THE NUMBER OF SUPPORT VECTORS.

Datasets	LIBSVM Accuracy–b–SVs	MIC-SVM on KNC MIC Accuracy–b–SVs	MIC-SVM on Ivy Bridge Accuracy–b–SVs	GPUSVM on Kepler Accuracy–b–SVs	GPUSVM on Fermi Accuracy–b–SVs
epsilon	87.5%–0.0200–18116	87.5%–0.0200–18114	87.5%–0.0200–18113	87.5%–0.0200–18113	87.5%–0.0200–18113
gisette	97.7%––0.0034–1666	97.6%––0.0033–1665	97.6%––0.0033–1665	97.6%––0.0036–1665	97.6%––0.0036–1665
forest	82.2%–0.0120–11612	82.2%–0.0120–11609	82.2%–0.0120–11610	82.2%–0.0120–11610	82.2%–0.0120–11610
usps	99.2%––0.9464–39570	99.2%––0.9464–38596	99.2%––0.9464–38589	99.2%––0.9464–38581	99.2%––0.9464–38581
adult	84.4%–0.576–12281	84.4%–0.576–12273	84.4%–0.577–12273	84.4%–0.576–12278	84.4%–0.576–12278
sraa	97.0%––1.33–24430	97.0%––1.33–24392	97.0%––1.33–24395	misconvergence	misconvergence

- baseline: serial LIBSVM
- another solver on gpu: GPUSVM
- metrics: accuracy, support vectors number, b(discrepancy E)
- result: nearly the same

D. Performance Comparisons and Analysis

TABLE VI
ITERATIONS, TRAINING TIME AND SPEEDUPS.

Datasets	LIBSVM Iterations–Time(s)–Speedup	MIC-SVM on MIC Iterations–Time(s)–Speedup	MIC-SVM on Ivy Bridge Iterations–Time(s)–Speedup	GPUSVM on Kepler Iterations–Time(s)–Speedup	GPUSVM on Fermi Iterations–Time(s)–Speedup
epsilon	11040–2959–1×	**11686–35.1–84**×	11616–64.2–46×	11537–38.6–76×	11537–46.1–64×
gisette	1938–117.8–1×	**1887–2.75–43**×	1872–5.68–21×	1894–5.46–22×	1894–7.65–15×
forest	23520–77.2–1×	28912–17.6–4.4×	28983–2.14–36×	**29018–2.04–38**×	28946–5.10–15×
usps	34992–21945–1×	47201–806–27×	47173–884–25×	**47195–239–92**×	47195–429–51×
adult	8028–84.1–1×	8092–14.5–5.8×	**8150–2.14–39**×	8097–2.41–35×	8097–5.67–15×
sraa	14802–1128–1×	13687–81.3–14×	**13676–24.9–45**×	misconvergence	misconvergence

- With powerful SIMD mechanism (512 bits), MIC is a good candidate for the dense high-dimension datasets with modest number of training samples because data parallelism can be achieved efficiently through vectorization;
- Equipped with sufficient caches and high clock frequency, Ivy Bridge CPUs are suitable for sparse high-dimension datasets since these datasets often require coarse-grained parallel processing;
- GPUs are proper for the datasets with large number of training samples and low dimension because they are more likely to benefit from millions of fine-grained lightweight threads.

D. Performance Comparisons and Analysis

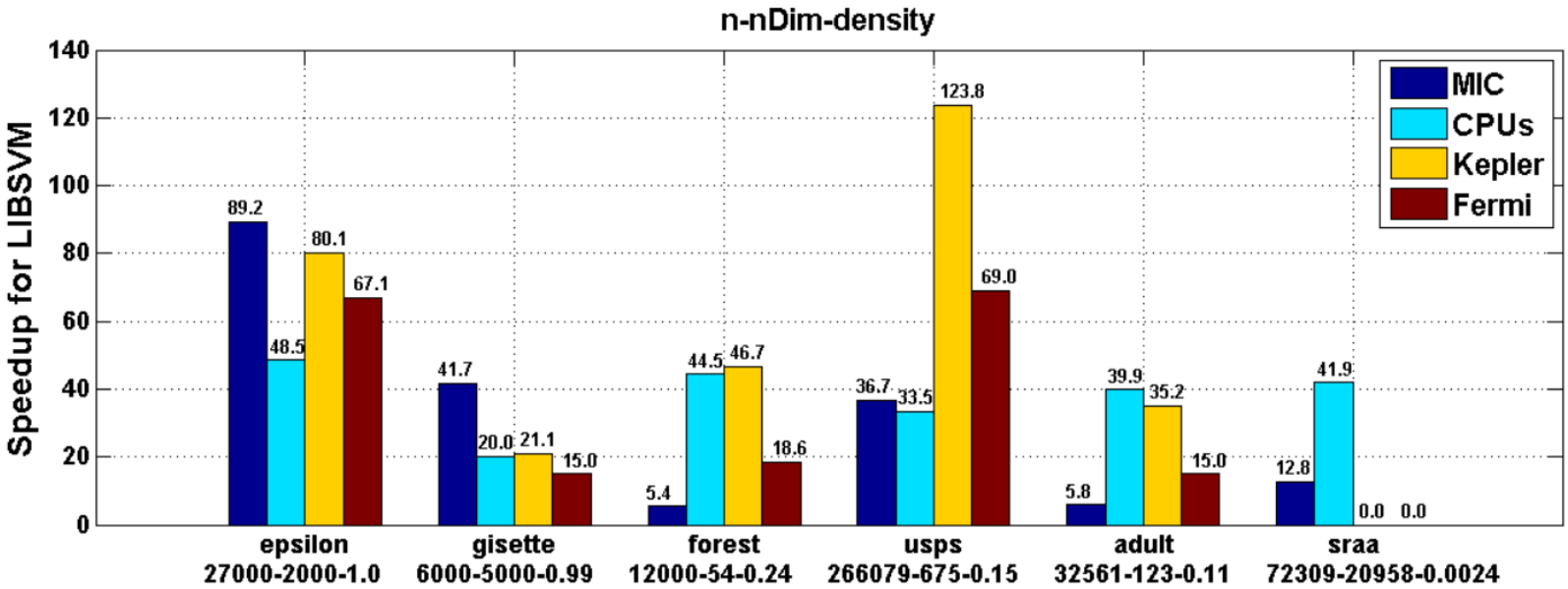


Fig. 11. This figure shows the speedups over LIBSVM based on the time of each iteration (rather than the whole time), n is the number of training samples, $nDim$ is the dimension of each training sample, and density represents the sparseness of a given dataset. All the results data shown in this figure are the average of many runs.

Outline

Conclusion

Conclusion

- propose MIC-SVM, a highly efficient parallel support vector machine for x86 based multi-core and many-core architectures such as Intel Ivy Bridge CPUs and Intel KNC MIC.
- propose various novel analysis and optimization strategies that are general and can be easily applied to accelerate other machine learning methods.
- explore and improve the deficiencies of the current SVM tools.
- provide insights on how to map the most suitable architectures to specific data patterns in order to achieve the best performance.

Future work

- In future, we plan to extend the current MIC-SVM to distributed memory environment using multiple MICs.

Outline

Comments on the paper

Pros

- gives a clear structure of the process of investigation.
- experiments are concise and informative.
- gives general analysis method and insights, which will be helpful to the community.

Cons

- well known optimizing techniques, and plain results, no much exciting things
- limited details on the effectiveness of optimizing techniques, e.g., the solution to fill the gap of RCMA and RCMB should be very cool, but...

References

Platt, J., 1998. Sequential minimal optimization: A fast algorithm for training support vector machines.

Y. You et al., Mic-svm: Designing a highly efficient support vector machine for advanced modern multi-core and many-core architectures, in Parallel and Distributed Processing Symposium, 2014 IEEE 28th International, 2014, pp. 809818.

The End