Training Support Vector Machines on Multiprocessors and GPUs

鄧偉祥
10567212

林翰緯
0456808

Outline

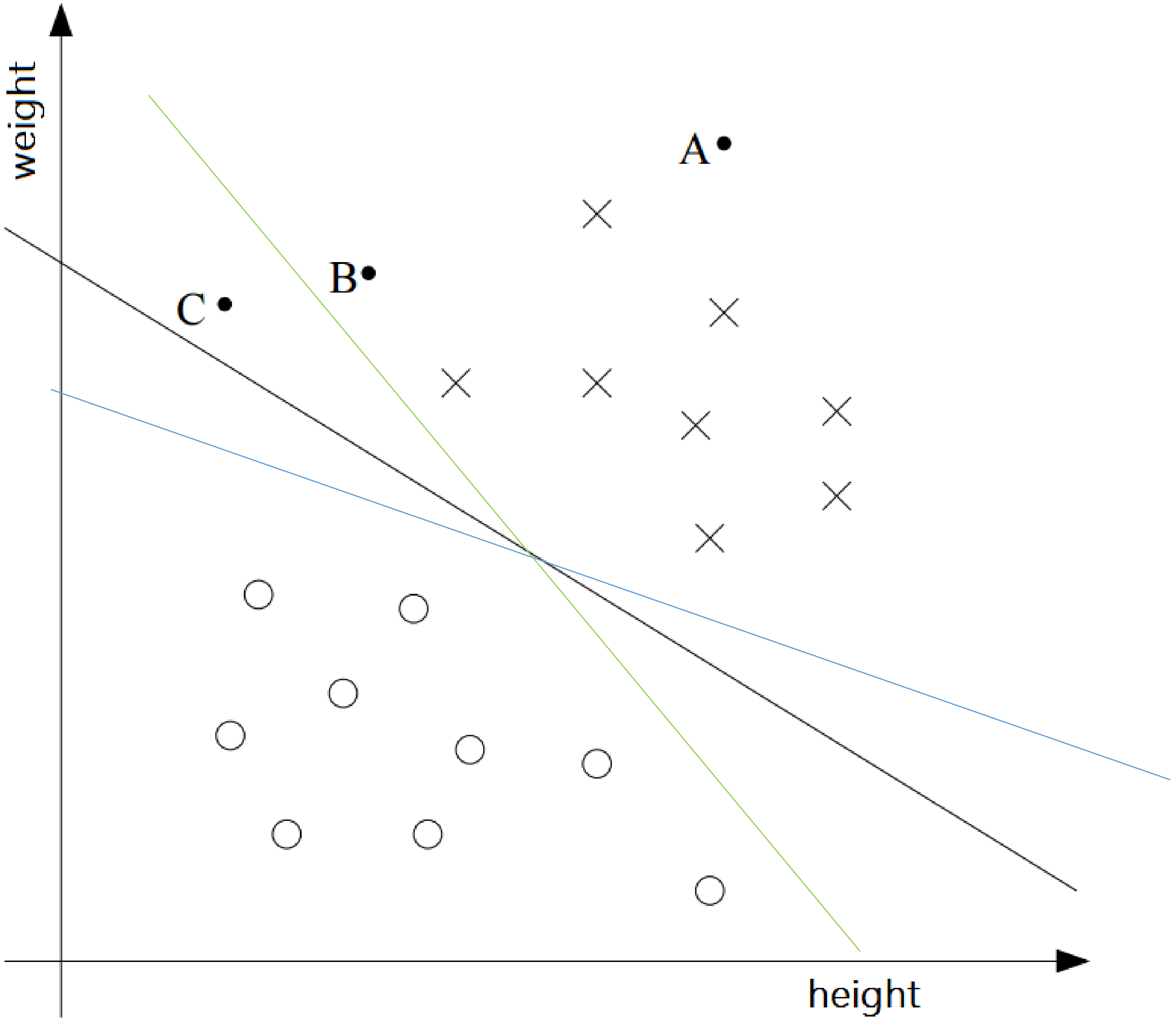
- SVM
  - heuristic instruction
  - math
  - SMO algorithm
  - serial version profiling
- MNIST database
  - data format
- MPI
  - flow chart
  - results
- CUDA
  - strategies
  - results
- References

SVM - heuristic instruction

- Support Vector Machine
- Optimal margin classifier
- Decision boundary

  $w^T x + b = 0$

- Solve w^T and b



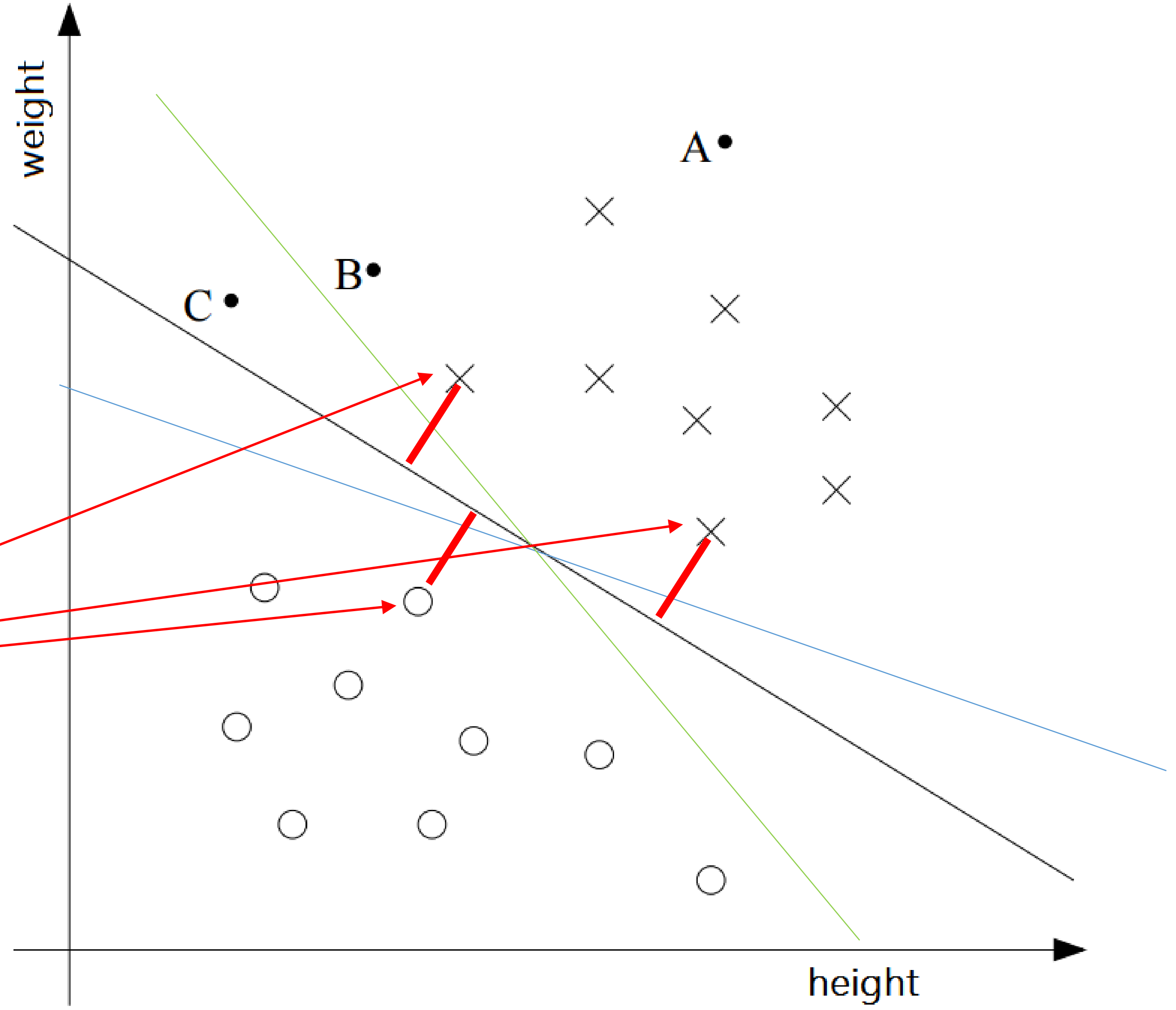
SVM - heuristic instruction

- Support Vector Machine
- Optimal margin classifier
- Decision boundary

  $$w^T x + b = 0$$

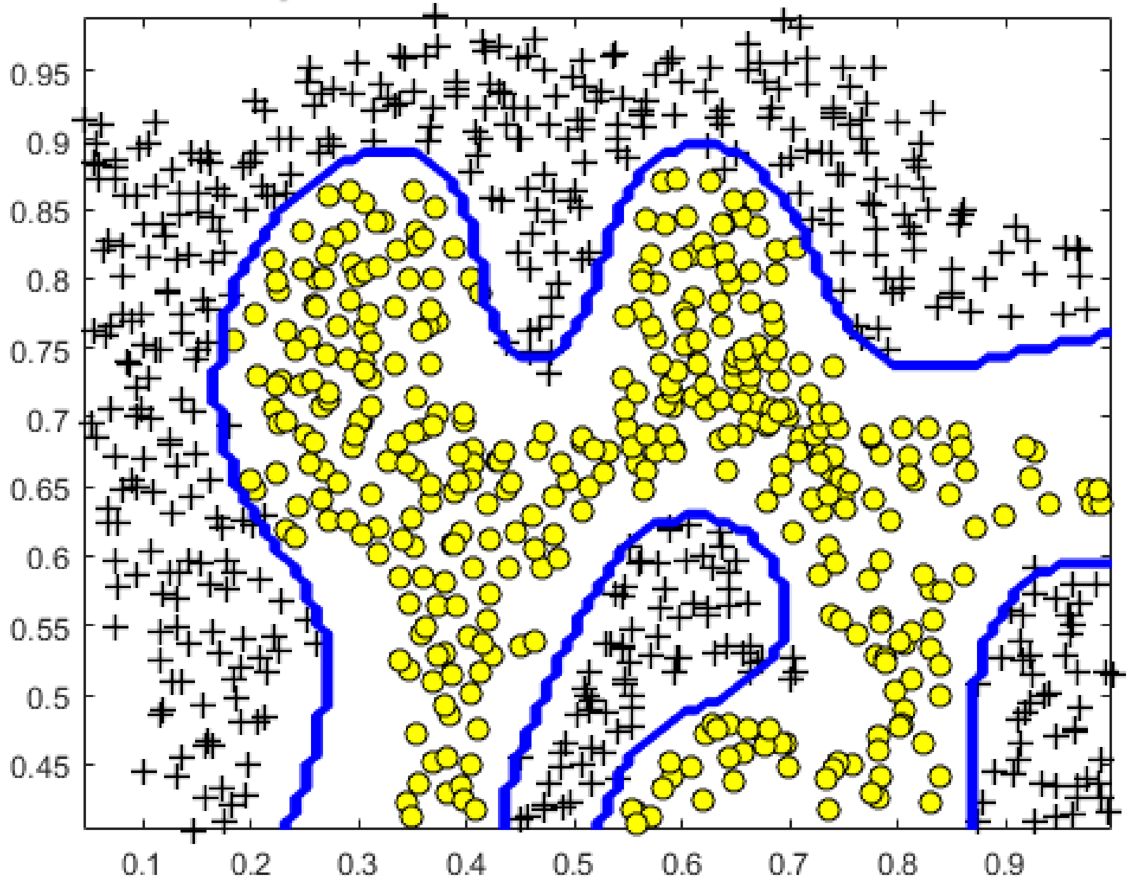
- Solve w^T and b

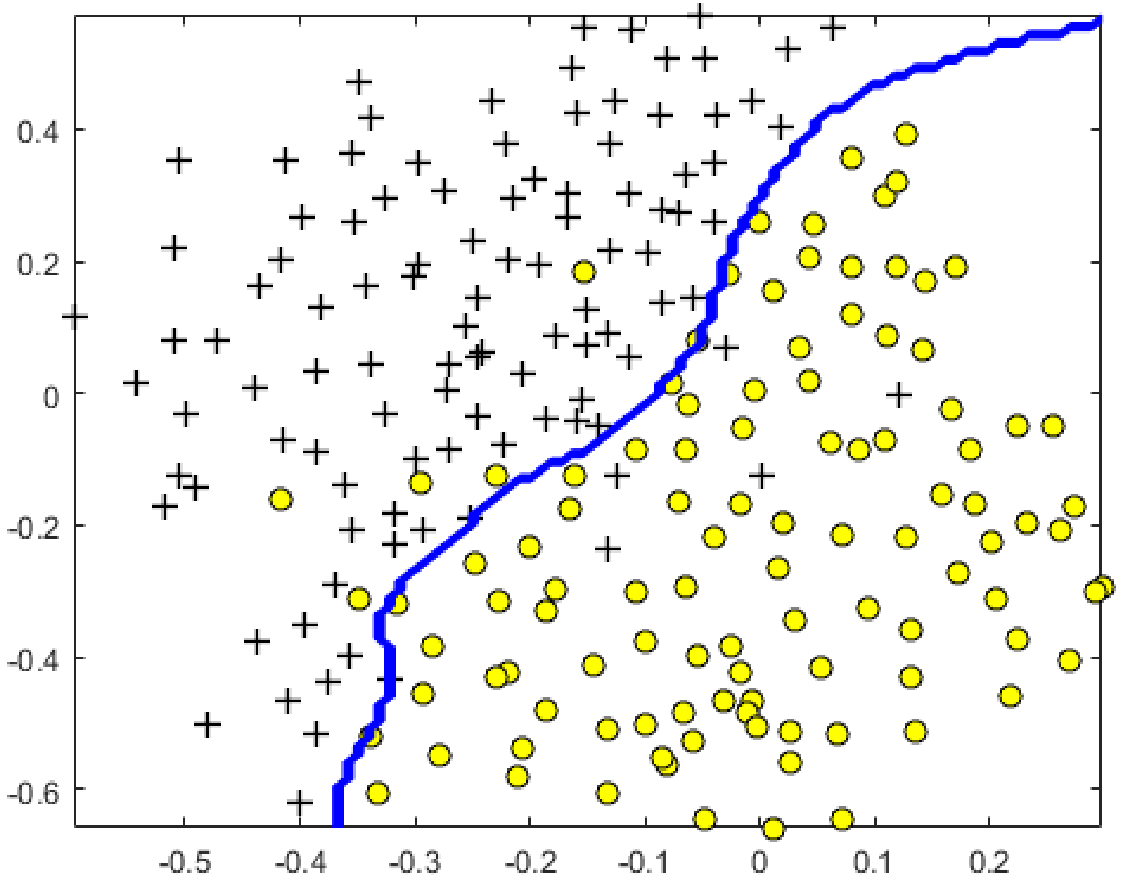
Support vectors



SVM - heuristic instruction

Examples:



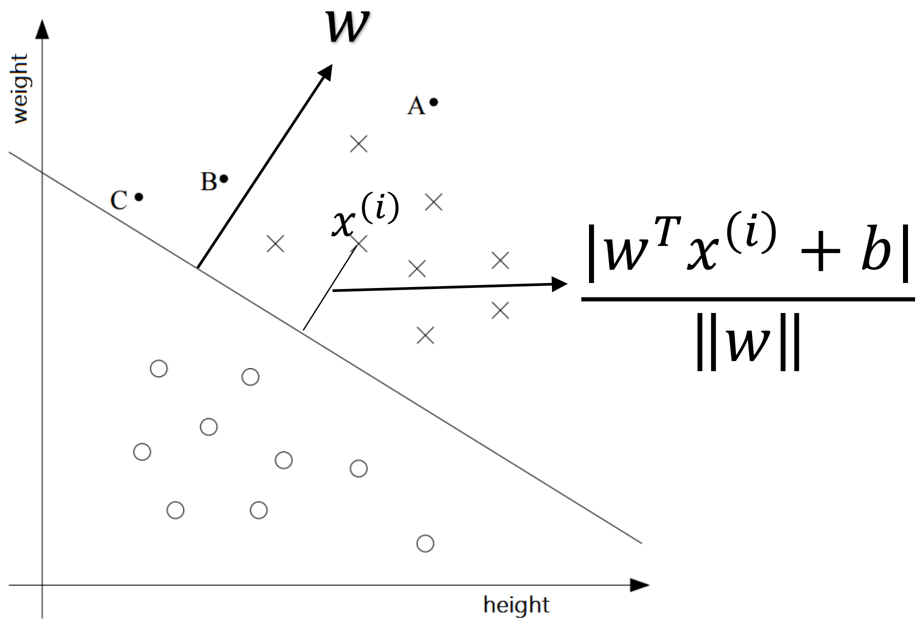


SVM - math

- $min_{w,b} \frac{1}{2} \|w\|^2$
  $s.t.\ y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1, i = 1, \dots, m$

- $max_\alpha\ W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle$
  $s.t.\ \alpha_i \geq 0, i = 1, \dots, m$
  $\sum_{i=1}^m \alpha_i y^{(i)} = 0$

- $w^T x + b = \left(\sum_{i=1}^m \alpha_i y^{(i)} x^{(i)}\right)^T x + b$
  $= \sum_{i=1}^m \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b$



SVM - math

- $min_{w,b} \frac{1}{2} \|w\|^2$
  $s.t.\ y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1, i = 1, \dots, m$

- $max_\alpha\ W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \left\langle x^{(i)}, x^{(j)} \right\rangle$
  $s.t.\ \alpha_i \geq 0, i = 1, \dots, m$
  $\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$

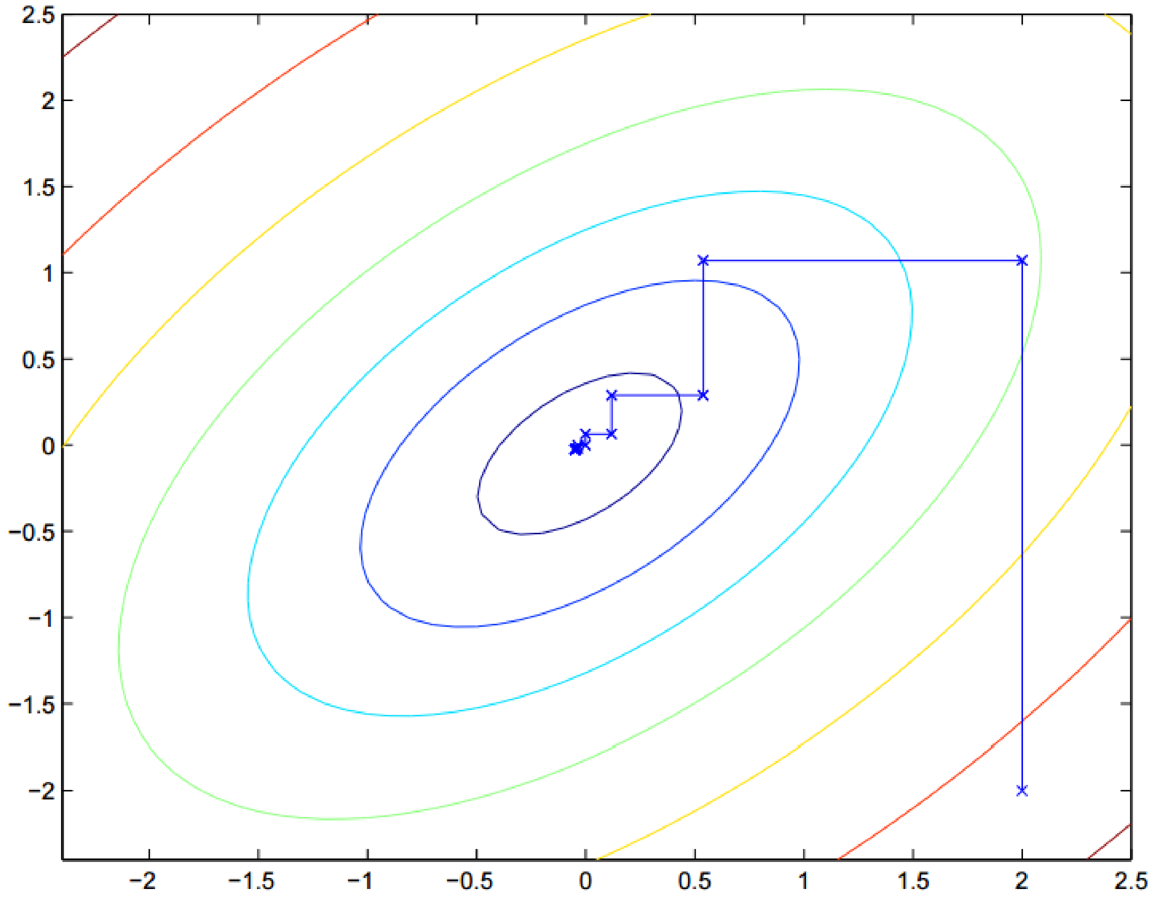
Input: training set $x^{(i)}, y^{(i)}$
Output: α_i, b

- $w^T x + b = \left(\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}\right)^T x + b$
  $= \sum_{i=1}^{m} \alpha_i y^{(i)} \left\langle x^{(i)}, x \right\rangle + b$

Input: test set x, y
Output: prediction accuracy

SVM – SMO algorithm

- Coordinate ascent



Loop until convergence: {

For i = 1, … , m {

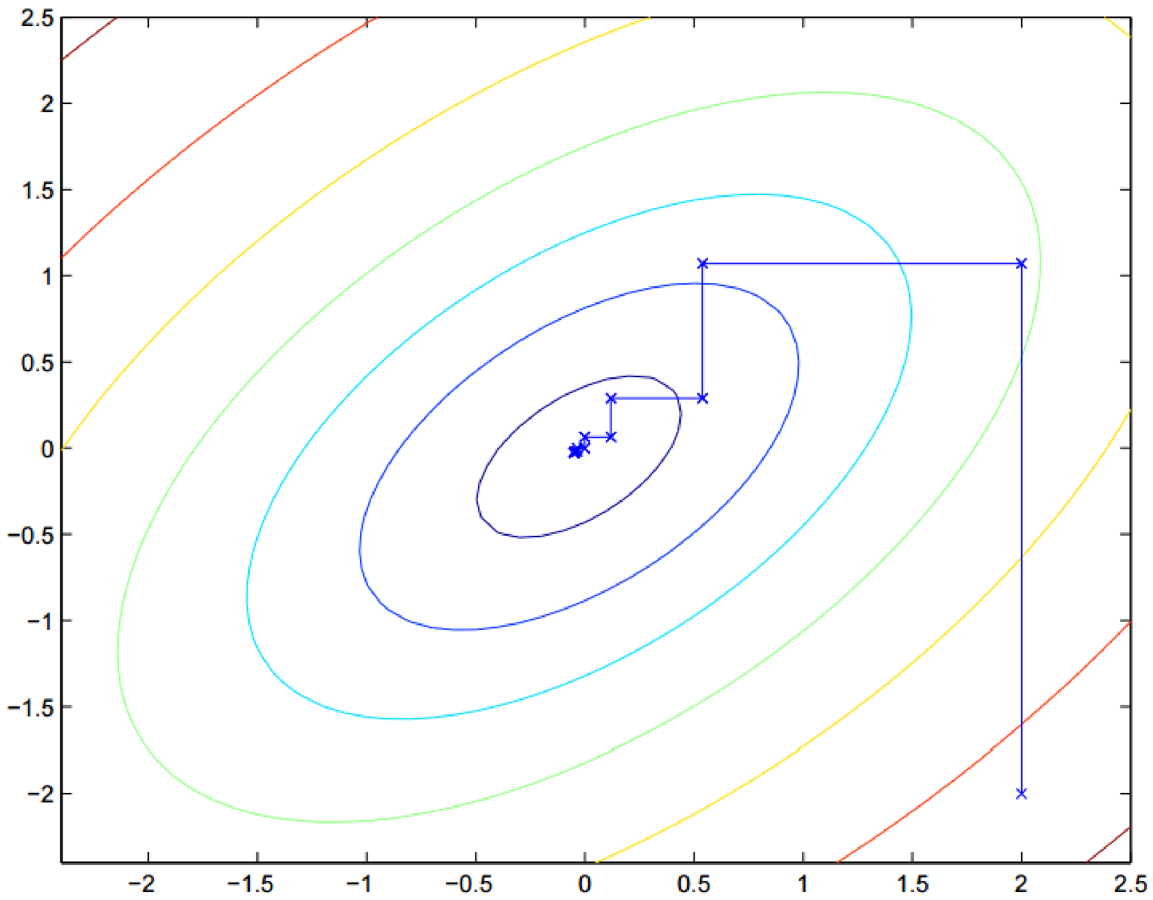
$$\alpha_i = \arg max_{\hat{\alpha}_i} W(\alpha_1, \ldots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \ldots, \alpha_m)$$

}

}

SVM – SMO algorithm

- Coordinate ascent



BUT

$$max_{\alpha}\, W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle$$

$s.t.\ \alpha_i \geq 0, i = 1, \dots, m$

$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$

We can't make any change to α_1 without violating this constraint

Repeat until convergence: {

1. Select some pair α_i and α_j to update next
2. Reoptimize $W(\alpha)$ with respect to α_i and α_j, while holding all the other α_k's $(k \neq i, j)$ fixed.

}

SVM – SMO algorithm

Algorithm 1 Sequential Minimal Optimization

Input: training data x_i, labels y_i, $\forall i \in \{1..l\}$
Initialize: $\alpha_i = 0$, $\boxed{f_i = -y_i}$, $\forall i \in \{1..l\}$
Compute: b_{high}, I_{high}, b_{low}, I_{low}
Update $\alpha_{I_{high}}$ and $\alpha_{I_{low}}$
repeat
  $\boxed{\text{Update } f_i,\ \forall i \in \{1..l\}}$
  Compute: b_{high}, I_{high}, b_{low}, I_{low}
  Update $\alpha_{I_{high}}$ and $\alpha_{I_{low}}$
until $b_{low} \leq b_{high} + 2\tau$

$$f_i = \sum_{j=1}^{m} \alpha_j y^{(j)} \langle x^{(i)}, x^{(j)} \rangle - y^{(i)}$$

$$f_i' = f_i + \left(\alpha'_{I_{high}} - \alpha_{I_{high}}\right) y^{(I_{high})} \left\langle x^{(i)}, x^{(I_{high})} \right\rangle + \left(\alpha'_{I_{low}} - \alpha_{I_{low}}\right) y^{(I_{low})} \left\langle x^{(i)}, x^{(I_{low})} \right\rangle$$

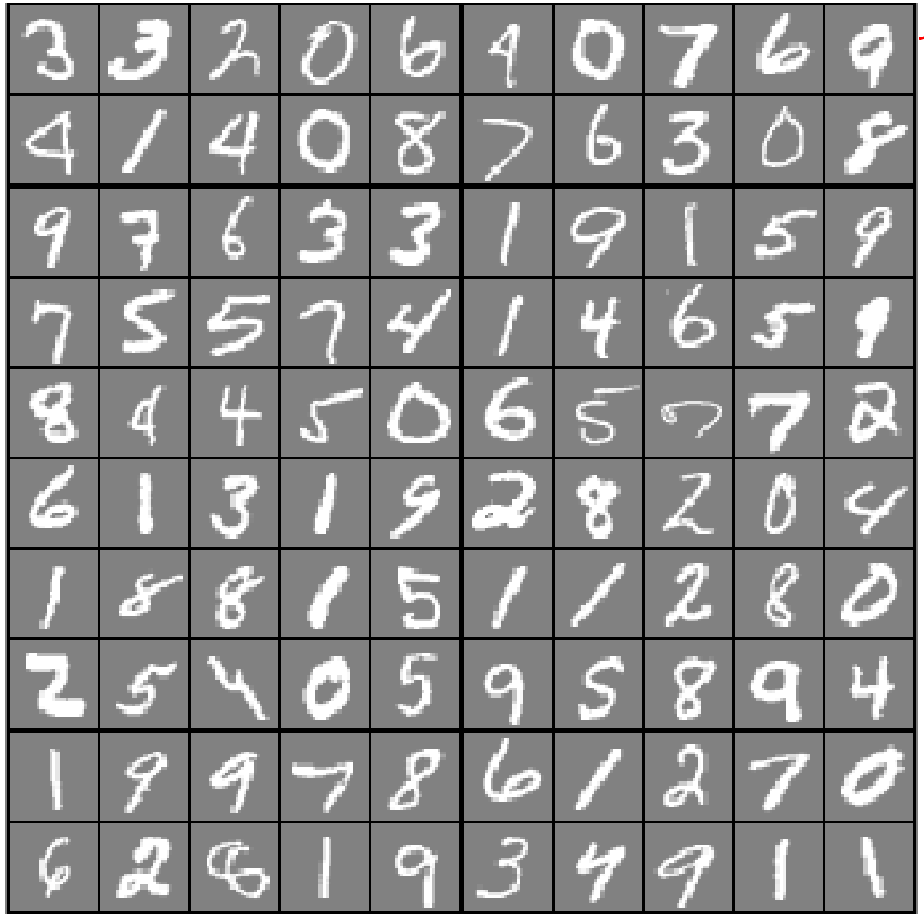
SVM – serial version profiling

Algorithm 1 Sequential Minimal Optimization

Input: training data x_i, labels y_i, $\forall i \in \{1..l\}$
Initialize: $\alpha_i = 0$, $f_i = -y_i$, $\forall i \in \{1..l\}$
Compute: b_{high}, I_{high}, b_{low}, I_{low}
Update $\alpha_{I_{high}}$ and $\alpha_{I_{low}}$
repeat
  $\boxed{\text{Update } f_i, \forall i \in \{1..l\}}$ >99%
  Compute: b_{high}, I_{high}, b_{low}, I_{low}
  Update $\alpha_{I_{high}}$ and $\alpha_{I_{low}}$
until $b_{low} \leq b_{high} + 2\tau$

```
[u10567212@gpuws-sslab CUDA]$ ./modified_SMO train-mnist-60000 train-mnist-60000-s.model 60000 784 1 0.001 0.001
computeNumChaned    : 0.232681 secs
update f_i          : 8944.073319 secs
update b_up, b_low  : 19.081569 secs
computeDualityGap   : 10.651465 secs
b = -7.189573
The total elapsed time is 8974.045847 seconds
total sv: 17704
```

MNIST database – data format



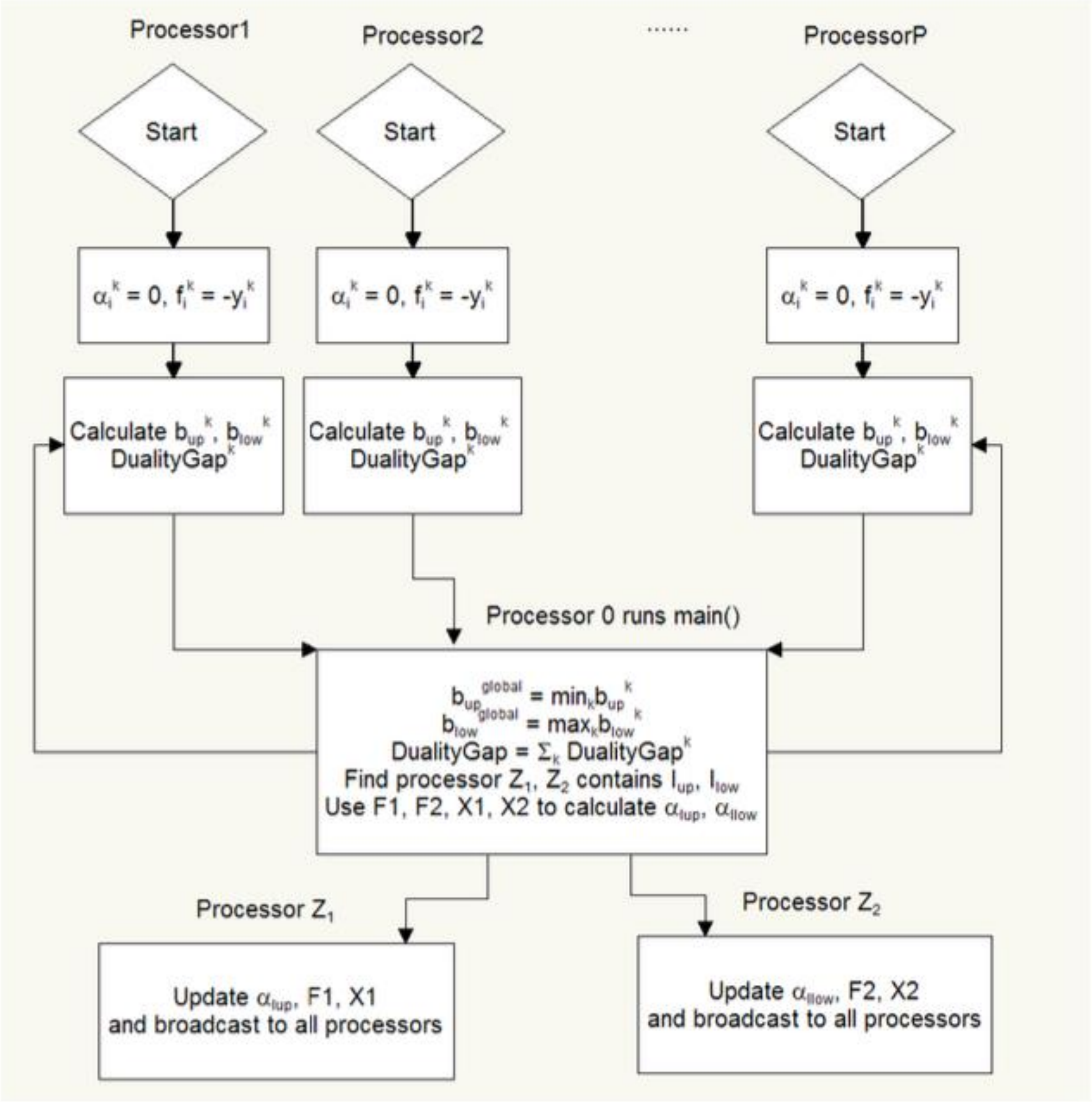
$[0\ 0\ 0\ 0\ \ldots 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \ldots 0.2157\ 0.5333 \ldots 0\ 0\ \ldots]$

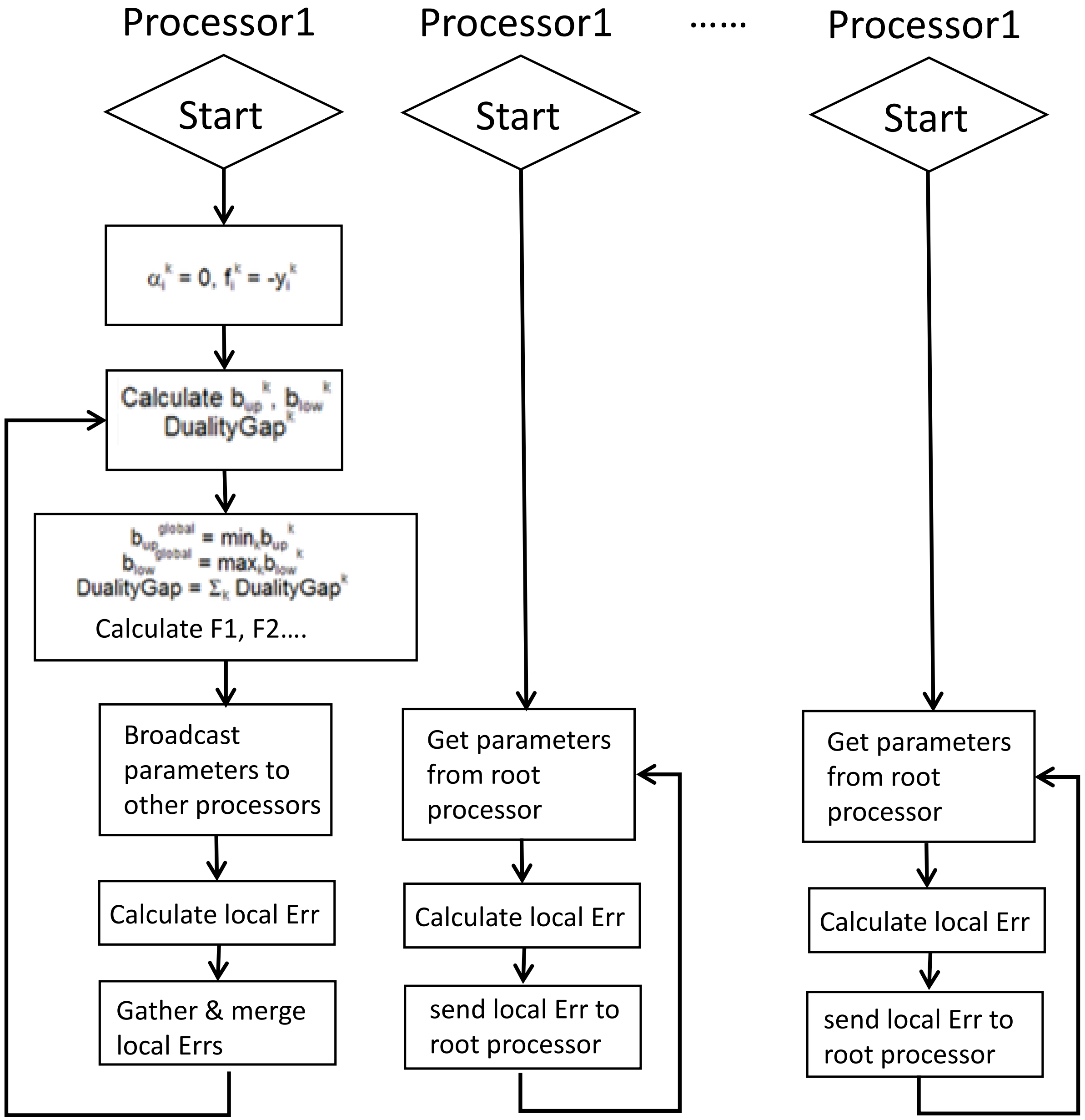
$28 \times 28 = 784$ dimensional vector

60000 handwritten digits for training
10000 for testing

	10000 training set	60000 training set
classification	2 or 5	Even or odd

MPI - flow chart





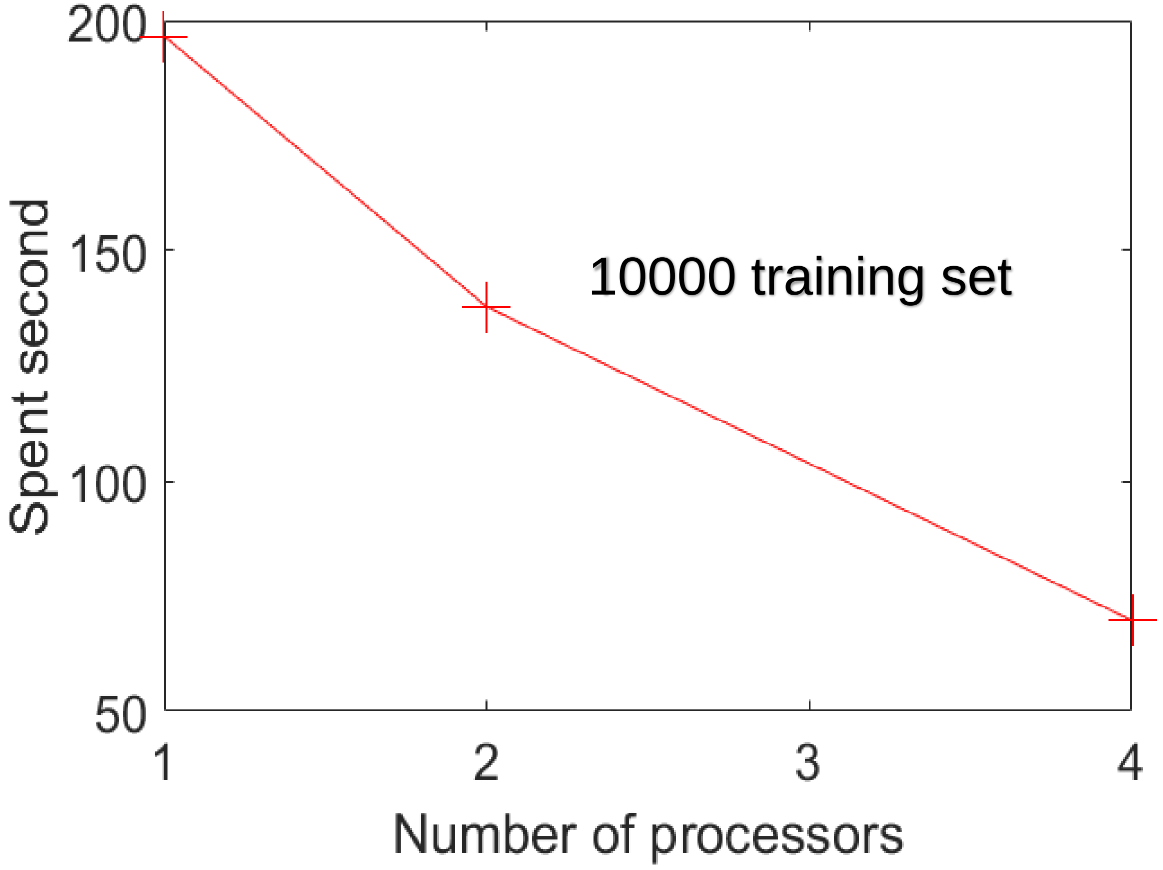
MPI - Serial Code Structure

```
while(DualityGap > prob->tau * ABS(Dual) && numChanged != 0)
{
        a1_old = prob->alphas[I_up];
        a2_old = prob->alphas[I_low];
        y1 = prob->y[I_up];
        y2 = prob->y[I_low];
        F1 = Err[I_up];
        F2 = Err[I_low];

        s1 = seconds();
        numChanged = computeNumChaned(prob, I_up, I_low, a1_old, a2_old, y1, y2, F1, F2, &Dual, &a1, &a2);
        t1 += (seconds() - s1);

        prob->alphas[I_up] = a1;
        prob->alphas[I_low] = a2;

        /* update Err[i] */
        s2 = seconds();

        for (i = 0; i < prob->size; i++) {
                Err[i] += (a1 - a1_old) * y1 * rbf_kernel(prob, I_up, i)
                        + (a2 - a2_old) * y2 * rbf_kernel(prob, I_low, i);
        }

        t2 += (seconds() - s2);

        s3 = seconds();
        computeBupIup(Err, prob, &b_up, &I_up);
        computeBlowIlow(Err, prob, &b_low, &I_low);
        prob->b = (b_low + b_up) / 2;
        t3 += (seconds() - s3);

        s4 = seconds();
        DualityGap = computeDualityGap(Err, prob);
        t4 += (seconds() - s4);

        num_iter++;
}
```

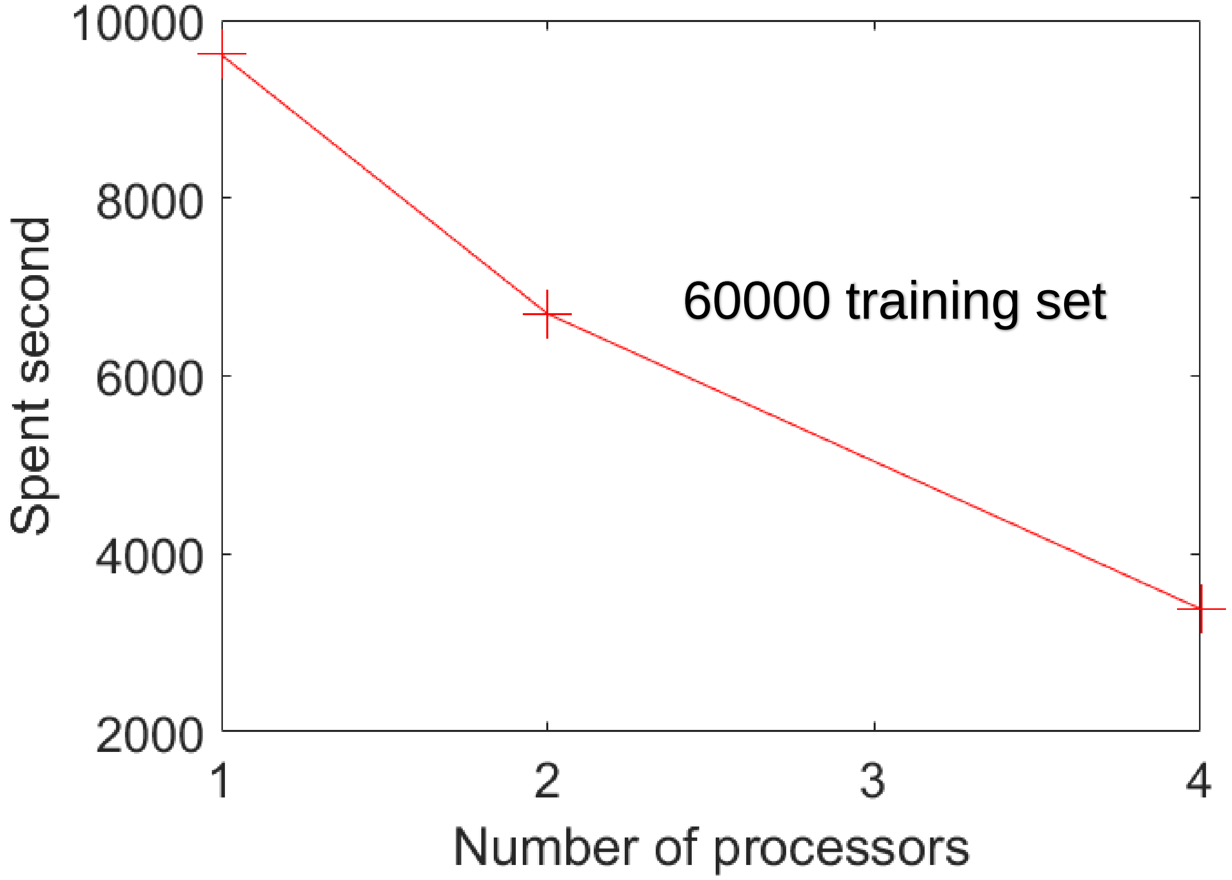
MPI - Code Structure

```
MPI_Bcast(Err, prob->size, MPI_FLOAT, 0, MPI_COMM_WORLD);

for (i=0; i<CLUSTER_SIZE; i++)
        clusterErr[i] = Err[i + my_rank*CLUSTER_SIZE];

syncLoopParam(&L, numChanged, Dual, DualityGap);

while(L.DualityGap > prob->tau * ABS(L.Dual) && L.numChanged != 0)
{
        if (my_rank == 0) {

                .....

                s2 = seconds();
        }
        syncParam(&P, a1, a1_old, a2, a2_old, y1, y2, I_up, I_low);

        for (i = 0; i < CLUSTER_SIZE; i++) {
                clusterErr[i] += (P.a1 - P.a1_old) * P.y1 * rbf_kernel(prob, P.I_up, i + my_rank*CLUSTER_SIZE)
                        + (P.a2 - P.a2_old) * P.y2 * rbf_kernel(prob, P.I_low, i + my_rank*CLUSTER_SIZE);
        }
        MPI_Gather(clusterErr, CLUSTER_SIZE, MPI_FLOAT, Err, CLUSTER_SIZE, MPI_FLOAT, 0, MPI_COMM_WORLD);

        if (my_rank == 0) {
                t2 += (seconds() - s2);

                .....

        }
        syncLoopParam(&L, numChanged, Dual, DualityGap);
}
```

MPI – results

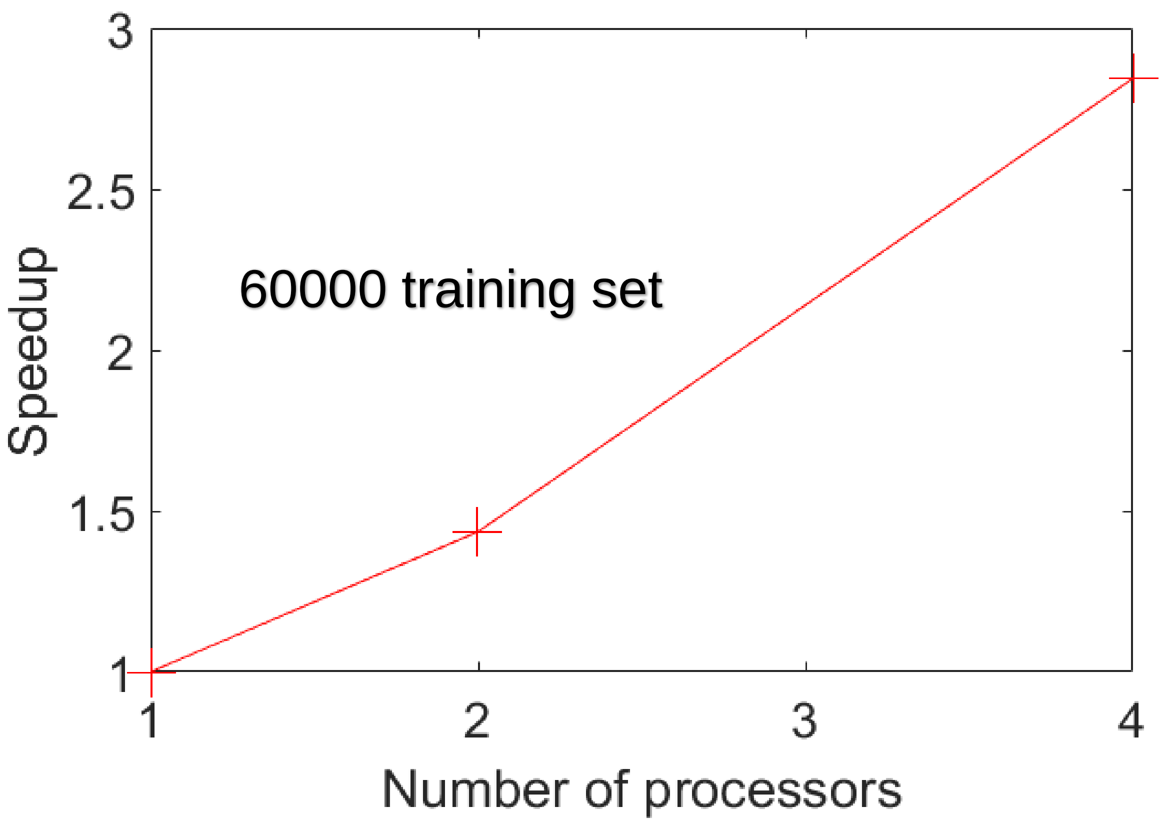


200
150
100
50
Spent second
1
2
3
4
Number of processors
10000 training set

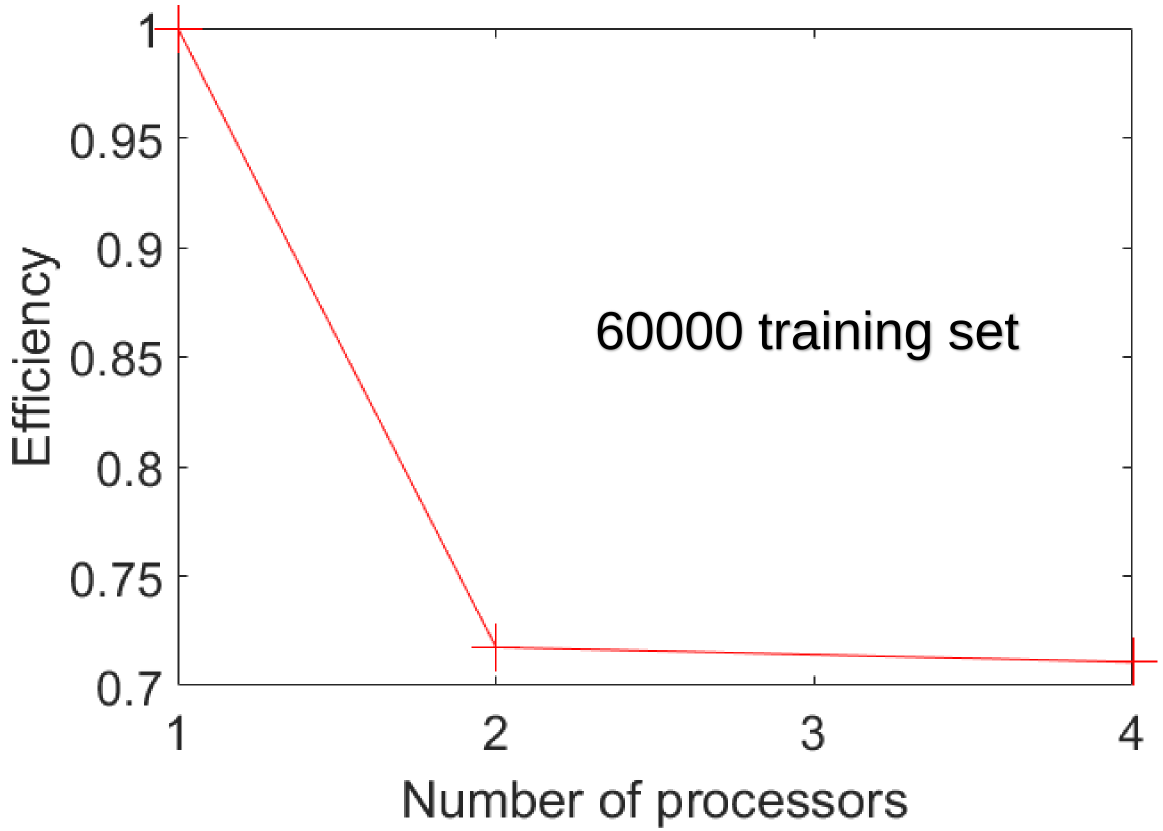


10000
8000
6000
4000
2000
Spent second
1
2
3
4
Number of processors
60000 training set

MPI – results



3
2.5
2
1.5
1
Speedup
60000 training set
1
2
3
4
Number of processors



1
0.95
0.9
0.85
0.8
0.75
0.7
Efficiency
60000 training set
1
2
3
4
Number of processors

MPI – results

	2P	4P
speedup	2.12	3.92

Class	LIBSVM	Sequential SMO	Parallel SMO					
			1P	2P	4P	8P	16P	30P
0	2931.668	3597.97	3948.83	1862.49	1006.46	483.51	283.19	210.10
1	2753.418	3717.91	3326.05	1845.33	895.45	462.50	266.70	196.09
2	5160.932	5644.19	5595.01	2781.18	1302.27	656.56	372.72	248.32
3	5737.956	6021.50	5404.18	2749.00	1330.94	703.06	399.22	271.97
4	5145.859	6044.60	6143.85	2771.65	1544.05	719.86	400.72	274.08
5	4825.642	5568.70	5529.62	2551.38	1408.74	655.09	378.62	267.57
6	3448.498	4232.65	4226.76	2099.81	973.81	491.43	294.33	194.78
7	5421.564	5788.88	5796.86	3124.36	1467.97	731.57	412.99	292.19
8	6565.783	7183.05	7243.13	3321.72	1800.28	822.35	468.53	314.70
9	7642.706	8033.80	7960.56	3645.48	1844.40	932.33	554.03	353.78
Averaged	4963.403	5583.325	5517.485	2675.24	1357.437	665.826	383.105	262.358

L.J. Cao, et al.
"Parallel sequential minimal optimization for the training of support vector machines".

CUDA – strategy 1

```
for (i = 0; i < prob->size; i++) {
        Err[i] += (a1 - a1_old) * y1 * rbf_kernel(prob, I_up, i)
                + (a2 - a2_old) * y2 * rbf_kernel(prob, I_low, i);
}
```

Matrix with size x size

CUDA – strategy 1

```
for (i = 0; i < prob->size; i++) {
        Err[i] += (a1 - a1_old) * y1 * rbf_kernel(prob, I_up, i)
                + (a2 - a2_old) * y2 * rbf_kernel(prob, I_low, i);
}
```

Matrix with size x size

Strategy: pre-calculate this matrix then use GPU to accelerate matrix multiplication

```
for (i = 0; i < prob->size; i++) {
        Err[i] += (a1 - a1_old) * y1 * K[I_up * prob->size + i]
                + (a2 - a2_old) * y2 * K[I_low * prob->size + i];
}
```

CUDA – strategy 1

- It works!

  For 10000 training data, serial version takes 181.48 second while CUDA version takes 5.16 second.

  Speedup is 35.2

```
[u10567212@gpuws-sslab CUDA]$ ./modified_SMO train-mnist-10000 train-mnist-10000-s.model 10000 784 1 0.001 0.001
computeNumChaned    : 0.027730 secs
update f_i          : 180.890815 secs
update b_up, b_low  : 0.365614 secs
computeDualityGap   : 0.194550 secs
b = 0.203094
The total elapsed time is 181.479650 seconds
total sv: 1366
```

```
[u10567212@gpuws-sslab CUDA]$ ./mat_modified_SMO train-mnist-10000 train-mnist-10000-matrix-cuda.model 10000 784 1 0.001 0.001
b = 0.048123
total sv: 1366
The total elapsed time is 5.157143 seconds
```

CUDA – strategy 1

- It works!

  For 10000 training data, serial version takes 181.48 second while CUDA version takes 5.16 second.

  Speedup is 35.2

```
[u10567212@gpuws-sslab CUDA]$ ./modified_SMO train-mnist-10000 train-mnist-10000-s.model 10000 784 1 0.001 0.001
computeNumChaned     : 0.027730 secs
update f_i           : 180.890815 secs
update b_up, b_low   : 0.365614 secs
computeDualityGap    : 0.194550 secs
b = 0.203094
The total elapsed time is 181.479650 seconds
total sv: 1366
```

```
[u10567212@gpuws-sslab CUDA]$ ./mat_modified_SMO train-mnist-10000 train-mnist-10000-matrix-cuda.model 10000 784 1 0.001 0.001
b = 0.048123
total sv: 1366
The total elapsed time is 5.157143 seconds
```

But… it needs a amount of memory, can’t apply for 60000 training data!

CUDA – strategy 2

```
/*********************************************************************
 *     Update f_i
 *********************************************************************/
__global__ void update_fi(float *devErr, float *devX, float a1, float a2, float a1_old, float a2_old, int y1, int y2, int I_up, int I_l
ow, float gamma, int dim, int size) {
        int i = blockIdx.x * blockDim.x + threadIdx.x;
        float k1 = 0, k2 = 0;
        if (i < size) {
                for (int m = 0; m < dim; m++)
                {
                        k1 += (devX[I_up * dim + m] - devX[i * dim + m]) * (devX[I_up * dim + m] - devX[i * dim + m]);
                        k2 += (devX[I_low * dim + m] - devX[i * dim + m]) * (devX[I_low * dim + m] - devX[i * dim + m]);
                }
                k1 = expf(-1 * gamma * k1);
                k2 = expf(-1 * gamma * k2);
                devErr[i] += (a1 - a1_old) * y1 * k1 + (a2 - a2_old) * y2 * k2;
        }
}
```

CUDA – result

#SV

	Serial version	CUDA version1	CUDA version2
10000 training set	1366	1366	1366
60000 training set	17704	-	17879

b

	Serial version	CUDA version1	CUDA version2
10000 training set	0.203094	0.048123	0.048117
60000 training set	-7.189573	-	-6.730177

Test accuracy

	Serial version	CUDA version1	CUDA version2
10000 training set	0.982 (1473/1500)	0.982 (1473/1500)	0.982 (1473/1500)
60000 training set	0.936 (9358/10000)	-	0.935 (9350/10000)

CUDA – result

Time (sec)

	Serial version	CUDA version2	Speedup
10000 training set	181	5.7	31.7
60000 training set	8974	189	47.4

TABLE 6.3. *C*-SVC Training Results

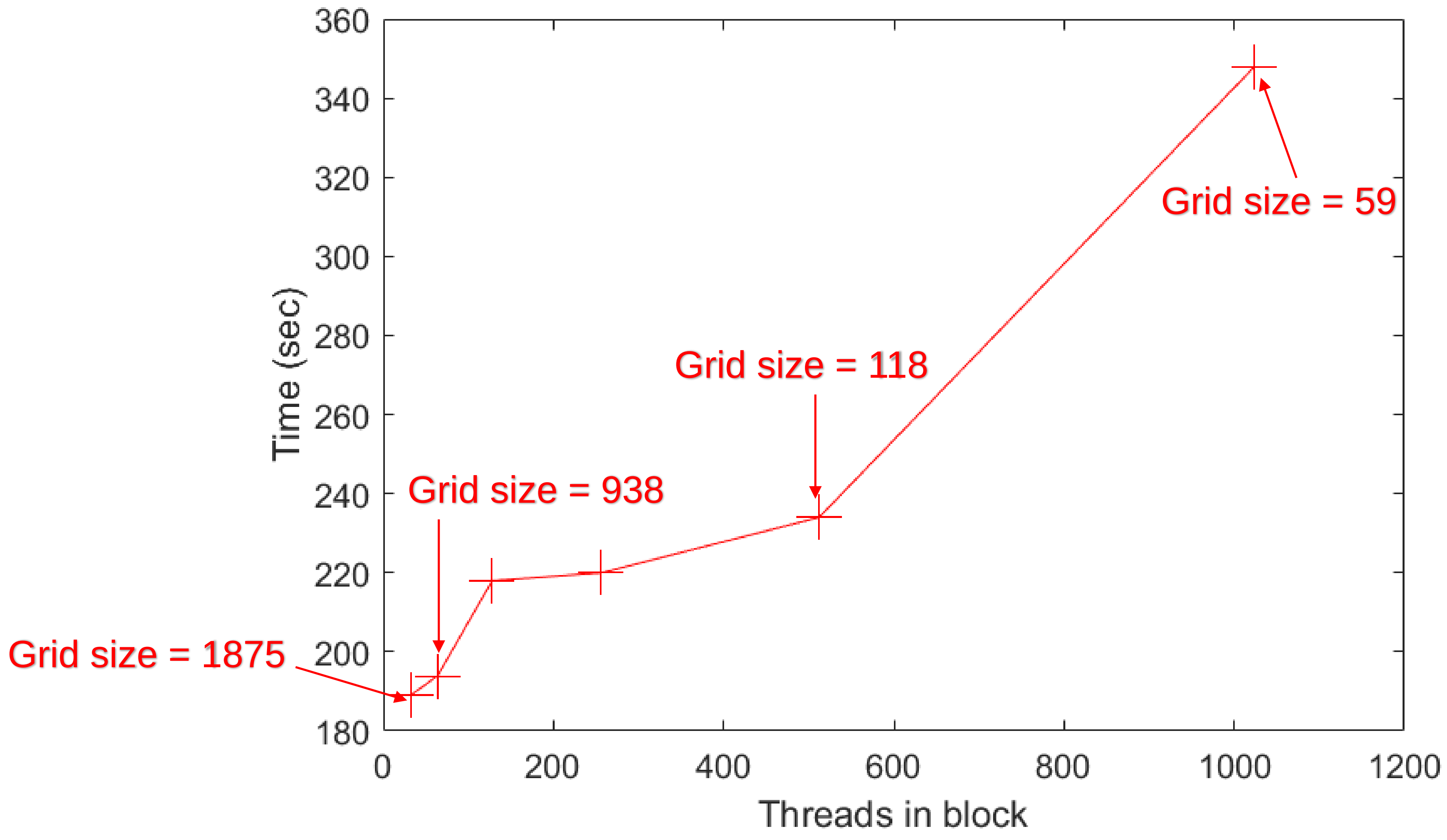
Dataset	# Support Vectors		Abs Difference in b	Training Time (s)		Speedup (x)
	cuSVM	LIBSVM		cuSVM	LIBSVM	
Adult	18,676	19,059	2.8×10^{-6}	31.6	541.2	17.1
Web	35,220	35,231	2.6×10^{-4}	228.3	2,906.8	12.7
MNIST	43,751	43,754	2.0×10^{-7}	498.9	17,267.0	34.6
Forest	270,305	270,304	8.0×10^{-3}	2,016.4	29,494.3	14.1

A. Carpenter. "CUSVM: A CUDA implementation of support vector classification and regression".

Dataset	# support vectors		Abs difference In b	Training Time(s)		Speedup (x)
	CUDA	Serial		CUDA	Serial	
MNIST	43397			771.2		

CUDA – result

Different Execution Configuration:



References

- http://cs229.stanford.edu/

  CS229 Lecture notes part V: Support Vector Machines by Andrew Ng.
- S.S. Keerthi, S.K. Shevade, C. Bhattaacharyya and K.R.K. Murthy. “Improvements to Platt’s SMO algorithm for SVM classifier design” *Neural Computation*, Vol. 13, pp. 637-649, 2001.
- L.J. Cao, et al. “Parallel sequential minimal optimization for the training of support vector machines”. *Neural Networks, IEEE Transactions on*, 17(4):1039-1049, July 2006.
- A. Carpenter. “CUSVM: A CUDA implementation of support vector classification and regression”. Technical report (2009)

Thank you