Parallel Support Vector Machine

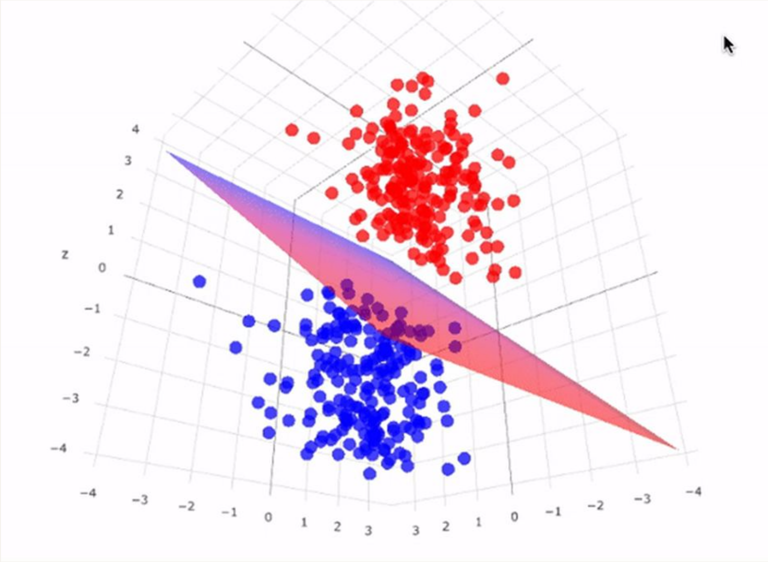
Wesley Osogo, Tianfan Xu, Connor Buchheit, Peter Chen

Problem – SVM

- **SVM: Finding the optimal hyperplane that separates a set of points belonging to two classes.**

- **Training an SVM model is computationally expensive, and scales poorly with dataset size, so training is *slow* on large datasets.**

- **Traditionally, coding SVM relies on Quadratic Programming (QP) for optimization, which is the main bottleneck. We aim to use existing techniques to reframe this problem as a parallel one, thus allowing for usage of MPI so that the model can accommodate larger datasets.**



Data Generation

- Data Fetching: dataset from the UCI repository, which includes features stored in X and targets stored in y.

- Standardization: The features X are standardized using StandardScaler from sklearn.preprocessing. This normalization ensures that each feature contributes equally to the analysis by giving them mean of zero and a standard deviation of one.

- Transformation of Targets: The target variable y is transformed based on a midpoint value calculated as the average of the maximum and minimum values of y. Each value in y is mapped to -1 if it is below this midpoint, otherwise to 1.

Math Model

Solve Quadratic Programming(QP)

$$\min \quad \mathcal{P}(\mathbf{w}, b, \boldsymbol{\xi}) = \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1} \xi_i$$

$$s.t. \quad 1 - y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) \le \xi_i, \quad \xi_i > 0,$$

$$\min \quad \mathcal{D}(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{Q}\boldsymbol{\alpha} - \boldsymbol{\alpha}^T\mathbf{1}$$

$$s.t. \quad \mathbf{0} \le \boldsymbol{\alpha} \le \mathbf{C}, \ \mathbf{y}^T\boldsymbol{\alpha} = 0,$$

Interior Point Methods(IPM)

IPM: Solve QP with IPM, with computation bottleneck on matrix inverse in SVM

Parallel Incomplete Cholesky Factorization(ICF)

ICF: approximate a positive definite matrix (kernel matrix) with a lower triangular matrix
Parallel Factorization: Each machine performs ICF on its subset of the data independently

Parallel Interior Point Methods(IPM)

PIPM: Sherman-Morrison-Woodbury formula

$$\begin{aligned}
\Sigma^{-1}z &= (D+Q)^{-1}z \approx (D+HH^T)^{-1}z \\
&= D^{-1}z - D^{-1}H(I + H^TD^{-1}H)^{-1}H^TD^{-1}z \\
&= D^{-1}z - D^{-1}H(GG^T)^{-1}H^TD^{-1}z.
\end{aligned}$$

Parallel Support Vector Machine Implementation

Stages: Kernel Computation, Parallel ICF, Parallel IPM, Prediction

Scientific Library: Eigen(with row based memory layout)

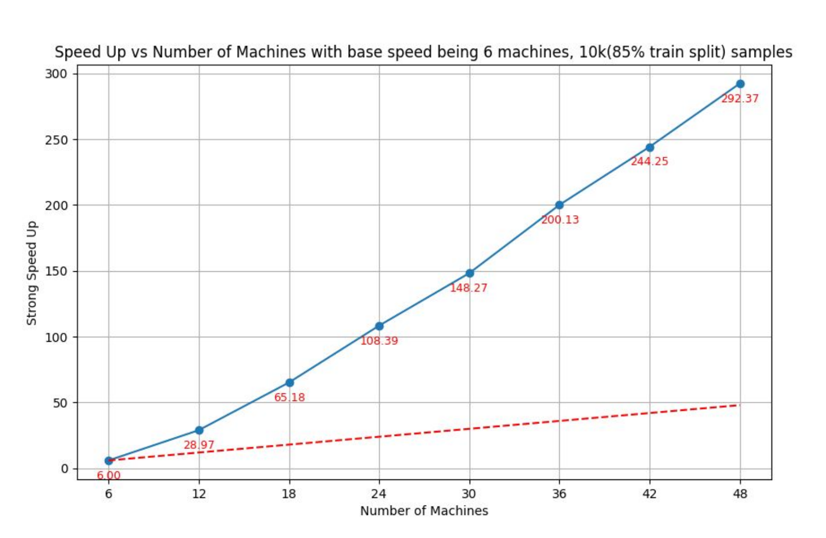
<u>Kernel Compute:</u>
A single process read input data and computes kernel matrix locally through OpenMP

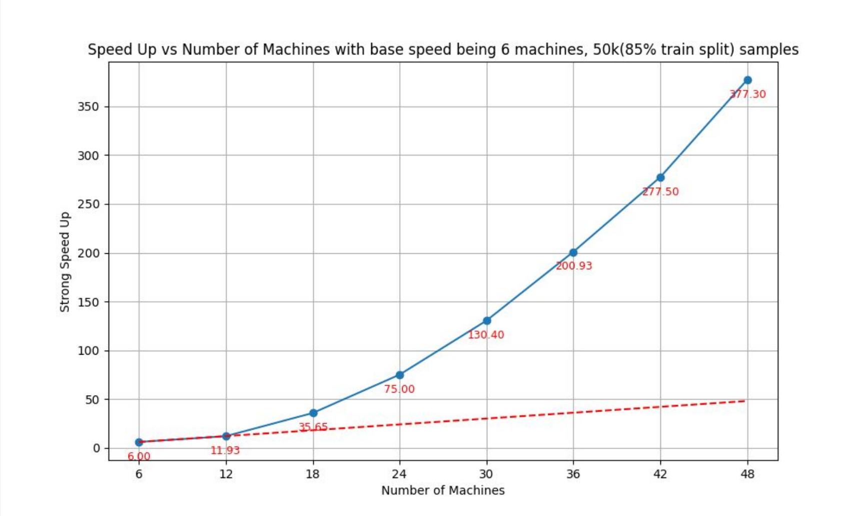
<u>Parallel ICF:</u>
Distribute different rows of computed kernel to distributed machines. Use OpenMp to parallelize local updates and MPI to decide pivot value/index and stopping condition

<u>Parallel IPM:</u>
Use SMW formula to compute necessary parts locally and MPI to communicate, alleviating the heaviest computational task of solving inverse system

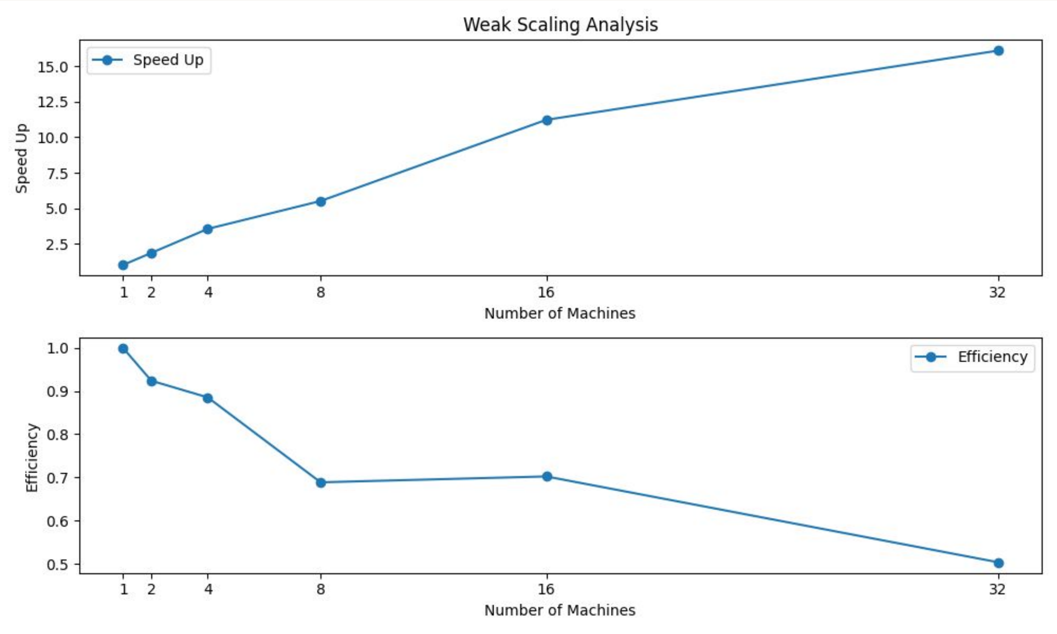
<u>Prediction:</u>
Each machine computes with the local weights and gathers result through MPI

Strong Speed up (10k and 50k)

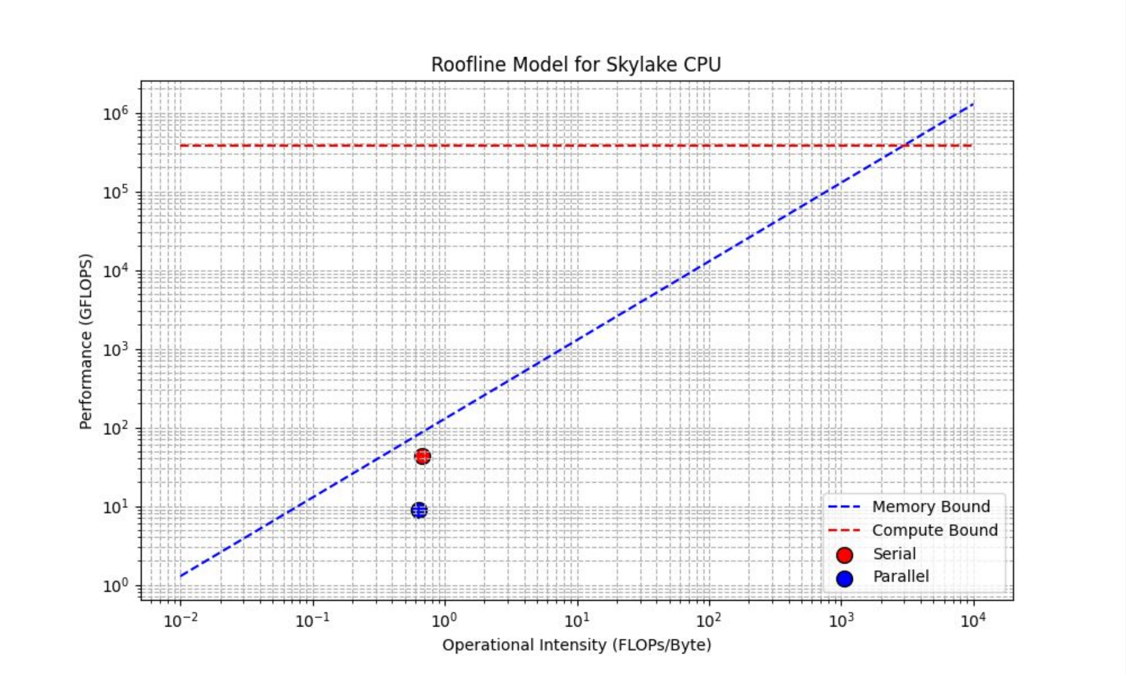




Performance – Weak Scaling Analysis



Performance – Roofline Model



Compute Bound

8 cores
* 3.0 GHz
* 16 Flops/Cycle/Core

Memory Bandwidth

50 Gb/s

Next Steps

1) Performance — We saw a test accuracy of ~80% for most applications. Although we are very pleased with this performance as an initial result, experimenting with different SVM kernels and parameters could lead to even better performance.
2) Extreme scalability — Taking measures such as dimensionality reduction for our features to reduce problem complexity would likely yield even better results than we achieved.We would also have worked on calculating kernel matrix Q distributedly as it limits scaling beyond memory capacity of one machine.
3) CUDA — Our code relies heavily on matrix multiplication. Given more time and access to GPUs, applying what we learned about CUDA and running our code on a GPU would result in even greater speedups than observed.

Given more time, we would have experimented with such techniques to make a much more fine-tuned model.

Appendix

Strong Scaling

# Machines	Train time Avg(s)
6	15724.86
12	7908.88
18	2646.64
24	1257.97
30	723.533
36	469.573
42	339.994
48	250.062
Train Size	50k
Iters	50
Accuracy	84%

# Machines	Train time Avg(s)
6	687.485
12	142.406
18	63.2843
24	38.0546
30	27.8197
36	20.6108
42	16.8879
48	14.1087
Train Size	10k
Iters	50
Accuracy	71%

Weak Scaling

Machines	Work Size	Time_1(s)	Time_2(s)	Time_3(s)	Avg Time(s)
1	1600	462.292	460.79	443.29	455.4573
2	3200	494.647	490.527	493.555	492.9097
4	6400	515.204	514.625	513.368	514.399
8	12800	660.103	662.108	661.896	661.369
16	25600	649.446	648.785	647.518	648.583
32	51200	904.972	904.657	903.967	904.532