LASSO Parallel with MPI

Yihe Yu

May 7, 2021
LASSO is short for least absolute shrinkage and selection operator.

It is a **regression analysis method** that performs both variable selection and regularization.

It enhances the prediction accuracy and interpretability of the resulting statistical model.

It has a variety of interpretations in terms of geometry, Bayesian statistics and convex analysis.

It helps in the models analysis and provide an **optimum linear combination**.

Its applications include **cross-section of return forecasts** and asset portfolio management etc.
Lasso: Formulation

A subset-selection problem in **linear regression**:\[ y = X\beta \]

where $y$ is $n \times 1$, $X$ is $n \times K$, $\beta$ is $K \times 1$. $n$ is the sample size, $K$ is the number of features (candidate variables).

We can solve $\beta$ by

\[
\min_{\beta \in \mathbb{R}^p} \left\{ \frac{1}{2} \| y - X\beta \|_2^2 + \lambda \| \beta \|_1 \right\}
\]
We can solve the optimization problem by considering it as an OLS problem with a constraint, i.e.,

\[
\beta^{OLS} = (X^T X)^{-1} X^T Y
\]

s.t.

\[
\sum_{j=1}^{K} |\beta_j| \leq c
\]
LASSO: Computational Complexity

For \((X^T X)^{-1} X^T Y\),

- \((X^T X)\) takes \(O(nK^2)\) time and produces a \((K \times K)\) matrix.
- The inverse of a \((K \times K)\) matrix takes \(O(K^3)\) time.
- \((X^T Y)\) takes \(O(nK^2)\) time and produces a \((K \times K)\) matrix.
- The final matrix multiplication of two \((K \times K)\) matrices takes \(O(K^3)\) time.

The computational complexity of LASSO implemented using LARS algorithm (Efron et al., 2004) is \(O(K^3 + nK^2)\).
For typical LASSO settings $K \gg n$, so the computational complexity $O(K^3 + nK^2)$ then become $O(K^3)$.

Therefore the data parallelism which divide the matrix $X$ along example dimension $n$ does not boost the regression process.

A possible way to improve the performance is to apply MPI to the matrix multiplication.
Consider matrix multiplication $M_1 \cdot M_2 = M_3$. 

![Diagram of matrix multiplication using MPI functions](image)
Algorithm 1 LASSO coefficients

**Input:** DataSet\((X, Y)\), \(lr\) (learning rate), \(p\) (penalty)

**Output:** \(\beta\) (LASSO coefficients)

1: \textbf{for} \(X, Y\) in DataSet \textbf{do}
2: \hspace{2em} \(Y_{\text{pred}} \leftarrow X \cdot \beta\)
3: \hspace{2em} \(d\beta \leftarrow (-2 \cdot X \cdot (Y - Y_{\text{pred}}) + I(\beta) \cdot p)/X.\text{shape}[0]\)
4: \hspace{2em} \(\beta \leftarrow \beta - lr \cdot d\beta\)
5: \hspace{2em} \textbf{end for}
6: \hspace{2em} \textbf{return} \(\beta\)
Results

We conduct our experiment on UB-CCR debug partition nodes, which has 12 cores per node.

- If cores smaller than 12, deploy on 1 node, else on 2 nodes
- For MPI, 1 core as master and the rest as computational cores
- Matrix Multiplication: \( M_1 \cdot M_2 \), each matrix is of \( 500 \times 500 \)
- LASSO: 30 examples, each example has 250 features
Results

We compare the time using in total 24 cores on 6, 8, 12 and 24 nodes for the Matrix Multiplication.

<table>
<thead>
<tr>
<th># nodes * # codes per node</th>
<th>Time (s)</th>
<th>Speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 * 4</td>
<td>17.954</td>
<td>1x</td>
</tr>
<tr>
<td>8 * 3</td>
<td>18.973</td>
<td>0.946x</td>
</tr>
<tr>
<td>12 * 2</td>
<td>18.434</td>
<td>0.974x</td>
</tr>
<tr>
<td>24 * 1</td>
<td>12.253</td>
<td>1.465x</td>
</tr>
</tbody>
</table>
Conclusion

- MPI in matrix multiplication and LASSO achieves linear speedup wrt. number of cores on single node.
- LASSO has similar speed up with matrix multiplication, which shows the correctness of our computational complexity analysis and implementation.
- MPI on multiple nodes may suffer from the communication as shown in the previous section, the best performance is achieved when we utilize all the cores.
References


Thank you