Parallel Parameter Update for Deep Neural Network

CSE633 Parallel Algorithm (Spring 2018)
Instructor: Prof. Russ Miller

Presenter: Xin Ma
xma24@buffalo.edu
05/03/2018
Outline

• Background

• Parallel Parameter Update

• Evaluation

• Conclusions
Background

Artificial Neural Network (ANN) model

MNIST data set

60,000 training samples

784

60,000

13.3 MB
Background

Convolutional Neural Network (CNN) model

Convolutional operation

Pooling operation

MNIST data set
Parallel Parameter Update

Sequentially

Batch 0

Batch 1

How to improve it?

Is it possible to run the program in a parallel way?

Method 1: Run the python program using multiple threads at the same time

Method 2: Run the python program with different processes (Nodes)
  - Assume each node with one available process
  - Using massage passing to cooperative those python programs

GIL - Global Interpreter Lock
Parallel Parameter Update

Process 0; Node 0
- Batch 0: Sequentially
- Batch 1

Process 1; Node 1
- Batch 0
- Batch 1: Parallely

Process 2; Node 2
- Batch 1

Exchange Gradients
Experiment Setup

Settings:

• ANN model:
  • hidden nodes: \([10, 50]\); regularization parameter: 5;
  • # of hidden layer: 1; # of params: \(795.2 \times \#_{\text{hidden}}\).

• CNN model:

  ```python
  def CNN():
    (conv1): Sequential(
      (0): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
      (1): ReLU()
      (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
    )
    (conv2): Sequential(
      (0): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
      (1): ReLU()
      (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
    )
    (out): Linear(in_features=1568, out_features=10, bias=True)
  ```

  • # of parameters: 28938.

• MPI:
  • Scatter() and Allgather().
Experiment Results

Average run time of each iteration with different # of nodes:

<table>
<thead>
<tr>
<th>Node number</th>
<th>1</th>
<th>3</th>
<th>6</th>
<th>9</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Time (s)</td>
<td>3.50104</td>
<td>9.1768</td>
<td>10.5955</td>
<td>12.57973</td>
<td>13.45111</td>
</tr>
</tbody>
</table>
Experiment Results

Average run time of each iteration with different batch size:

- Parameters -> Same
- One-hop connection

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Time (s)</td>
<td>9.1768</td>
<td>11.94646</td>
<td>12.99369</td>
<td>11.68846</td>
</tr>
</tbody>
</table>
Experiment Results

Recognition accuracy with different batch size:

- Large batch size is good for model convergence.
Experiment Results

Recognition accuracy with different batch size:

- Nodes: 1; Batch Size: 5
- Nodes: 3; Batch Size: 5
- Nodes: 1; Batch Size: 15
Experiment Results

Recognition accuracy with different # of nodes:

If Batch Size is large enough, More parallel nodes ---> may not accelerate the convergence.
Experiment Results

Recognition accuracy with different # of nodes:

More parallel nodes ---> good for SGD (Model is easy to train)
Conclusions

• A larger batch size is good for the training of the model (related to convergence).
• When the batch size is large enough, increasing the number of parallel nodes has not had that many benefits.
• When the batch size is quite small, increasing the number of parallel nodes will do good to the training process to some degree. However, this will also increase the overhead of the system.
• Using more parallel nodes can smooth SGD algorithm.
Thanks!
Questions?