Parallel Implementation of Gradient Descent

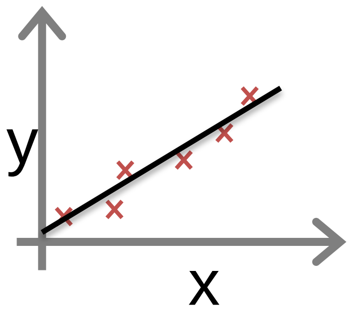
CSE 633: Parallel Algorithms (*2012 Fall*)

Hui Li
Department of Computer Science and Engineering
University at Buffalo, State University of New York

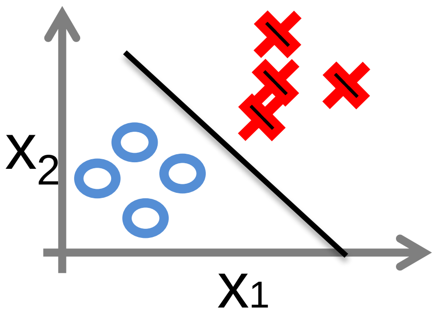
Table of Contents

Background

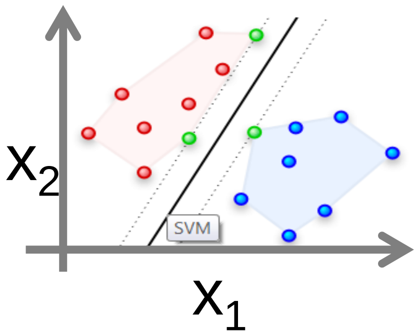
Gradient descent is a general purpose optimization technique which can be applied to optimize some arbitrary cost function J on many prediction and classification algorithms.



Linear Regression



Logistic Regression



SVM

...

Gradient Descent Algorithm

- **Gradient descent update equations**

We want to choose θ so as to minimize cost function J(θ) with learning rate α.

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

This update is simultaneously performed for all values of j = 0, . . . , n

- **Batch gradient descent.**

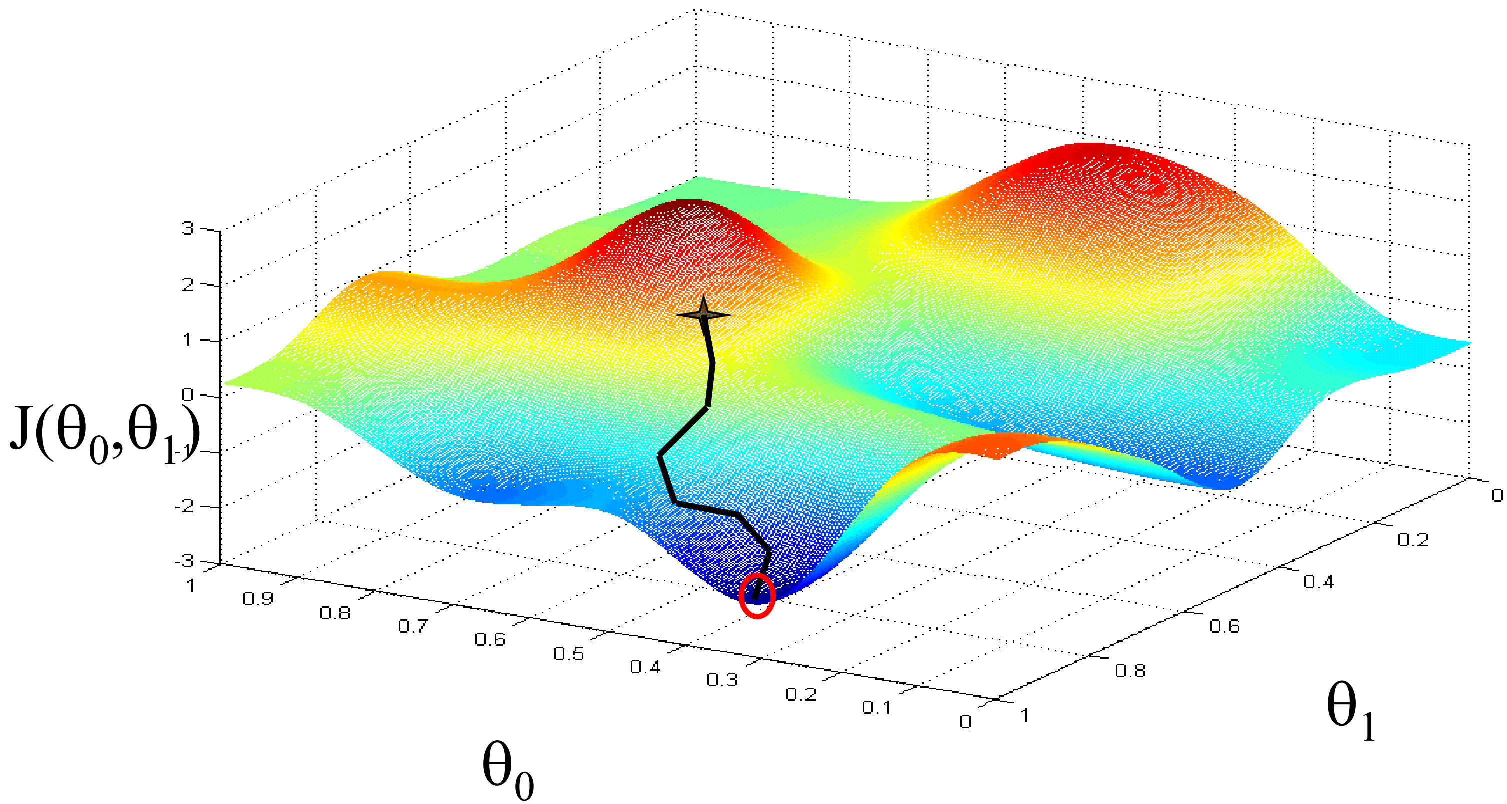
Repeat until convergence {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)} \qquad \text{(for every } j\text{).}$$

}

Here, m is the number of samples**.**

Gradient Descent Illustration



Time Complexity analysis

Basically, for t iteration of a batch gradient descent on m training samples, it requires a time $t \times (T\_1 \times m + T\_2)$. Here, T_1 is the time required to process each sample, and T_2 is the time required to update the parameters.

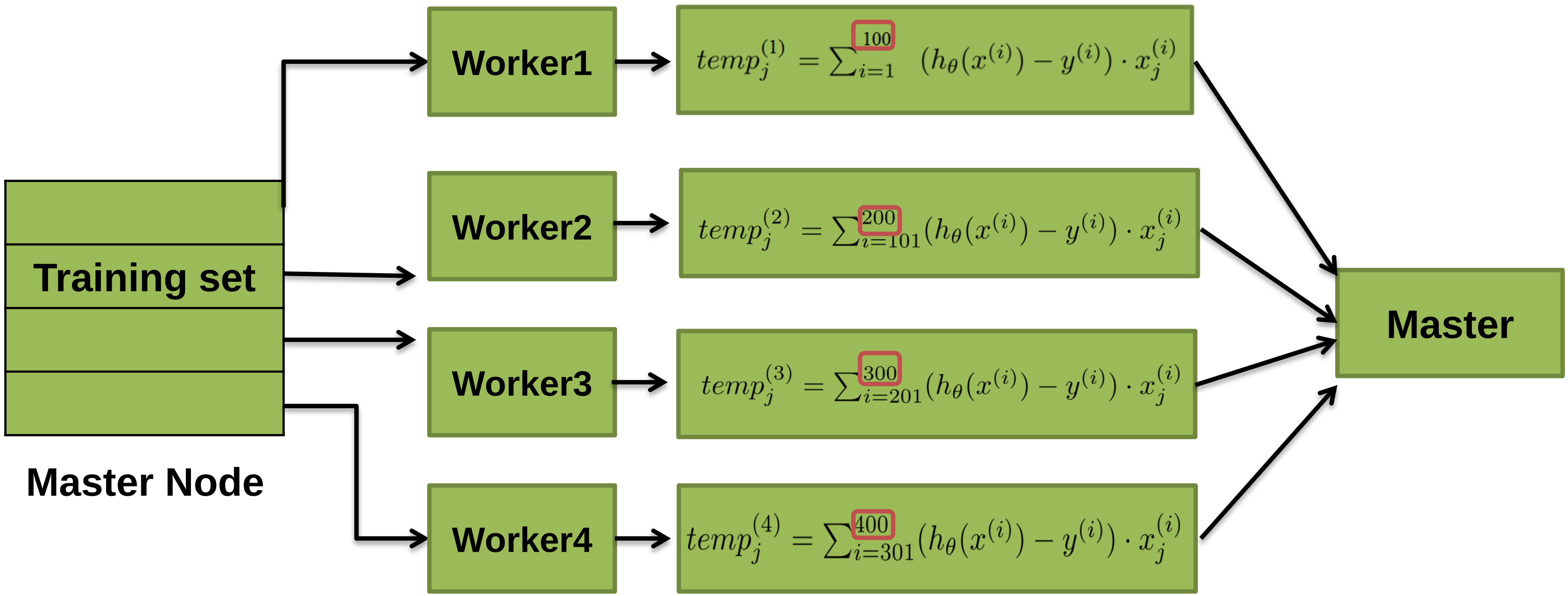
Normally $m >> j$, j is the number of parameter. For example, m would be very large, say 100,000,000. So when m is large, it can be very time consuming!

If we consider optimization problem, the algorithm is more expensive.

We need to parallel batch gradient descent!

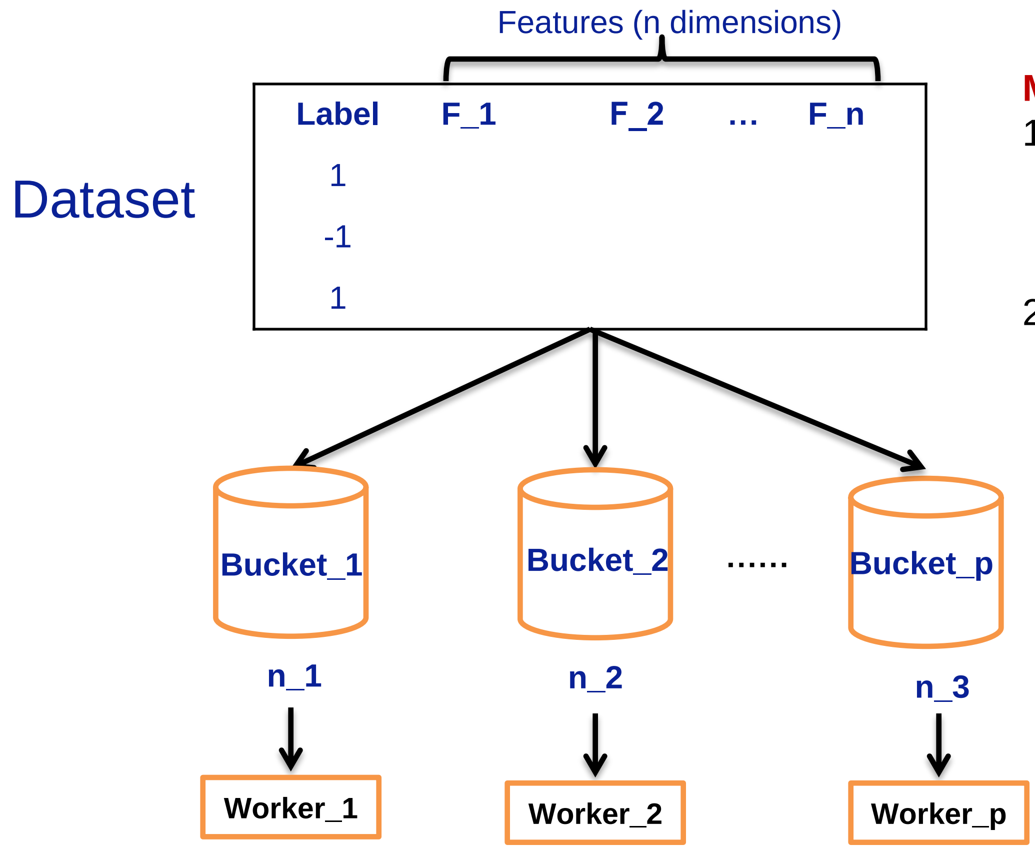
Parallel Scenario

For each iteration (400 samples, for example):



- Each work calculates local gradient
- Send to a centralized master server and put them back together
- Update θ using $\theta_j := \theta_j - \alpha \frac{1}{400}(temp_j^{(1)} + temp_j^{(2)} + temp_j^{(3)} + temp_j^{(4)})$
- Ideally, we can get 4X speed up

Parallel Implementation -- Initialization



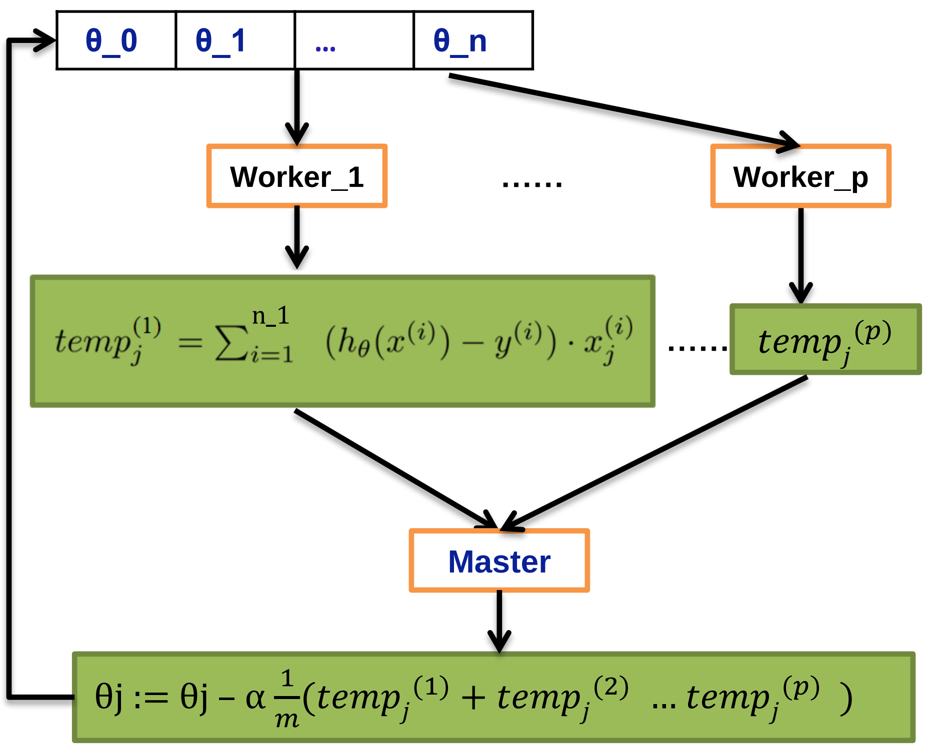
Master Node:

1. Split data to p buckets for workers_1 to worker_p evenly and the last bucket also store the extra samples.

2. Send number of samples to workers such as n_1, n_2, …n_p for initialization

Parallel Implementation -- Update Gradient

Update Weight θj at the Master Node



Master Node:
Send weight to each worker. We initialize weight to 1 at first time

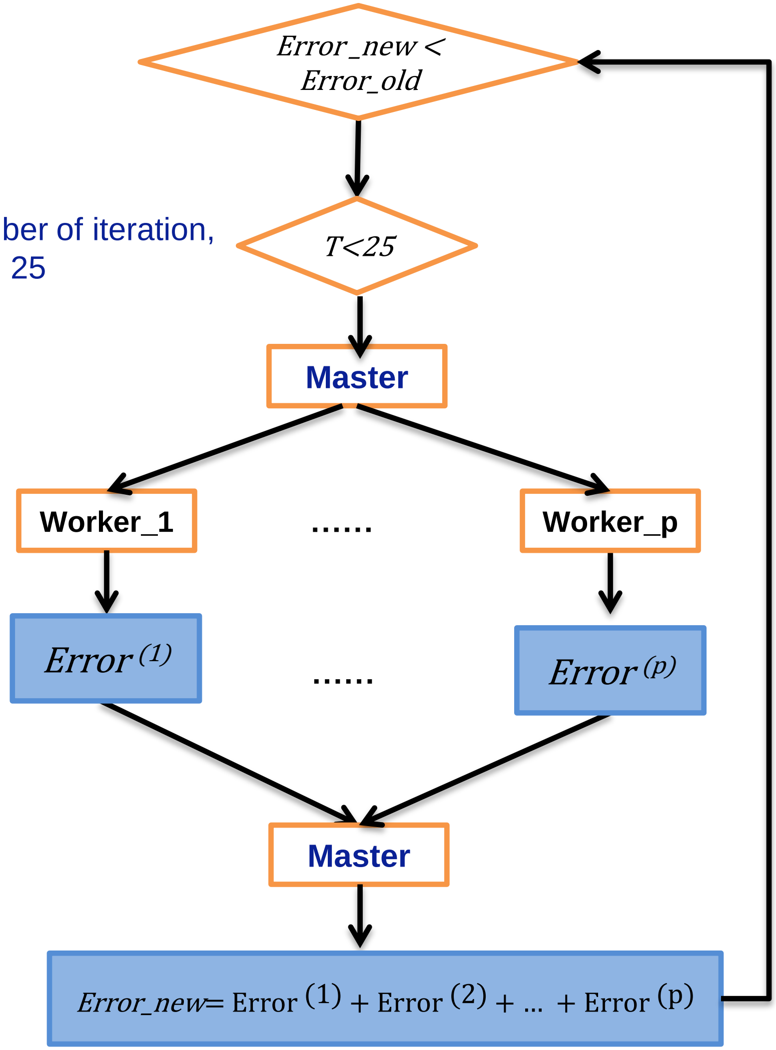
Worker:

1. Receive data from corresponding bucket by id and number of samples sent from Master node
2. Calculate local gradient for each worker, for example, $temp_j{}^{(1)}$ is the gradient for the first worker.
3. Send local gradient to the master code

Master Node:
Sum up local gradient and update weight for θj (j=1 ... n) simualtaniously

Parallel Implementation -- Cost and Termination

T is the number of iteration, for example, 25



Master Node:
If *Error_new* is less than *Error_old*, update *Error_old with Error_new* and repeat program. Actually *Error_old* keep decreasing until finding a minimum. We initialize *Error_old* to a large number. Else, end program.

Worker:
1. Calculate local error which is the number of samples we got wrong for each worker.
2. Send local error to the master code

Master Node:
Sum up local error and compared with the minimum error *Error_old*

Experiment Setup

- Dataset: NHANES -- National Health and Nutrition Examination Survey ($24,000 \times 9999$) contains data of $24,000$ persons ages 2 months and older for disease risk factor analysis.

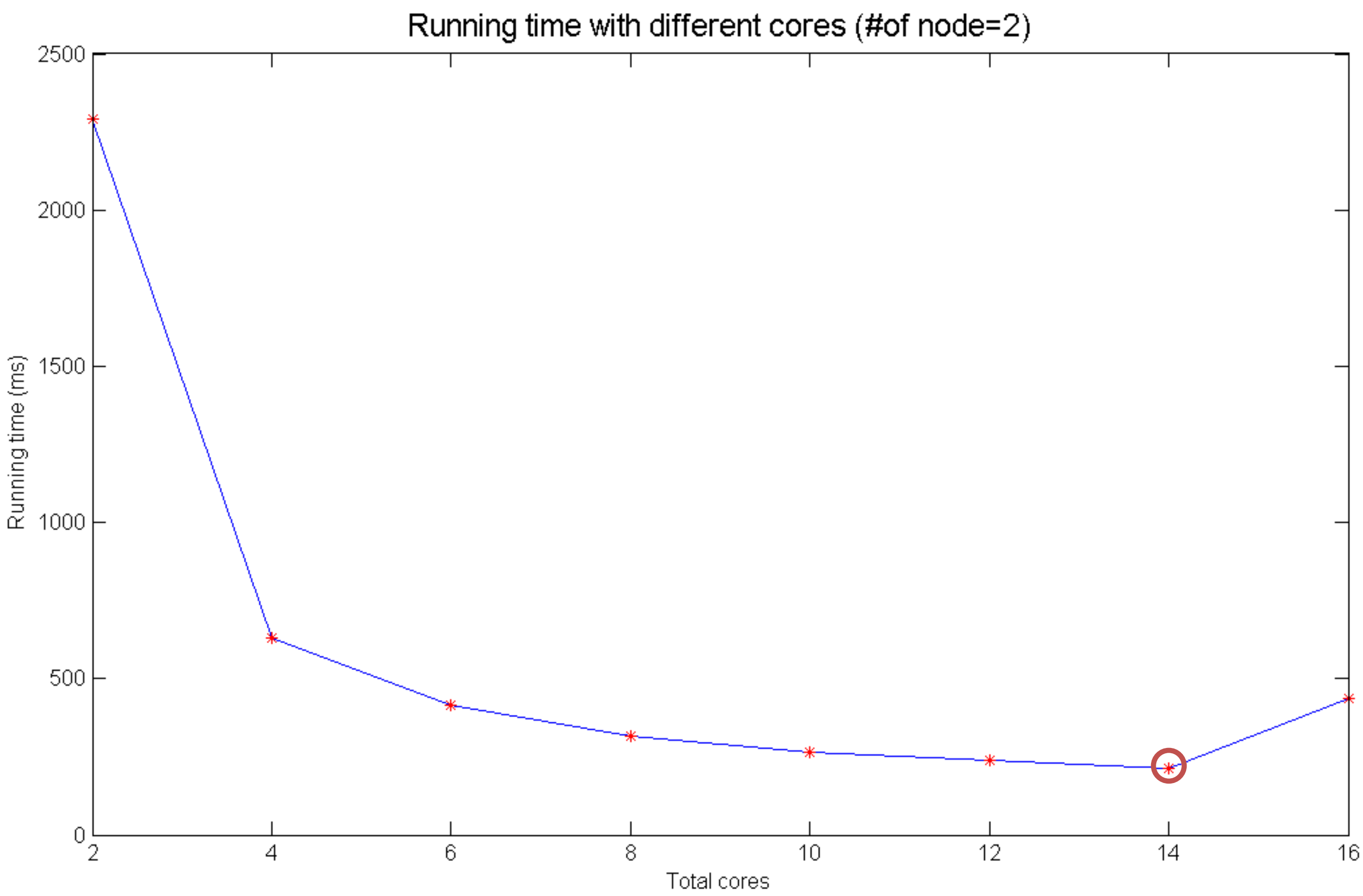
- Master node is in charge of job distribution and collection. Worker do computation.

- Experiment 1: # of node = 2 (fixed)
  # of PPN = 2,3,4,5,6,7, 8

- Experiment 2: # of PPN = 2 (fixed)
  # of node = 1,2,3,4,5,6,7,8

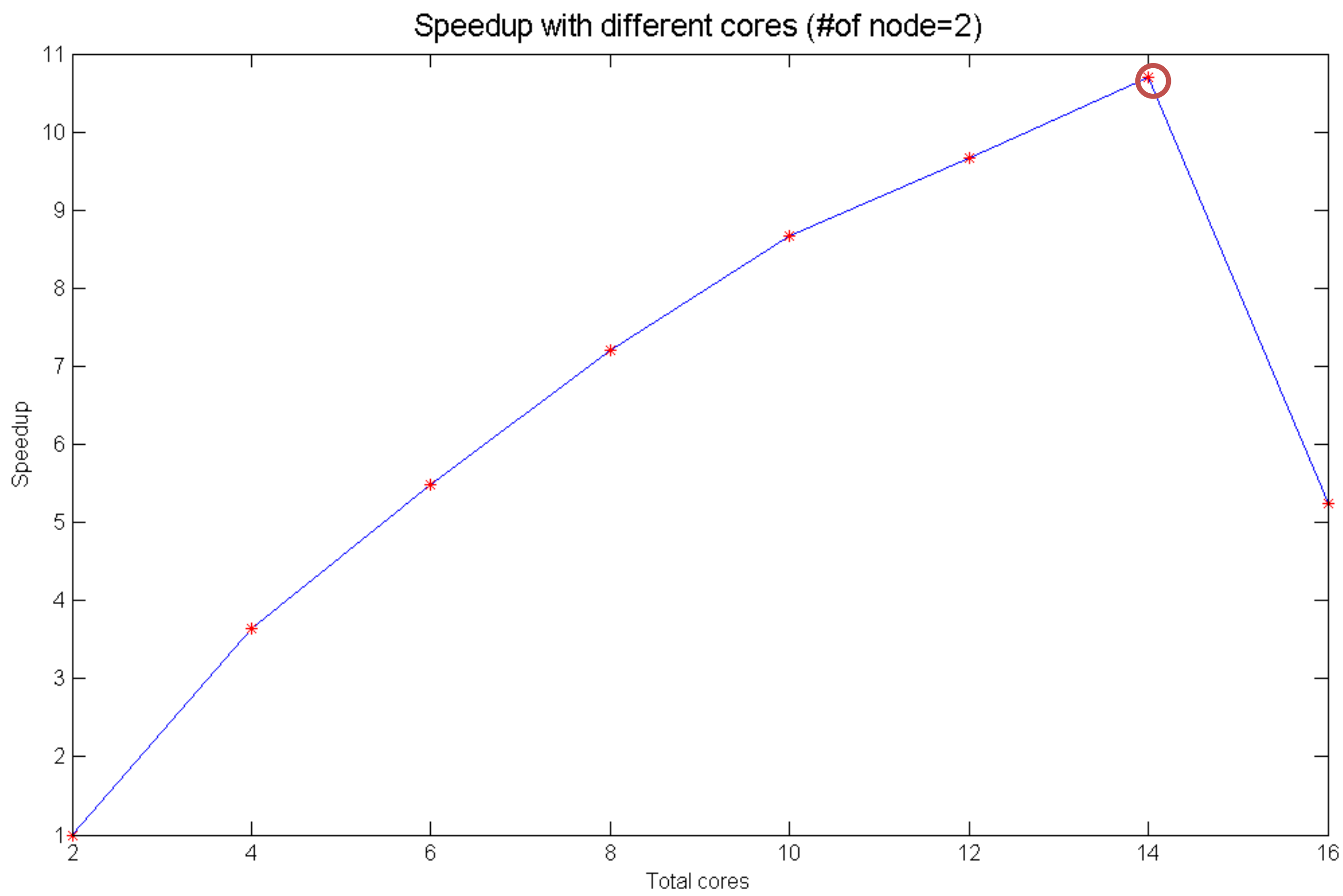
- We use 2,4,6, … 64 cores to set up the experiment and plot performance graph. One core works as master node, is mainly in charge of collecting data. Other cores do computation work.

Experiment Results -- fixing the number of node



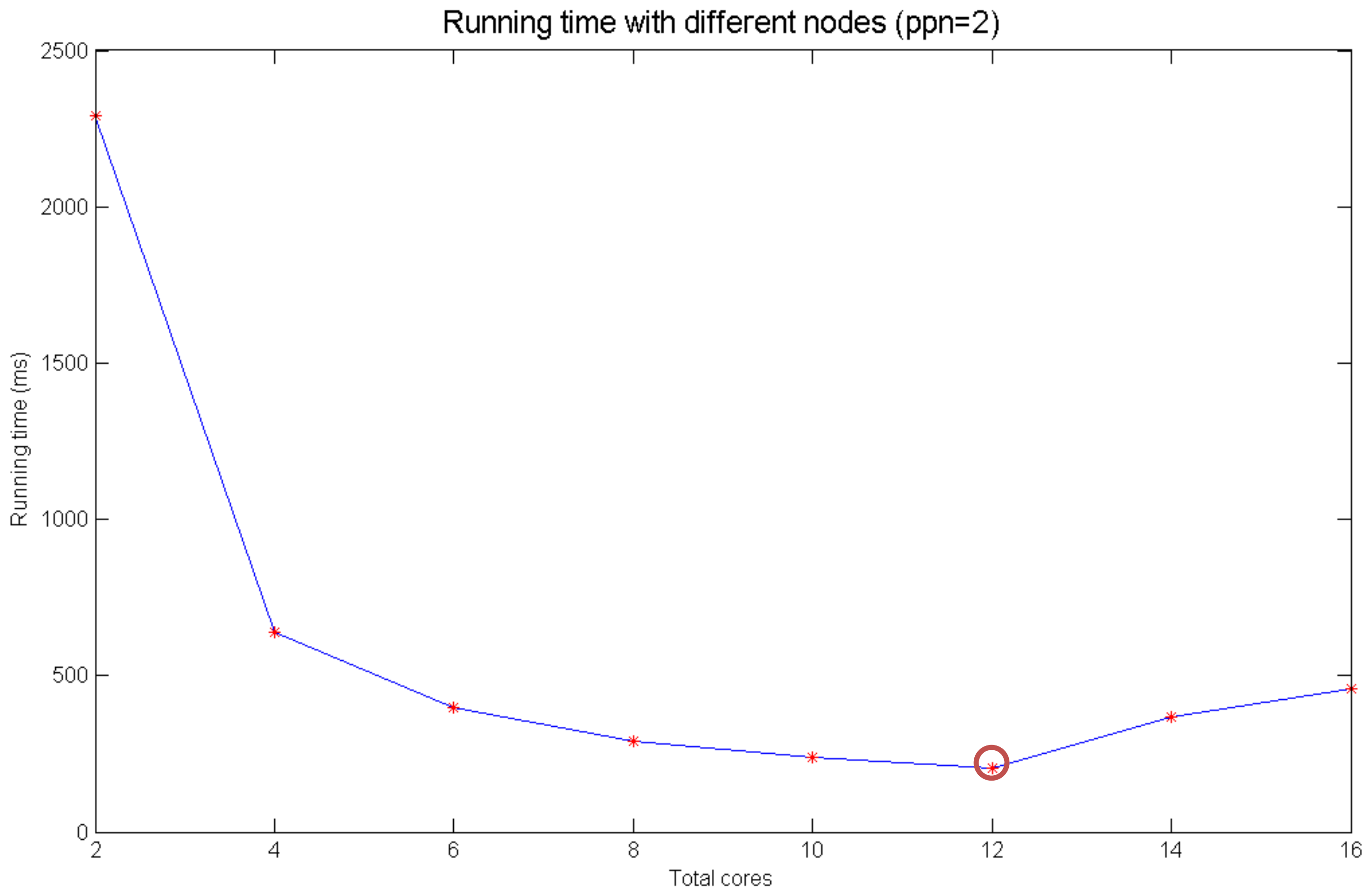
- # of ppn: from 2 to 8
- # of node: 2
- Total cores: (# of node) × (# of ppn)

Experiment Results -- fixing the number of node



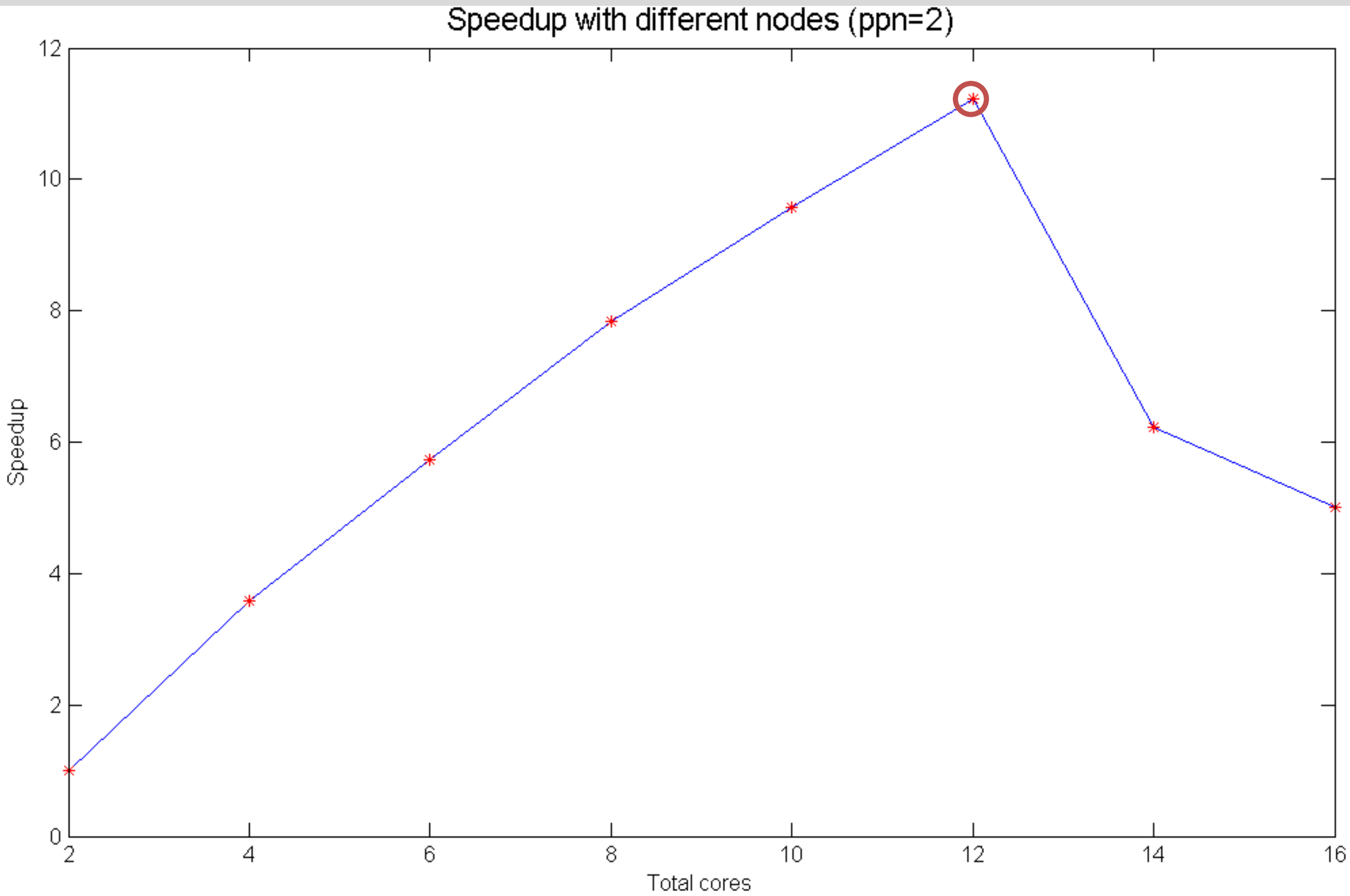
- # of ppn: from 2 to 8
- # of node: 2
- Total cores: (# of node) × (# of ppn)

Experiment Results -- fixing the number of ppn=2



- **# of ppn: 2**
- **# of node: from 1 to 8**
- **Total cores: (# of node) × (# of ppn)**

Experiment Results -- fixing the number of ppn



- # of ppn: 2
- # of node: from 1 to 8
- Total cores: (# of node) × (# of ppn)

Results Analysis

# of node	# of PPN	# of total core	Run time	Speedup
2	2	4	630	3.635
2	3	6	417	5.492
2	4	8	318	7.201
2	5	10	264	8.674
2	6	12	237	9.662
2	7	14	214	10.700
2	8	16	437	5.240

Table1 : Experiment when fixing node number

# of node	# of PPN	# of total core	Run time	Speedup
2	2	4	640	3.578
3	2	6	399	5.739
4	2	8	292	7.842
5	2	10	239	9.582
6	2	12	204	11.225
7	2	14	368	6.223
8	2	16	457	5.011

Table2 :Experiment when fixing PPN number

- The bottom point is T(14)=214ms for the fixed node case. While the bottom point is T(12)=204ms for the fixed PPN case.
- The fixed node running time is slight less than the fixed PPN case.

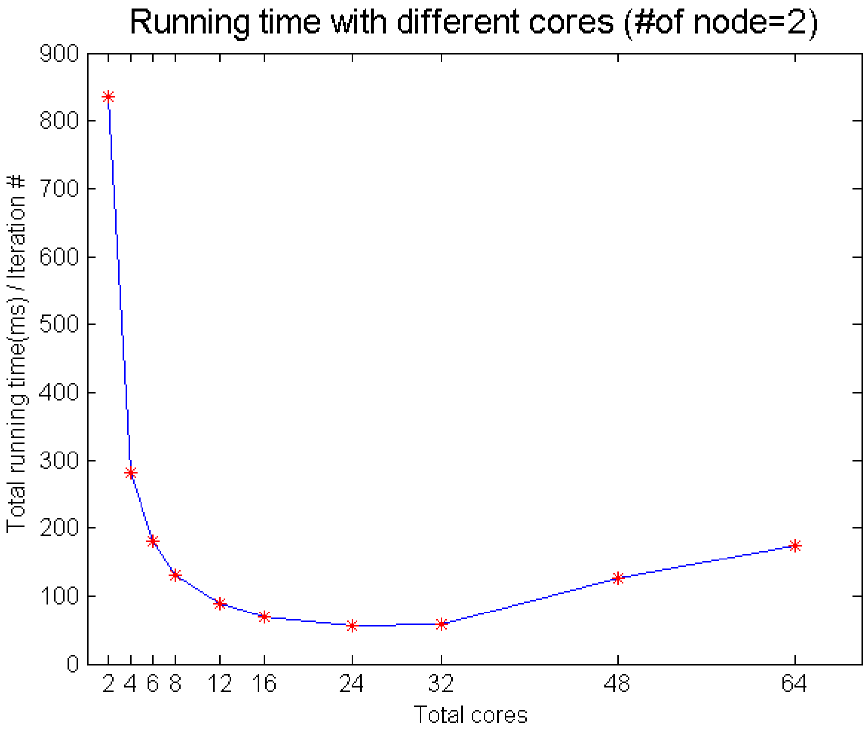
Conclusion: Intra-Node communication performance gives better performance than Inter-Node communication for this dataset (24,000 samples)

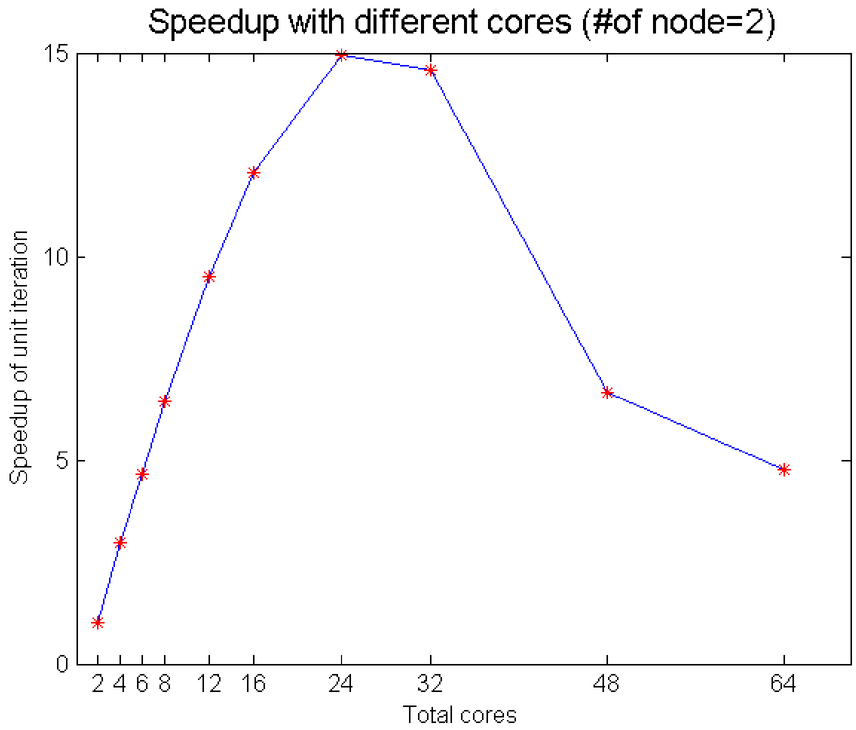
Experiment Result -- Unit iteration

Program converges within different iterations. So to measure our performances, we'd better provide the unit iteration performance.

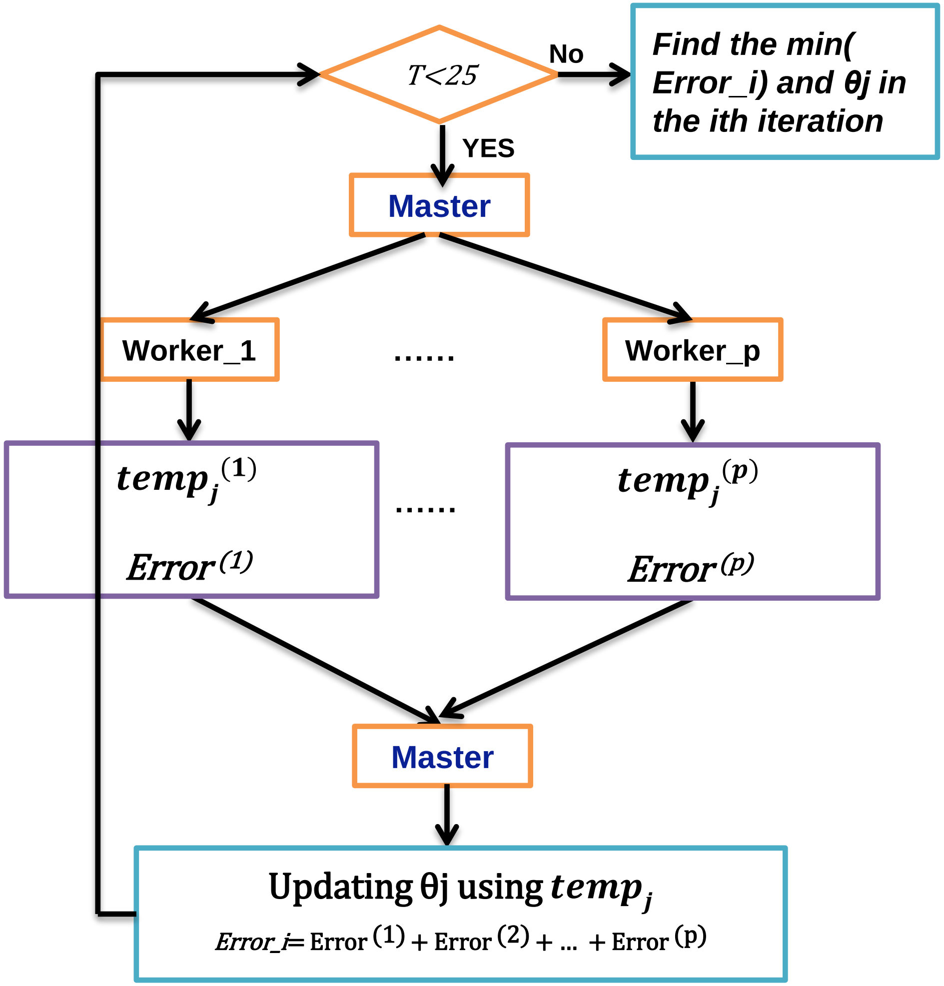
Core #	Iteration #	Running time (ms)
2	4	3344
4	4	1123
6	4	719
8	4	519
12	4	352
16	4	277
24	4	224
32	5	287
48	7	880
64	7	1226

Experiment Result -- Unit iteration





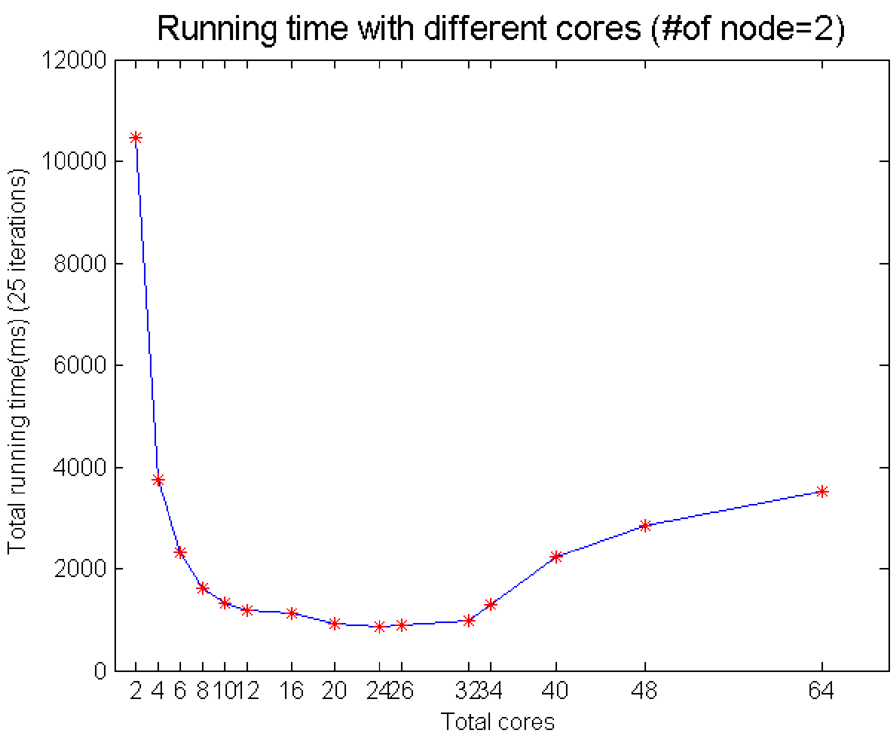
Algorithm Improvement

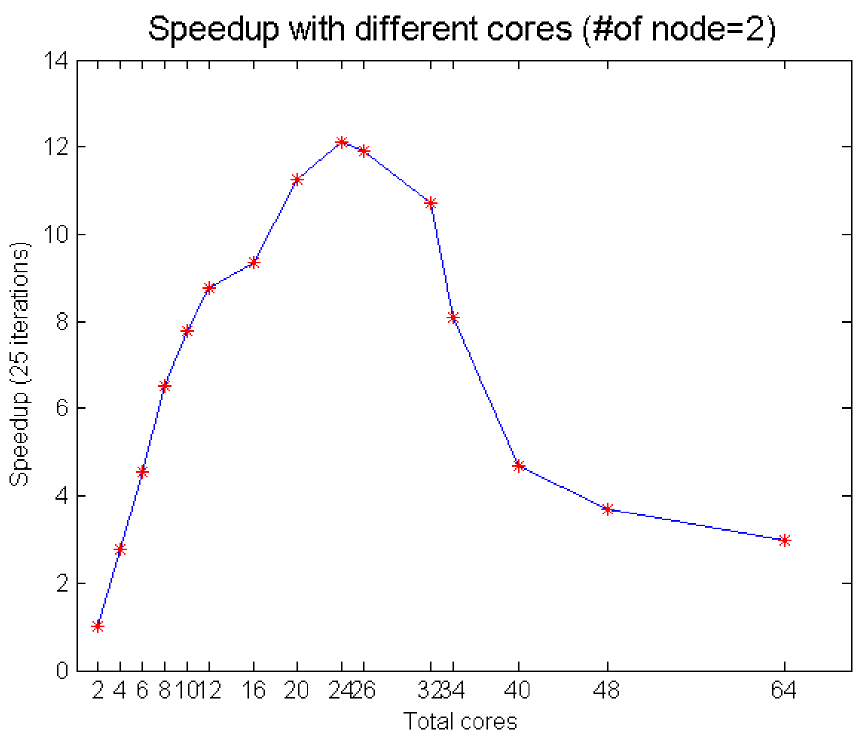


Improvement on Cost and Termination for previous algorithm:

Instead of comparing the new error with the old error, we fix iteration to 25 since program always converges within 10 iterations. In each iteration, the Worker calculates local gradient and local error *Error_i* and the Master node updates the parameter *θj* , saving the global error for the *ith* iteration. After 25 iterations, we choose the parameter which minimizes the global error.

Experiment Results for Algorithm Improvement





Discussion

Questions
&
Answers


Thank You
Thank You
Thank You