Java Parallel Implementations of Kohonen Self-Organizing Feature Maps

YANG Shang-ming¹,  HU Jie²
(1. School of Computer Science and Engineering, UESTC Chengdu 610054 China;
2. Department of Computer Science, Minnesota State University, St. Cloud Minnesota 56301 USA)

Abstract  The Kohonen self-organizing map (SOM) is an important tool to find a mapping from high-dimensional space to low dimensional space. The time a SOM requires increases with the number of neurons. A parallel implementation of the algorithm can make it faster. This paper investigates the most recent parallel algorithms on SOMs. Using Java network programming utilities, improved parallel and distributed system are set up to simulate these algorithms. From the simulations, we conclude that those algorithms form good feature maps.

Key words  SOM; parallel and distributed system; Java network programming

1  The Self-Organizing Map and the Parallel and Distributed Algorithm

1.1  The Self-Organizing Map

The self-organizing map (SOM) is an unsupervised learning neural network. The goal of this network is to produce a similarity graph of input data. It is a two-dimensional mesh of neurons each with a weight vector. Generally, the network organizes the nodes in the grid into local neighborhoods and the learning process acts as feature classifiers on the input data. The feature map is formed automatically through a cyclic process by comparing input patterns to vectors at each node. No training response is specified for input.

Parallel and distributed algorithms are currently an area of intense research and application activity, motivated by a many factors. There has always been necessary for the solution of very large computational problem. Recent technological advances of computer have made some such problems solvable through the parallel and distributed computation.

Among there is the SOM, where training requires much computation. In order to provide an efficient application, parallel and distributed implementations of the SOM are developed, which reduces the training time to a large degree.
where $\alpha(t)$ is the learning rate.

### 1.2 Parallel and Distributed Algorithm

Parallel and distributed algorithms divide a computation into sub-computations which are then executed in parallel on processors. For example, many interesting algorithms for the solution of some systems have the structure

$$x(t + 1) = f(x(t)) \quad t = 0, 1, 2, \ldots$$

(4)

where $x(t)$ is an $n$ dimensional vector, and $f$ is the function from $\mathbb{R}^n$ into itself. They are called iterative algorithms, if

$$x_i(t + 1) = f_i(x_i(t), \ldots, x_n(t)) \quad i = 1, 2, \ldots, n$$

(5)

the iterative algorithm $x(t + 1) = f(x(t))$ can be parallelized by letting each one of $n$ processors update a different component of $x$ according to the equation at each round. During the computation, the $i$th processor gets the value of all components of $x_i(t)$ from other processors, computes the new value $x_i(t + 1)$, and communicates it to other processors for the next iteration.

### 2 The Block Algorithm

The block algorithm is a parallel algorithm for the self-organizing map. In the block algorithm, every processor processes a certain number of neurons, called a block, it determines a local winner from the processed neurons and then one processor determines the global winner for all the neurons. During the update phase, only the winning neurons and their neighborhoods are active.

#### 2.1 The Neighborhood and the Input for the Block Algorithm

To reduce the training time, the strategy is one processor processes several neurons. But giving size of map and neighborhood, we have to keep one neuron can be only in one neighborhood. If a map is a square of size $n$, the neighborhood is a square of size $r < n$. We put $m$ neighborhood’s squares on the map. To choose different squares in this arrangement, the lower and upper bound probability of no intersection is as follows

$$e^{-\pi(m-1)r^2/2n^2} < p(m) < e^{-(m-1)r^2/2n^2}$$

(7)

where $p(m)$ is the probability that two squares do not overlap.

Using the probabilities, it is possible to present more than one sample at the same time to the map. Then each neuron can modify its weights according to the input. If a neuron is selected by more than one input, as Kohonen maps preserve the neighborhood properties of the input space, the neural system modifies multiply selected neurons with the closest sample to it on the map.

#### 2.2 New Implementation for the Block Algorithm

SIMD computers is composed of one instruction fetch unit and multiple data processors. Originally, the block algorithm is implemented on a parallel system called MasPar. A MasPar parallel processing system consists of a front end and a data parallel unit: a SIMD system. Using Java networking utilities, we can build up an environment that is similar to the MasPar parallel system to implement this algorithm.

In this paper, all simulations use the same group of Animal Data Set in Tab.1. The animal data set was originally introduced by Ritter and Kohonen to illustrate the SOM for high-dimensional data set. It consists of the 16 animals represented by binary property lists. These 16 animals are grouped into three classes; class one represents bird, class two represents carnivore, and class three represents herbivore.

In this implementation, five computers are connected to form a parallel system using Java client-server model. There are four servers and one client: each server takes care of four cells. The client works as the control unit, which will compute the global winner and pass messages to each server. The servers work as processors to update the neurons in the neural network system.

For each sample, system needs to determine the neuron that is the closest to the input. This involves communications among the processors to determine the global minimum distance. The main problem is to distribute the SOM on the processors to determine the global winner quickly. In this system, communications
are necessary to identify the global winner. Processors pass the local winners to the control unit, which determines the global winners. Then it determines the neighborhoods and passes them to processors. Processors modify the weights. For \( N \) samples, comparing with the classical algorithm for block size one, this allows to decrease communication time \( t_{\text{comm}} \) from \( N^{2D}(\alpha+\beta L) \) to \( 2^{D}(\alpha+\beta N L) \), where \( \alpha \) is a fixed startup overhead, \( \beta \) is the incremental transmission time per byte, \( D \) is the dimensions of hypercube in the system, and \( L \) is the length of each message.

The neighborhood size decreases from 3 to 0. In the last 10 epochs, only the winner is updated in each epoch. The simulation steps for block algorithm are:

1) Initialize the weights for all 16 neural cells.
2) Present two input samples to each processor. Each processor computes the local winner and passes it to the control unit.
3) Control unit receives local winners from each processor and computes the global winner and its neighbors.
4) Control unit broadcasts the results in step 4 back to each processor.
5) After received message from control unit, winning cells and their neighbors update their weights. Non-winning cells return to step 2.
6) Winning cells finish update neurons and return to step 2.

The input vectors are the rows of Tab.1. The initial neural weight is a random 4x4 vector matrix

\[
\begin{pmatrix}
\mu_{11} & \mu_{12} & \mu_{13} & \mu_{14} \\
\mu_{21} & \mu_{22} & \mu_{23} & \mu_{24} \\
\mu_{31} & \mu_{32} & \mu_{33} & \mu_{34} \\
\mu_{41} & \mu_{42} & \mu_{43} & \mu_{44}
\end{pmatrix}
\]

After the input data are presented one by one, the calibrated feature maps of the block algorithm for the animal data set are shown in Fig.1. In the figure, the number 1, 2 and 3 represent the feature map of the three classes of animals after the updates.

![The distribution of data and the calibrated maps constructed by block algorithm](image-url)

<table>
<thead>
<tr>
<th>Medium</th>
<th>Big</th>
<th>Two legs</th>
<th>Four legs</th>
<th>Hair</th>
<th>Hooves</th>
<th>Mane</th>
<th>Feathers</th>
<th>Hunt</th>
<th>Run</th>
<th>Fly</th>
<th>Swim</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dove</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Hen</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Duck</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Goose</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Owe</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Hawk</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Eagle</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Fox</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Dog</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Wolf</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Cat</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Tiger</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Lion</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Horse</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Zebra</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Cow</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

Tab.1 The animal data set
From Fig.1, it’s clear that after 50 epochs of updates of the block algorithm, the same group data are neighbors. From epoch 10 to epoch 50, there are more and more first and second classes appeared on the map.

3 The Batch Algorithm

The batch algorithm presents more than one input to the neural system in each epoch to increase the computation speed. There will be one more winners and there are at least two neurons updated in each round of the learning process.

3.1 Definition of the Batch Algorithm

The Batch Algorithm is defined by batch winner selection, which means the weight vectors are updated only after the presentation of a group of \( T \) inputs. \( T \) is called epoch length. The time is now as \( t=kt+\tau \) \( k=0, 1, 2, 3, \ldots \) is the epoch number and \( 0 \leq \tau \leq T-1 \) is the iteration step running in an epoch. The winner \( w(kt+\tau) \) is selected by comparing the input \( \xi(kt+\tau) \) with the weight vectors \( \mu_i(kt) \) at the beginning of the \( k \)th epoch.

\[
w(kt+\tau) = \min_i d(\xi(kt+\tau), \mu_i(kt))
\]

and a batch weight update, since the modification of the weight is computed from the weight vector \( \mu_i(kt) \) at the beginning of the \( k \)th epoch

\[
\mu_i(kt+\tau+1) = \mu_i(kt+\tau) + \alpha(kt)\bullet(\xi(kt+\tau) - \mu_i(kt))
\]

where \( \tau=0, 1, 2, \cdots, T-1 \) and \( \mu_i(t) \) is the batch winner or it’s neighborhood. By taking \( T=1 \), Eq.(9) becomes the original SOM algorithm.

3.2 New Networking Implementation for Batch Algorithm

In this system implementation, five computers are pipelined, four of them are called processors, and one (instruction unit) is used for broadcasting. Using the idea of block algorithm to reduce the communication time, each processor will process 4 neurons. The instruction unit will pass instructions to each processor. Two input models are chosen for each epoch. The instruction unit determines the batch winners for the epoch and determines the neighbors for each winner. After receiving this information from the instruction unit, processors will update these batch winning cells and their neighbors at the same time. The convergence condition for \( \alpha(t) \) is \( 0<\alpha(kt)<1 \) for any \( k \). To be sure the modification is convergent, it needs to update \( \alpha(t) \) very often. The networking implementation steps are:

1) Initialize weight for each cell in processors.
2) Present batch (2 inputs) input vectors to each processor.
3) Processors compute batch distances and sends them to the instruction unit.
4) Instruction unit decides batch winners by comparing all the distances and then computes their neighbors.
5) Instruction unit broadcasts the winners and their neighborhood information to each processor.
6) After receiving information from instruction unit, batch winners and their neighbors update their weights and non-winners return to step 2).
7) Winning cells and their neighbors return to step 2).

The update results for different epochs are in Fig.2. After 50 epoch updates, class two and class three are grouped, but class one doesn’t come out. It needs more epochs to get feature map. For comparing the three algorithms, we only set up same number of epochs here.

\[
\begin{array}{cccc}
333 & 3 & 3 & 3 \\
333 & 3 & 3 & 3 \\
333 & 3 & 3 & 3 \\
333 & 3 & 3 & 3 \\
333 & 3 & 3 & 3 \\
\end{array}
\]

(10 epochs) (20 epochs) (30 epochs) (40 epochs) (50 epochs)

Fig.2 The distribution of data and the calibrated maps constructed by batch algorithm

4 The Asynchronous Stochastic Algorithm

The asynchronous algorithm may further increase the speed of forming feature maps. In fact, all the processors in the system update their neurons without
waiting for other neuron’s update results. In this part, the energy function of the SOM is introduced. Using this measure, the asynchronous algorithm of SOM is discussed, and the asynchronous system is simulated by using Java network utilities.

4.1 Definition of the Asynchronous Algorithm

For the synchronous approaches such as the Block algorithm and the batch algorithm, all the weight vectors in the networks are updated during the time step \( t \) of the algorithms. In the Asynchronous Algorithm, however, updates of cell weight vectors occur over a set of time steps with a bound on the number of steps required to deal with any given observation. In the probabilistic algorithm, the energy function of SOM is defined as [6]

\[
E = \sum_{i=1}^{M} \left[ \log \sum_{n=1}^{N} g_{ij} p(x_i)p_j(x_i) \right]
\]

where \( g_{ij} = 1 \) when neurons \( i \) and \( j \) are neighbors and 0 otherwise, the input data set is \( \{x_1, x_2, \ldots, x_M\} \), and

\[
p_j(x_i) = p(x_i, \mu_j) = \frac{1}{\sqrt{2\pi} \sigma_j} \exp\left(-\frac{\|x_i - \mu_j\|^2}{\sigma_j^2}\right)
\]

The objective of this algorithm is to maximize the correlation between neighboring neuron outputs. The network parameters (\( \mu \)) is a weight vector) should maximize the cost function Eq.(11).

Assume that there is a set of times \( T = \{0,1,2,\ldots\} \) at which one or more components \( \mu \) of \( \mu \) are updated by some processors of a distributed system. If there are \( N \) processors, each processor is associated at each time \( t \) with a vector \( \mu(t) \in \mathbb{R}^n \). Let \( T' \) be equal to set of times at which \( \mu \) is updated. The algorithm requires that for every \( i \) and for every \( t > 0 \), at least one of the elements of the set \( \{t, t+1, \ldots, t+B\} \) belongs to \( T' \). It further requires that information used by a processor be updated by at most \( B \) time units. \( B \) is called the asynchronous measure and the algorithm is termed partially asynchronous with those restrictions. For the asynchronous algorithm, the processor updating the vector component \( \mu \) of length \( n \) may not have access to the most recent values of the \( n \) components of \( \mu \) that other processors have generated. Consequently, the update calculation becomes

\[
u_i(t+1) = u_i(t) + \alpha(t)s_i(t)
\]

where \( \alpha(t) \) is a positive step size and \( s_i(t) \) is the update direction

\[
s_i(t) = \sum_{j=1}^{N} w_{ij}[x_{ij(t)} - u_i(t)]
\]

\[
w_{ij} = \frac{g_{ij} p(x_{ij(t)}, u_i(t))p(x_{ij(t)}, u_i(\sigma_i(t)))}{\sum_{n=0}^{b} g_{in} p(x_{in(t)}, u_i(\sigma_i(t)))p(x_{in(t)}, u_n(\sigma_n(t)))}
\]

where \( \sigma_i(t) \) are times satisfying

\[t - B \leq \sigma_i(t) \leq t \quad \forall t \in T'
\]

and \( \sigma_i(t) \) is \( t \) for all \( i \) and \( t \in T' \), at all times \( t \in T' \), \( \mu \) is left unchanged, giving

\[s_i(t) = 0 \quad \forall t \in T'
\]

At any given time \( t \in T' \), when an update is computed for processor \( i \), the \( a(t) \) is as a function of the sample number for the corresponding \( x = x_{i(t)} \) used in the update where samples taken from \( \{x_1, x_2, \ldots, x_M\} \). Associate with processor \( i \) at time \( t \) a vector \( \mu(t) \) with \( n \) element vector components that need not be the most recent ones in the processors where they were determined. This is denoted as

\[
\mu(t) = (\mu_{\sigma_i(t)}, \mu_{\sigma_2(t)}, \ldots, \mu_{\sigma_n(t)})
\]

where each \( \mu_{\sigma_i(t)} \) is an \( n \)-element vector. The input data \( x_{i(t)} \) are randomly selected and the resulting algorithm is an asynchronous stochastic gradient algorithm.

5.2 New System Implementations

Similar to the block and batch algorithm, Five computers are used to build up a network system for this algorithm. Computers used to update neurons are called processors. Using the block algorithm again to reduce communication time, each processor computes four neurons. The one for communication is called the central unit. The unit is used to get the updated neural weights from each processor and then broadcasts all the updated neurons back to each processor for the next update. Since communication may be delayed in the asynchronous algorithm, we also
need to use a queue to store a finite number of weights for these neighbors. The communication of processors for the neural weights exchange will take some time in this system since each processor needs to exchange its updated neurons with the others. In this point, we need to find out an approach so that the communication will take least time in this network system. For the same data model as in the block algorithm, it takes more than four times long to finish the simulation. We have the following basic simulation steps for asynchronous algorithm.

1) Initialize weight for each cell.
2) Each cell computes their neighbor matrix, for each element $g_{ij}$ in the matrix, it is 1 if $i$th and $j$th neurons are neighbors and 0 otherwise.
3) Present input model to each cell one by one.
4) Each cell update their weights using their neighbor weights received in the queue
   \[ \mu(t+1) = \mu(t) + \alpha(t) \ast s(t) \]
5) Each processor passes the updated weights to the central unit.
6) After getting all the updated neurons, the central unit broadcasts them back to each processor for next updates.
7) Each processor receives the global updates weights and save them in the queue.
8) Return to step 2).

Each algorithm has different features. Fig.4 shows the comparison of the three implementations in epoch and time in Unix network system. The asynchronous algorithm takes about 1400 s and the block algorithm takes about 320 s and the batch algorithm takes about 170 s for 50 epoch updates. The asynchronous algorithm takes the longest time because in each epoch, the system needs to broadcast all the updated neuron weights (16x13 decimal numbers for each epoch) to each processor. In block and batch, the only communication is to send and get back the winner and the neighborhood information from broadcasting unit. The batch algorithm takes shortest time to finish 50 epochs of update, but it needs more epochs to get the feature map. For the block and asynchronous algorithms, although they take longer in communication, the data are grouped very quickly.

5 Conclusions

Parallel algorithms for Kohonen self-organizing maps can be implemented in Java on networking system. These implementations demonstrate how the SOM algorithms efficiently generate topologically ordered feature maps.

The block algorithm proposes a new parallel method for the SOM using a block strategy. The network implementation shows that the block strategy performs very well. This idea is used in the implementation of the batch and asynchronous
algorithms in the new implementations. The simulation of the batch algorithm confirms the theoretical results. In the network implementation, the new algorithm includes the block strategy. The batch algorithm updates overcome the problem when the neighborhood becomes smaller; most processors are not active since they are not the winner. In this implementation, at least two more processors are involved in the computation in each epoch. Meanwhile, the winner is selected in batch, and it also speeds up the data process.

Using of energy function leads to an asynchronous parallel stochastic gradient method for self-organizing map. The simulation demonstrates the utility of asynchronous approaches to this algorithm. We can also see the practical efficiencies involving sampling for the approximate calculation of an expensive summation in the algorithm. The new implementation for this algorithm also includes the block strategy. For the asynchronous implementation, it forms the feature map using very few epochs, but the communication in this implementation needs to improve further since it takes too long to finish the same epoch updates.

References


Brief Introduction to Author(s)

YANG Shang-ming (杨尚明) was born in 1962. He is currently a lecturer and Ph.D. candidate in School of Computer Science and Engineering, UESTC. He received his B.Cs and M.Sc degree from Sichuan Normal University, Chengdu, China in 1984 and Minnesota State University, St. Cloud, Minnesota, USA in 2000 respectively. His research interests include: the theory and applications of SOM, data mining, and independent component analysis.

HU Jie (胡洁) was born in 1962. She received her B.Sc degree and M.Sc degree from Xi’an Jiaotong University, China in 1983 and 1989 respectively, and she received Ph.D. degree in computer science from Tohoku University, Japan in 1996. She is currently working as and Assistant Professor in the Department of Computer Science at Minnesota State University, St. Cloud, Minnesota, USA. Her research interests include: self-organizing maps, computer security and network computing.