計算機アーキテクチャ 147－1
ハイパフォーマンス
コンピューティング 89－1
（２００２．３．７）

非一様な並列計算機環境におけるニューラルネットワークの学習

東 大亮　菅谷至寛　阿曽弘具
東北大学大学院工学研究科

階層フィードフォワード型ニューラルネットワークの学習法である誤差逆伝播 (Back Propagation: BP) 学習法を、ワークステーションクラスタ上に効率的に実装する手法を提案する。BP アルゴリズムの並列実装では、2 種類の並列化法 (データ並列およびノード並列) を組み合わせることが一般的であるが、この 2 つの並列化法の並列度の組み合わせと問題の規模によって性能が変化することが知られている。本手法では、アルゴリズム中で実行時間が変化し得る部分をモデル化し、それを用いて最も計算時間が短くなるように並列度を自動的に選択する。さらに、各プロセッサの速度に応じて学習サンプルを分配することで、非一様環境にも対応した負荷分散を行なう。

Training Neural Network on Heterogeneous Parallel Computer

Daisuke HIGASHI, Yoshihiro SUGAYA and Hirotomo ASO
Graduate School of Engineering, Tohoku University.

We propose a effecient method to map back propagation (BP) training algolithm of multilayer feed forward neural network onto workstation cluster. Most of existing mapping combines two kind of parallelism in BP algolithm (data parallelism and node parallelism) to exploit high performance but it is known that performance varies according to degree of each paralellism and size of problem. Our method selects optimal degree of each parallelism to minimize training time. Also our method provides good load balancing for heterogeneous parellel computer assigning training samples to each processor according to speed of them.

1 はじめに

階層フィードフォワード型ニューラルネットワークの学習法である誤差逆伝播 (以下 BP と略す) 法は代表的な教師あり学習法で、工学的に最も多く適用されている学習アルゴリズムの 1 つであるが、多数の結合重みを逐次的に更新していく手法のため計算量が膨大である。そのため計算時間の短縮のために様々な並列計算機上に BP アルゴリズムが実装されてきた [1][2]。BP アルゴリズムを並列計算機上に実装し高い性能を得るには、BP アルゴリズムに内在する 2 種類の並列性 (データ並列性およびノード並列性) を組み合わせることが必要である [1]。しかし、データ並列とノード並列の割合によって性能が変化することが知られており、並列度の組み合わせを自動的に選択する手法が求められる。本研究では、いくつかの WS がイーサネットスイッチで結合されているという構成の単純な WS クラスタを対象とするが、各プロセッサ (WS) の性能が一様ではない場合も考慮する。本手法では、各 WS の能力に応じてそれぞれが担当する学習サンプルを割り当てる。

2 誤差逆伝播学習法

2.1 3 層パーセプトロン

本研究で対象とするニューラルネットワークは図 1 のような 3 層パーセプトロンである。これは、n 個のユニットからなる入力層 $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$、$m$ 個のユニットからなる中間層 $\boldsymbol{f} = (f_1, f_2, \ldots, f_m)^T$、$l$ 個のユニットからなる出力層 $\boldsymbol{h} = (h_1, h_2, \ldots, h_l)^T$ が、重み行列 $\boldsymbol{W} = [w_{ji}](i = 0, 1, \ldots, n,$ $j = 1, 2, \ldots, m)$ および $\boldsymbol{V} = [w_{kj}](j = 0, 1, \ldots, m,$ $k = 1, 2, \ldots, l)$ により、

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{\sigma W x} \quad (1)$$

$$\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{\sigma V f}(\boldsymbol{x}) \quad (2)$$

という形で接続されているという構造のニューラルネットワークである。なお、$\boldsymbol{\sigma}$ はベクトルの各要素に非線型関数 $\sigma(s)$ をかける演算であるが、ここではアナログニューロンモデルとしてよく用いられるシグモイド関数 $\sigma(s) = \frac{1}{1+e^{-s}}$ を採用する。このネットワークに入力 $\boldsymbol{x}$ を与えたときの $\boldsymbol{h}(\boldsymbol{x})$ がネットワー

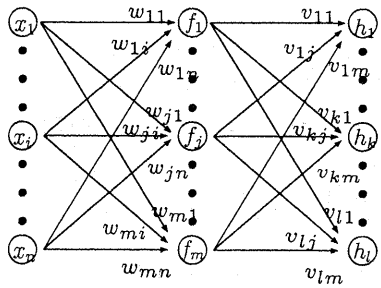

図 1: 3 層パーセプトロン

クの出力であり、これは結合重み行列 $\boldsymbol{W}, \boldsymbol{V}$ によって決定される。

2.2 誤差逆伝播学習法

ある入力 $\boldsymbol{x} \in X$ (X:学習サンプル集合) に対して出力 $\boldsymbol{h}(\boldsymbol{x})$ が、所望の値 (教師信号ともいう) $\boldsymbol{d}(\boldsymbol{x})$ になるように重み $\boldsymbol{V}, \boldsymbol{W}$ を逐次的に更新していく手法の 1 つが BP 法である [3]。BP 法では、全学習サンプル $\boldsymbol{x} \in X$ について、**重み更新量** $\Delta \boldsymbol{V}(\boldsymbol{x}), \Delta \boldsymbol{W}(\boldsymbol{x})$ を求め、式 (3),(4) のように重み $\boldsymbol{V}, \boldsymbol{W}$ を更新する。

$$\boldsymbol{V}^{new} := \boldsymbol{V}^{old} - \epsilon \sum_{\boldsymbol{x} \in X} \Delta \boldsymbol{V}(\boldsymbol{x}) \quad (3)$$

$$\boldsymbol{W}^{new} := \boldsymbol{W}^{old} - \epsilon \sum_{\boldsymbol{x} \in X} \Delta \boldsymbol{W}(\boldsymbol{x}) \quad (4)$$

ϵ は学習定数と呼ばれる微小定数である。
$\Delta \boldsymbol{V}(\boldsymbol{x}), \Delta \boldsymbol{W}(\boldsymbol{x})$ は、

$$\Delta \boldsymbol{V}(\boldsymbol{x}) = \boldsymbol{\delta_2}(\boldsymbol{x}) \boldsymbol{f}(\boldsymbol{x})^T \quad (5)$$

$$\Delta \boldsymbol{W}(\boldsymbol{x}) = \boldsymbol{\delta_1}(\boldsymbol{x}) \boldsymbol{x}^T \quad (6)$$

であるが、ここで $\boldsymbol{\delta_2}(\boldsymbol{x})$ と $\boldsymbol{\delta_1}(\boldsymbol{x})$ は修正誤差と呼ばれ、次のようにして求める。

$$\boldsymbol{\delta_2}(\boldsymbol{x}) = \begin{pmatrix} (h_1(\boldsymbol{x}) - d_1(\boldsymbol{x}))h_1(\boldsymbol{x})(1 - h_1(\boldsymbol{x})) \\ (h_2(\boldsymbol{x}) - d_2(\boldsymbol{x}))h_2(\boldsymbol{x})(1 - h_2(\boldsymbol{x})) \\ \vdots \\ (h_l(\boldsymbol{x}) - d_l(\boldsymbol{x}))h_l(\boldsymbol{x})(1 - h_l(\boldsymbol{x})) \end{pmatrix} \quad (7)$$

$$\boldsymbol{\delta_1}(\boldsymbol{x}) = \begin{pmatrix} f_1(\boldsymbol{x})(1 - f_1(\boldsymbol{x})) \\ f_2(\boldsymbol{x})(1 - f_2(\boldsymbol{x})) \\ \vdots \\ f_m(\boldsymbol{x})(1 - f_m(\boldsymbol{x})) \end{pmatrix} * (\boldsymbol{V}^T \boldsymbol{\delta_2}(\boldsymbol{x})) \quad (8)$$

なお、"*" はベクトルの要素同士を乗じる演算子である。

実際の学習においては、式 (3),(4) (この操作をまとめて **イタレーション** と呼ぶ) を、出力 $\boldsymbol{h}(\boldsymbol{x})$ と教師信号 $\boldsymbol{d}(\boldsymbol{x})$ の 2 乗誤差の和

$$E = \sum_{\boldsymbol{x} \in X} ||\boldsymbol{d}(\boldsymbol{x}) - \boldsymbol{h}(\boldsymbol{x})||^2 \quad (9)$$

が所定の閾値以下になるまで繰り返す。

$$\sum_{\boldsymbol{x} \in X} \Delta \boldsymbol{V}(\boldsymbol{x}) = \underbrace{\sum_{\boldsymbol{x} \in X^{(1)}} \Delta \boldsymbol{V}(\boldsymbol{x})}_{\text{プロセッサ 1}} + \underbrace{\sum_{\boldsymbol{x} \in X^{(2)}} \Delta \boldsymbol{V}(\boldsymbol{x})}_{\text{プロセッサ 2}} + \cdots + \underbrace{\sum_{\boldsymbol{x} \in X^{(N)}} \Delta \boldsymbol{V}(\boldsymbol{x})}_{\text{プロセッサ } N}$$

図 2: データ並列。$\Delta \boldsymbol{W}$ についても同様。

3 学習アルゴリズムの並列化

3.1 データ並列

データ並列では、それぞれのプロセッサに同一の完全な重み行列 $\boldsymbol{V}, \boldsymbol{W}$ を持たせた上で、全学習サンプル X をプロセッサ数 N で分割 ($X = \{X^{(1)}, X^{(2)}, \ldots, X^{(N)}\}$) し、それらを各プロセッサに割り当てる。各プロセッサは割り当てられたサンプルすべてについて重み更新量部分和 $\sum_{\boldsymbol{x} \in X^{(i)}} \Delta \boldsymbol{V}(\boldsymbol{x}), \sum_{\boldsymbol{x} \in X^{(i)}} \Delta \boldsymbol{W}(\boldsymbol{x})$ を求める。それらの合計が最終的な重み更新量であるため、ここでプロセッサ間通信を行ない、各プロセッサが保持する重み更新量部分和の総和を求め、その結果を各プロセッサに配り直す (図 2)。

3.2 ノード並列

ノード並列では、各学習サンプル $\boldsymbol{x}$ に関する重み更新量 $\Delta \boldsymbol{V}(\boldsymbol{x}), \Delta \boldsymbol{W}(\boldsymbol{x})$ の計算を並列化する。

以下、プロセッサ数 3 の場合について説明する。図 3 は中間層 $\boldsymbol{f}(\boldsymbol{x})$ の並列計算、および出力層 $\boldsymbol{h}(\boldsymbol{x})$ の並列計算の様子である。入力層と中間層を接続する重み行列 $\boldsymbol{W}$ を **列方向** にプロセッサの数に等分し、プロセッサ p に部分行列 $\boldsymbol{W}^{(p)}$ を配置する。一方中間層と出力層を接続する重み行列 $\boldsymbol{V}$ は **行方向** に等分し、同様に配置する。また、学習サンプル $\boldsymbol{x}$ は分割せずに全プロセッサに配置しておく。

中間層 $\boldsymbol{f}$ の計算では、各プロセッサへ部分ベクトル $\boldsymbol{f}^{(1)}, \boldsymbol{f}^{(2)}, \boldsymbol{f}^{(3)}$ に分割されてしまうが、次のよう

$$\begin{pmatrix} \frac{\boldsymbol{f}^{(1)}}{\boldsymbol{f}^{(2)}} \\ \hline \boldsymbol{f}^{(3)} \end{pmatrix} = \begin{pmatrix} \frac{\boldsymbol{W}^{(1)}}{\boldsymbol{W}^{(2)}} \\ \hline \boldsymbol{W}^{(3)} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \end{pmatrix}$$

$$\begin{pmatrix} \boldsymbol{h} \end{pmatrix} = \left(\begin{array}{c|c|c} \boldsymbol{V}^{(1)} & \boldsymbol{V}^{(2)} & \boldsymbol{V}^{(3)} \end{array} \right) \begin{pmatrix} \frac{\boldsymbol{f}^{(1)}}{\boldsymbol{f}^{(2)}} \\ \hline \boldsymbol{f}^{(3)} \end{pmatrix}$$

図 3: ノード並列における中間層と出力層の計算

に各プロセッサがそれぞれ独立に計算できる。

$$\begin{aligned} \boldsymbol{f}^{(1)} &= \sigma \boldsymbol{W}^{(1)} \boldsymbol{x} \quad : \text{プロセッサ } 1 \\ \boldsymbol{f}^{(2)} &= \sigma \boldsymbol{W}^{(2)} \boldsymbol{x} \quad : \text{プロセッサ } 2 \\ \boldsymbol{f}^{(3)} &= \sigma \boldsymbol{W}^{(3)} \boldsymbol{x} \quad : \text{プロセッサ } 3 \end{aligned}$$

さらに、各プロセッサに分割されて保持されている $\boldsymbol{f}^{(p)}$ と $\boldsymbol{V}^{(p)}$ を用いて、次のように出力層 $\boldsymbol{h}$ を求める。

$$\boldsymbol{h} = \sigma(\underbrace{\boldsymbol{V}^{(1)}\boldsymbol{f}^{(1)}}_{\text{プロセッサ } 1} + \underbrace{\boldsymbol{V}^{(2)}\boldsymbol{f}^{(2)}}_{\text{プロセッサ } 2} + \underbrace{\boldsymbol{V}^{(3)}\boldsymbol{f}^{(3)}}_{\text{プロセッサ } 3}) \quad (10)$$

ここで、プロセッサ間通信により各プロセッサで計算した $\boldsymbol{V}^{(p)}\boldsymbol{f}^{(p)}$ の総和を全プロセッサに配ることで、全プロセッサに完全な (分割されていない)$\boldsymbol{h}$ を持たせるようにすると、式 (10) の計算のための通信以外には全くプロセッサ間通信が不要となる。全プロセッサが完全な $\boldsymbol{h}$ を保持するため、式 (7) も各プロセッサ独立に計算可能で (教師信号 $\boldsymbol{d}(\boldsymbol{x})$ も $\boldsymbol{x}$ と同様、全プロセッサが完全な形で保持しているとする)、$\boldsymbol{\delta}_2$ も全プロセッサがそれぞれ完全な形で保持することになる。さらに $\boldsymbol{\delta}_1$ の計算 (式 (8)) では、全プロセッサが $\boldsymbol{\delta}_2$ を完全な形で保持しているため、$\boldsymbol{f}$ の場合と同じ要領で各プロセッサ独立に計算できる。ただし、当然 $\boldsymbol{\delta}_1$ は各プロセッサへ分割 ($\boldsymbol{\delta}_1^{(1)}, \boldsymbol{\delta}_1^{(2)}$ および $\boldsymbol{\delta}_1^{(3)}$) される。最後に式 (5),(6) の計算であるが、$\boldsymbol{f}$ は各プロセッサへ分割されているが $\boldsymbol{\delta}_2$ は全プロセッサが完全な形で保持しているので $\Delta\boldsymbol{V}^{(p)}$ ($\Delta\boldsymbol{V}$ を $\boldsymbol{V}$ と同じ方法で分割した部分行列) は各プロセッサ独立に計算可能であり、一方 $\boldsymbol{\delta}_1$ は各プロセッサへ分割されているが $\boldsymbol{x}$ は全プロセッサが完全な形で保持しているので $\Delta\boldsymbol{W}^{(p)}$ ($\Delta\boldsymbol{W}$ を $\boldsymbol{W}$ と同じ方法で分割した部分行列) も各プロセッサ独立に計算可能である。

既存の手法では、ノード並列において出力層の値 $\boldsymbol{h}$ を各プロセッサへ分割しているものが多く、このため全対全のプロセッサ間通信を多数回行なっていた。本手法では、ノード並列において、出力層の値 $\boldsymbol{h}$ を全プロセッサが保持することで、それ以降の重み更新量の計算ではプロセッサ間の通信は全く行なわずに、各プロセッサがそれぞれ独立に重み更新量を計算可能となっている。

表 1: 3 種類の実験条件。n:入力層ユニット数、m:中間層ユニット数、l:出力層ユニット数、s:学習サンプル数

実験条件	n	m	l	s
a	50	1000	50	100
b	50	100	100	10000
c	50	100	50	1000

3.3 データ並列とノード並列の組み合わせ

BP アルゴリズムにおける 2 種類の並列化法は組合わせることが可能である。たとえば 8 個のプロセッサを用いる場合、それを 2 個ずつの 4 グループに分ける。そしてグループ内部では 2 個のプロセッサでノード並列を行ない、4 グループでデータ並列を行なうという方法である。この場合ノード並列度は 2、データ並列度は 4 となる。一般に、全プロセッサ数を N、データ並列度を N_{DP}、ノード並列度を N_{NP} とすれば、$N = N_{DP}N_{NP}$ という関係になる。

3.4 性能評価

並列 BP アルゴリズムを WS クラスタ上に実装し、性能評価を行なった。各 WS は UltraSparc II 300MHz を搭載し、WS 同士は 1000Base-SX のイーサネットスイッチで結合されている。WS は最大 8 台使用した。各 WS で動作している OS は Solaris 7、並列アルゴリズムの実装のためのプロセッサ間通信ライブラリとして LAM バージョン 6.5[5] を使用した。実験条件は表 1 のように 3 種類を選んだ。条件 a では学習サンプル数を少なく、条件 b では逆に多い。条件 c は a と b の中間の場合である。表 2 に使用した WS の台数と各並列化法の並列度を変化させた場合の性能を示す。この表で、例えば使用した WS の台数 $N = 8$ でノード並列度 $N_{NP} = 2$ の場合、データ並列度 $N_{DP} = N/N_{NP} = 4$ となっている。

実験条件 a では、全ての N において、ノード並列度最大 (データ並列度 1) の場合が最も高い性能を出している。条件 b では、逆にノード並列度 1 (データ並列度最大) の場合が常に最も高い性能を出している。一方、a と b の中間である条件 c では、ノード並列とデータ並列を組合わせた場合が最も高い性能を出している。このように、適切に (N_{DP}, N_{NP}) を選択すればプロセッサ台数効果を得られることがわかる。

表 2: イタレーション 1 回あたりに要した時間 (秒)。N:使用した WS の台数,N_{NP}:ノード並列度。データ並列度 $N_{DP} = N/N_{NP}$ となる。下線つき はそのプロセッサ台数で最も高い性能が出た場合。

条件 a	N_{NP}=1	N_{NP}=2	N_{NP}=4	N_{NP}=8
$N=1$	2.0	-	-	-
$N=2$	1.1	1.0	-	-
$N=4$	0.77	0.60	0.50	-
$N=8$	0.72	0.43	0.34	0.27
条件 b	N_{NP}=1	N_{NP}=2	N_{NP}=4	N_{NP}=8
$N=1$	29	-	-	-
$N=2$	14	15	-	-
$N=4$	7.8	7.9	8.3	-
$N=8$	4.3	4.4	4.5	4.9
条件 c	N_{NP}=1	N_{NP}=2	N_{NP}=4	N_{NP}=8
$N=1$	1.9	-	-	-
$N=2$	0.95	0.96	-	-
$N=4$	0.51	0.50	0.54	-
$N=8$	0.29	0.28	0.27	0.30

4 並列度の自動選択

4.1 各並列度により実行時間が変化し得る部分

第 3 章の結果から、使用するプロセッサの数や問題の規模によって、最適な並列度の組合わせ (N_{DP}, N_{NP}) が存在することが分かる。この最適な組合わせを自動的に選択するために、並列アルゴリズムの中で (N_{DP}, N_{NP}) によって実行時間が変化しうる部分を検討する。

本並列アルゴリズムでは、各プロセッサが独立に計算を行なっている部分については、問題の規模とプロセッサの数が一定である限り各プロセッサに割り当てられる計算量は (N_{DP}, N_{NP}) にかかわらずほぼ一定であり、実行時間が変化しないはずである。したがって、(N_{DP}, N_{NP}) により実行時間が変化し得るのは、プロセッサ間通信を行なっている部分である。すなわち、データ並列では各プロセッサが保持する重み行列の部分和を集めてその合計を各プロセッサに配り直す部分 (図 2)、ノード並列では出力層 $\boldsymbol{h}$ を計算している部分である (式 (10))。この部分は、(N_{DP}, N_{NP}) の組合わせによりプロセッサ間の通信量が変化するので、実行時間が変化し得る。前者の部分の実行時間を t_{DP}、後者の部分の実行時間を t_{NP} とすると、$T = t_{DP} + t_{NP}$ を最小化する (N_{DP}, N_{NP}) を選べばよいことになる。

4.2 オールリデュース操作

本手法でプロセッサ間通信が行なわれる部分の操作は、本質的には**オールリデュース操作**となっている。ここでオールリデュース操作とは、各プロセッサが保持している同じ要素数のベクトル (行列) をすべて足し合わせ、その結果を各プロセッサに配り直すという操作である。ここで、ある WS クラスタにおいて、長さ n のベクトルをプロセッサ数 p の間でオールリデュースするのにかかる時間を $T_{allreduce}(n,p)$ とすると、イタレーション 1 回あたりの t_{DP} および t_{NP} は次のように書ける。

$$t_{DP} = T_{allreduce}\left(\frac{nm}{N_{NP}}, N_{DP}\right) + T_{allreduce}\left(\frac{ml}{N_{NP}}, N_{DP}\right) \quad (11)$$

$$t_{NP} = T_{allreduce}\left(\frac{sl}{N_{DP}}, N_{NP}\right) \quad (12)$$

本研究で使用しているプロセッサ間通信ライブラリ LAM に付属するオールリデュース操作のアルゴリズムでは、木構造通信によりリデュースおよびブロードキャスト [4] を行なうので、ステージ数 $2\lceil \log_2 p \rceil$ で行なうことができる。したがって、

$$T_{allreduce}(n,p) = \alpha n \lceil \log_2 p \rceil + \beta \lceil \log_2 p \rceil \quad (13)$$

とモデル化できる。第 1 項はネットワークのスループットに関する項、第 2 項は (ベクトルの要素数に関わらず発生する) ネットワークの遅延に関する項である。未知数が 2 個なので、2 種類の適当な (n,p) の組を選び実際にオールリデュース操作を行なってそれに要した時間を測定すれば α と β が求まり、そのクラスタにおける $T_{allreduce}(n,p)$ を完全に決定できる。これを用いて $T = t_{DP} + t_{NP}$ を最小化する (N_{DP}, N_{NP}) を決定できる。

4.3 自動選択手法の評価

以上の並列度選択法を用いて、表 1 の実験条件の下でノード並列度 $N_{NP}(= N/N_{DP})$ を選択した結果が表 3 である。ノード並列が最大またはデータ並列が最大の場合が、最適な並列度の組合わせとなる構成の場合、求められた組合わせは最適な組合わせに一致した。しかし、データ並列とノード並列を組合わせた場合が最適な構成の場合、一致しなかった。これは 式 (13) のモデルが不十分なためと考えられる。

そこで、あらかじめ様々な (n,p) でオールリデュース操作を行ない、その時間を測定し、$T_{allreduce}(n,p)$ の表を作成しておく。なお、あらゆる (n,p) で時間を測定するの不可能なので、n は適当な間隔で行な

表 3: モデル $T_{allreduce}(n,p) = \alpha n\lceil \log_2 p \rceil + \beta \lceil \log_2 p \rceil$ を用いた並列度選択法で求められた N_{NP} (○が最適な並列度を求められた場合。×は求められなかった場合。

	$N=1$	$N=2$	$N=4$	$N=8$
条件 a	○1	○2	○4	○8
条件 b	○1	○1	○1	○1
条件 c	○1	×2	×4	×8

表 4: $T_{allreduce}(n,p)$ の表を用いた並列度選択法で求められた N_{NP} (○が最適な並列度を求められた場合。×は求められなかった場合。)

	$N=1$	$N=2$	$N=4$	$N=8$
条件 a	○1	○2	○4	○8
条件 b	○1	○1	○1	○1
条件 c	○1	○1	○2	○4

い、その間は線形補完した値を用いる。そのようにして得られた $T_{allreduce}(n,p)$ の表を用いて、改めて並列度の選択を行なった結果が表 4 である。この手法ではすべての場合で最適な並列度の組合わせを選択することができた。

この表の作成には非常にコストがかかるが、1 回作ってしまえばあらゆる規模の問題にたいして適用可能である。

5 非一様環境への対応

5.1 学習サンプルの非一様な割り当て

WS クラスタは、必ずしもすべての WS の性能が一様であるとは限らない。第 3 章で提案した並列化手法では、1 つでも性能の低いプロセッサが混じっていた場合、全体の性能がそのプロセッサの性能で制限されてしまう。そこで、各プロセッサの計算性能に応じて、割り当てる学習サンプルの数の割合を変える。まず、能力が上位のプロセッサから順に N_{NP} 個ずつのグループを作る。各グループの内部でノード並列を行ない、全グループでデータ並列を行なうのは 3.3 節で述べた手法と同様であるが、この時、各グループに割り当てる学習サンプルの数の割合を**そのグループで最も低い性能のプロセッサの能力**にしたがって分配する。例えば、8 台の WS があり、それぞれの能力が

$$\{2.1, 2.0, 1.8, 1.7, 1.0, 0.8, 0.5, 0.2\}$$

表 5: 非一様環境におけるイタレーション 1 回に要した時間 (秒)

実験条件 a				
N_{NP}	1	2	4	8
非一様環境対応あり	4.8	6.3	8.2	10
非一様環境対応なし	10	9.9	10	10

実験条件 b				
N_{NP}	1	2	4	8
非一様環境対応あり	0.72	0.45	0.40	0.57
非一様環境対応なし	0.72	0.65	0.61	0.62

と分かっているものとする。$N_{NP}=2$ とする場合、上位の能力のプロセッサから 2 個ずつのグループを作っていく。

$$\{2.1, 2.0\}, \{1.8, 1.7\}, \{1.0, 0.8\}, \{0.5, 0.2\}$$

そして、上位のグループから、学習サンプルの数を 2.0 : 1.7 : 0.8 : 0.2 の割合で分配する。

上位からグループを作っていくのは、能力がなるべく同じプロセッサを一緒にするためである。能力の高いプロセッサと能力の低いプロセッサを一緒にしてノード並列を行なうと、そのグループの性能は最も低い性能のプロセッサに押えられてしまう。学習サンプルを各グループで最も低い性能のプロセッサの能力で分配するのは、そのグループの各プロセッサは、その最も低い性能のプロセッサの能力しか発揮できないからである。

5.2 性能評価

以上の負荷分散手法を用いて、非一様な環境における性能を調べた。実験では 8 台の WS を用いたが、そのうちの 1 台にビジーループを実行するプログラムをバックグラウンドで動作させ、その WS の見かけの計算速度を 1/2 にした状態で実験を行なった。実験条件は第 3 章での条件 a,b と同じである。

表 5 に実験結果を示す。非一様環境への対応がないマッピングでは一様環境にくらべ大幅に性能が悪化しているが、対応があるマッピングでは性能の悪化が押えられている。

実験条件 a は、一様環境においては N_{DP} が最大の場合に最も高い性能が出ていた構成であり、非一様環境でも N_{DP} が最大の場合に最も高い性能が出ている。一方実験条件 b は、一様環境においては N_{NP} が最大の場合に最も高い性能が出ていた構成であるが、非一様環境ではデータ並列も少し組合わせた場合のほうが最も高い性能が出ている。これは、本手

法の非一様環境への対応がデータ並列のみを考慮したものであり、WS の能力の割合に応じて重み行列の分割の割合を変えるといった、ノード並列を考慮した負荷分散ではないためであると考えられる。したがって、一様環境で最適であった (N_{DP}, N_{NP}) か必ずしも非一様環境では最適ではないということが分かる。

5.3 非一様環境における並列度の自動選択

一様環境では、プロセッサ間通信が必要なく各プロセッサが独立に計算できていた部分は (N_{DP}, N_{NP}) の組合わせによって実行時間は変化しないと仮定できる。しかし、非一様環境になると (N_{DP}, N_{NP}) の組合わせにより変化し得る。したがって、t_{DP}, t_{NP} だけでなく、各プロセッサが独立に計算している部分実行時間 t_{IC} を加えた $T' = t_{DP} + t_{NP} + t_{IC}$ を最小化するような (N_{DP}, N_{NP}) を選択する必要がある。

本手法における非一様環境を考慮した負荷分散では、ノード並列のグループは、各プロセッサの能力の上位から N_{NP} 個ずつをグループ化したものである。しかし、ノード並列は全く考慮しない負荷分散なので、各グループが発揮できる計算能力はそのグループ内の最も低い性能のプロセッサで制限される。言い換えれば、そのプロセッサの能力を a_{worst} とすれば、そのグループが発揮できる総合能力はグループ内のプロセッサの能力の和ではなく、$a_{worst}N_{NP}$ に制限されてしまう。したがって、各グループ $i(i = 1, 2, \ldots, N_{DP})$ のプロセッサの中で、最も低い能力を $a^{(i)}_{worst}$ とすると、本手法における負荷分散で全プロセッサが発揮できる能力の合計 a_{total} は、

$$a_{total} = \sum_{i=1}^{N_{DP}} N_{NP} a^{(i)}_{worst}$$

である。これが、非一様環境で t_{IC} が変化し得る理由である。

この a_{total} を用いて、ある並列度の組合わせ (N_{DP}, N_{NP}) における $t_{IC}(N_{DP}, N_{NP})$ から、別の組合わせ (N'_{DP}, N'_{NP}) における $t_{IC}(N'_{DP}, N'_{NP})$ を計算することができる。すなわち、(N_{DP}, N_{NP}) における能力を $a_{total}(N_{DP}, N_{NP})$ とすれば、

$$t_{IC}(N'_{DP}, N'_{NP}) = \frac{a_{total}(N_{DP}, N_{NP})}{a_{total}(N'_{DP}, N'_{NP})} t_{IC}(N_{DP}, N_{NP})$$

と書ける。各プロセッサの能力が既知であれば、任意の (N_{DP}, N_{NP}) について $a_{total}(N_{DP}, N_{NP})$ を計算することができる。ある 1 つの (N_{DP}, N_{NP}) について、$t_{IC}(N_{DP}, N_{NP})$ を測定できれば、上式を用いてあらゆる (N_{DP}, N_{NP}) について、$t_{IC}(N_{DP}, N_{NP})$ を求めることができ、$T' = t_{DP} + t_{NP} + t_{IC}$ を最小化する (N_{DP}, N_{NP}) を得ることが可能である。

表 6: 非一様環境を考慮した並列度選択法の結果。○が最適な並列度を求められた場合。×は求められなかった場合。

実験条件 a	
求まった N_{NP}	○ 1
実験条件 b	
求まった N_{NP}	○ 4

以上の手法を用いて、5.2 節における実験条件の下で並列度の組合わせ (N_{DP}, N_{NP}) を求めた (表 6)。実験では能力の値として単位時間にその WS が処理できる積和演算の長さを用いている。最適な (N_{DP}, N_{NP}) が求まっていることがわかる。

6 まとめ

WS クラスタ上に BP 学習アルゴリズムを効率的にマッピングする手法を提案した。特にノード並列方式では、出力層の値を全プロセッサが保持するようにすることで、大幅にプロセッサ間通信を減らしている。また各並列化法の並列度を自動的に選択でき、さらに非一様な WS クラスタにも対応する。$T_{allreduce}$ のより正確なモデルを作ることが課題である。

参考文献

[1] J.Torressen, H.Nakashima, S.Tomita and O.Landsverk: General Mapping of Feed-Forward Neural Network onto an MIMD Computer, *IEEE International Conference on Neural Network*, 1995.

[2] 吉田英嗣, 安永守利: 超並列計算機 CP-PACS によるニューラルネットワーク計算の高速化, **情報処理学会研究報告** *97-HPC-67*,pp.25-30 (1997).

[3] 上坂吉則, 尾関和彦: パターン認識と学習のアルゴリズム, **文一総合出版**, 1990.

[4] P. パチェコ著, 秋葉博訳: MPI 並列プログラミング, **培風館**, 2001.

[5] LAM team: LAM/MPI, *http://www.lam-mpi.org*