修士学位論文

大規模ニューラルネットワーク学習の分散高速化アルゴリズムに関する研究

東北大学大学院 工学研究科 電気・通信工学専攻
東 大亮

目 次

第 1 章　序論 **1**

1.1　本研究の背景 1

1.2　本研究の目的 2

1.3　本論文の構成 4

第 2 章　パーセプトロンと誤差逆伝播学習法 **5**

2.1　3 層パーセプトロン 5

2.2　誤差逆伝播学習法 7

第 3 章　学習アルゴリズムの並列化 **14**

3.1　学習セッション並列 14

3.2　データ並列 14

3.3　ノード並列 15

3.4　データセット並列とノード並列の組み合わせ 18

3.5　遅延重み更新 18

3.6　性能評価 21

3.6.1　遅延重み更新なしの場合 21

3.6.2　遅延重み更新ありの場合 27

第 4 章　並列度の自動選択と非均一環境への対応 **29**

4.1　並列度の自動選択 29

4.1.1　各並列度により実行時間が変化し得る部分 29

4.1.2　オールリデュース操作 30

4.1.3　自動選択手法の評価 34

4.2　非均一環境への対応 37

4.2.1　学習サンプルの非均一な割り当て 37

4.2.2　性能評価 37

4.2.3　非均一環境における並列度の自動選択 40

第 5 章　結論 **45**

5.1　本研究の成果 45

5.2　今後の課題 45

参考文献 48

図 目 次

1.1 WS クラスタの構成例 3

2.1 3 層パーセプトロン 6
2.2 BP 学習アルゴリズム 13

3.1 データ並列 16
3.2 ノード並列における中間層と出力層の計算 19
3.3 ノード並列における δ_1 の計算 19
3.4 ノード並列における $\Delta \boldsymbol{V}(\boldsymbol{x}), \Delta \boldsymbol{W}(\boldsymbol{x})$ の計算 19
3.5 データセット並列とノード並列の組み合わせ 20
3.6 通常の重み更新処理 22
3.7 遅延重み更新 23
3.8 実験条件 a でイタレーション 1 回あたりに要した時間 (遅延重み更新なし) . 24
3.9 実験条件 b でイタレーション 1 回あたりに要した時間 (遅延重み更新なし) . 25
3.10 実験条件 c でイタレーション 1 回あたりに要した時間 (遅延重み更新なし) . 26

4.1 オールリデュース操作 32
4.2 LAM における木構造オールリデュース操作 33
4.3 $d = 100000$ でオールリデュース操作に要した時間 35
4.4 様々な (d, p) においてオールリデュース操作に要した時間 36
4.5 学習サンプルの非均一な割り当て 38
4.6 ノード並列においてそのグループの最も低い能力のプロセッサで全体の性能が制限されてしまう様子 39
4.7 非均一環境下での性能 (実験条件 a) 41
4.8 非均一環境下での性能 (実験条件 b) 42

表 目 次

3.1 3 種類の実験条件 21
3.2 イタレーション 1 回あたりに要した時間 (遅延重み更新を行なった場合) . . 28

4.1 プロセッサ 1 個に割り当てられる計算量 (プロセッサ独立に計算できる部分) 31
4.2 モデル $T_{allreduce}(d,p) = \alpha d\lceil \log_2 p \rceil + \beta \lceil \log_2 p \rceil$ を用いた並列度選択法で求めた N_{NP} 34
4.3 $T_{allreduce}(d,p)$ の表を用いた並列度選択法で求められた N_{NP} 35
4.4 非均一環境を考慮した並列度選択法の結果。○ が最適な並列度に一致した場合。× は一致しなかった場合。 44

第1章

序論

1.1 本研究の背景

1940 年代に誕生したコンピュータは、半世紀以上を過ぎた現在でも常に高速化・大容量化が追求されているが、一方で同時に高度な知的情報処理を実現するための手法が求められてきた。その 1 つが生体の神経細胞をモデル化した人工ニューラルネットワークである。しかし、人工ニューラルネットワークは多数の神経細胞とその結合の挙動をモデル化するものであるゆえに、その処理に要する計算量は一般に多大なものとなっている。これに対し、ネットワークの構造や学習法そのものを様々に工夫することで計算量を減らし高速化するための研究が多数なされている一方で、人工ニューラルネットワークを並列計算機にマッピングすることで高速化を目指すアプローチがある。

人工ニューラルネットワークの中でも、階層型ニューラルネットワークの 1 つである多層パーセプトロンとその学習法である誤差逆伝播学習法 (Back Propagation, 以下 BP と略する) は、良い識別性能を持つ識別器を学習サンプルから比較的簡単に構成できるために応用範囲が広く、様々な汎用並列計算機上に実装されている。Singer[1], Nordstrom [3] らによると、BP アルゴリズムには以下のような複数種類の並列性が内在しており、多くの並列実装ではこれらのいずれかを単体で、あるいは組み合わせて利用している。

- 学習セッション並列
  異なるプロセッサ上で、異なるネットワーク初期値からの学習セッションを開始する。
- データ並列
  学習サンプル集合を分割し、プロセッサに分配する。各プロセッサはそれぞれ完全な重み行列のコピーを持ち、それに基づいて与えられた学習サンプルの集合について重み更新量を計算する。最後に各プロセッサで計算された重み更新量を合計することで、完全な重み更新量を得る。

- ノード並列
  ネットワークを分割し、複数のプロセッサで処理を分担する。

Torressen [6] らは、並列計算機上に BP アルゴリズムをマッピングして高い性能を得るためには問題の規模によってこれら複数種類の並列化法を組み合わせるべきであると主張し、自由に各並列化法の並列度を選択できるマッピングを提案した。しかし、あるプロセッサ台数と問題の規模において最高の性能を発揮するための具体的な並列性の組み合わせの選択はユーザによる発見的手法に頼っており、特にプロセッサ数が多い場合などは最適な並列度の組み合わせを求めるのは困難である。したがって自動的に組み合わせを選択し、性能をチューニングする機構が求められる。

Torressen らは上記の研究で 2 次元トーラス型ネットワークで接続された MIMD マシン上へのマッピングを提案しているが、本研究では比較的安価でありながら高い性能を得ることができる並列計算機の 1 つであるワークステーションクラスタ (WS クラスタ) へのマッピングを対象とする。WS クラスタは、安価な PC や WS を多数用意し、これらを Ethernet などで結合することで並列計算機を構成するものである (図 1.1)。このような WS クラスタは、一般にプロセッサの能力に比してネットワークの能力 (転送速度や遅延) がそれほど高くない場合が多く、これがボトルネックとなり高い性能を発揮できないケースが多い。BP アルゴリズムは多数の神経細胞同士の結合をシミュレートするものであり、WS クラスタのような特にネットワーク性能が問題となる並列計算機においては単純なマッピングでは高い性能を引き出すことが困難であることは容易に推測できる。

さらに、WS クラスタはすべての WS の能力が必ずしも同等とは限らない。初めに各 WS の能力が揃っているクラスタを導入したが、後で異なる能力の WS を追加したという場合も考えられるし、もともと様々な種類の計算機が混在している LAN を、WS クラスタとして使いたいという要求もあるだろう。そのような環境でも、各 WS の能力に応じてタスクの割り当て量を変化させ、プロセッサの利用率を高く保つことで全体の計算時間を削減できるようなマッピングが望ましい。

1.2 本研究の目的

以上のような背景を踏まえ、本研究では以下のような方針により WS クラスタ上に BP アルゴリズムを効率的にマッピングする手法を提案する。

- ワークステーションクラスタのようなプロセッサ間通信の速度が低い並列計算機でも高い性能を得るような BP アルゴリズムのマッピングを目指す。
- BP アルゴリズムにおける並列性の並列度の組み合わせを自由に選択できるようなマッピングとその組み合わせを自動的に選択する機構の開発。
- 各 WS の能力が非均一な環境でも、能力に応じてタスク割り当て量を変化させ、高い性能を引き出せるようなマッピングを行う。

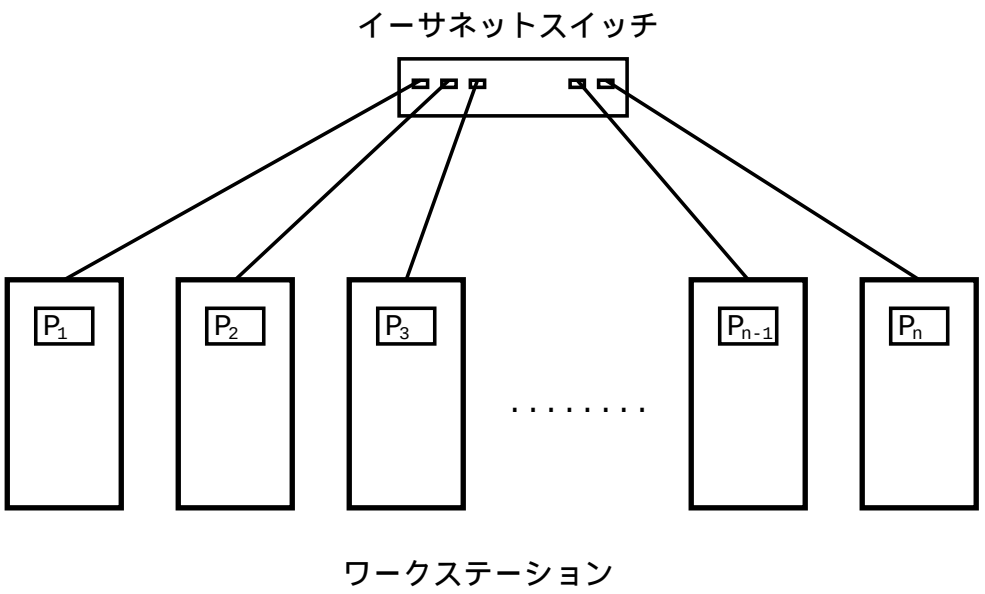


図 1.1: WS クラスタの構成例

1.3 本論文の構成

第 1 章 序章

本研究の目的と背景を述べる。

第 2 章 パーセプトロンと誤差逆伝播学習法

本研究で扱うニューラルネットワークである 3 層パーセプトロンの構造と、その学習法である誤差逆伝播学習 (BP) 法について述べる。

第 3 章 学習アルゴリズムの並列化

第 2 章で述べた BP アルゴリズムを WS クラスタにマッピングし、性能評価を行なう。

第 4 章 並列度の自動選択と非均一環境への対応

BP アルゴリズムの各並列化手法の並列度を自動的に選択する手法を述べる。さらに、非均一な WS クラスタにも対応できるようにタスク割り当てアルゴリズムを変更する。

第 5 章 結論

本研究の成果と、今後の課題について述べる。

第2章

パーセプトロンと誤差逆伝播学習法

パーセプトロンは、Rosenblatt [7] によってその原型が提案された階層型ニューラルネットワークである。これは、初めて脳の神経回路網を理論的に具現化したモデルとしてではなく、世界初の学習する機械としてパターン認識の分野等で大きな注目を集めたが、Minsky [8] らによりその学習能力の限界が指摘され、実際の工学的な応用にまでは至らなかった。しかし、後に Rumelhart [9] らが Rosenblatt のモデルを拡張し、新たな学習則である誤差逆伝播学習法 を適用することで、Minsky らが指摘した問題に対しても良い近似解を与えることを示し、パーセプトロンは再び注目を集めることとなる。現在では、パターン認識や時系列予測といった多くの分野への適用が行われ、神経回路網理論のみならず、工学的意味においても非常に重要な位置を占める階層型数理モデルの一つとして知られている。

本章では、本研究で扱うパーセプトロンの構造と、その学習法である誤差逆伝播学習法 [5] について述べる。

2.1 3層パーセプトロン

パーセプトロンは一般に多階層のネットワークであるが、本研究では 3 層構造のパーセプトロンを対象とする。3 層パーセプトロンは、図 2.1 のように n 個のユニットからなる入力層

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T \tag{2.1}$$

が m 個のユニットからなる中間層

$$\boldsymbol{f} = (f_1, f_2, \ldots, f_m)^T \tag{2.2}$$

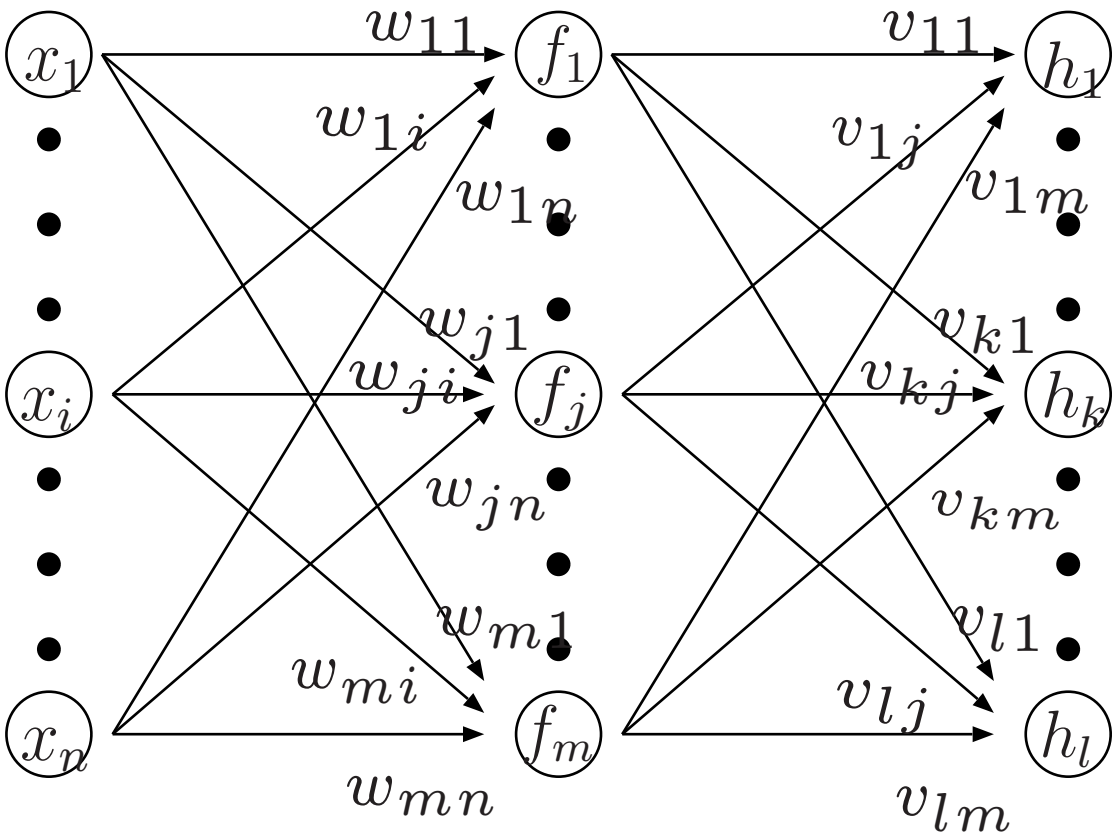


図 2.1: 3 層パーセプトロン

と結合重み行列

$$\boldsymbol{W} \quad = \quad [w_{ji}] \quad (i = 0, 1, \ldots, n, j = 1, 2, \ldots, m) \tag{2.3}$$

によって、

$$f_j \quad = \quad \sigma(w_{j0} + w_{j1}x_1 + w_{j2}x_2 + \ldots + w_{jn}x_n) \quad (j = 1, 2, \ldots, m) \tag{2.4}$$

という形で接続され、さらに l 個のユニットからなる出力層

$$\boldsymbol{h} = (h_1, h_2, \ldots, h_l)^T \tag{2.5}$$

が結合重み行列

$$\boldsymbol{V} \quad = \quad [v_{kj}] \quad (j = 0, 1, \ldots, m, k = 1, 2, \ldots, l) \tag{2.6}$$

によって中間層 $\boldsymbol{f}$ と

$$h_k \quad = \quad \sigma(v_{k0} + v_{k1}f_1 + v_{k2}f_2 + \ldots + v_{jm}f_m) \quad (k = 1, 2, \ldots, l) \tag{2.7}$$

という形で接続されているという構造である。ここで $\sigma(s)$ は一般に非線型関数であるが、本研究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

$$\sigma(s) \quad = \quad \frac{1}{1 + e^{-s}} \tag{2.8}$$

を採用する。このネットワークに入力 $\boldsymbol{x}$ を与えたときの式 (2.4) で決まる $\boldsymbol{f}$ の値を $\boldsymbol{f}(\boldsymbol{x})$、式 (2.7) で決まる $\boldsymbol{h}$ を $\boldsymbol{h}(\boldsymbol{x})$ と記す。ネットワークの出力 $\boldsymbol{h}(\boldsymbol{x})$ を $\boldsymbol{x}$ の関数と見ると、これはネットワークの機能を表し、$\boldsymbol{W}, \boldsymbol{V}$ いよって定まる。

この $\boldsymbol{W}, \boldsymbol{V}$ を正しく決めることで、例えば $\boldsymbol{x}$ として文字パターンが入力され、$\boldsymbol{h}$ にその文字に対応するクラスが出力されるような識別器を構成できる。

2.2 誤差逆伝播学習法

式 (2.1) の $\boldsymbol{x}$ は任意の n 次元ベクトルをとることができるので、n 次元ユークリッド 空間

$$S \quad \equiv \quad \{\boldsymbol{x} | \boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T, x_i \in \mathbf{R}\} \tag{2.9}$$

を考え、これを パターン空間 と呼ぶ。また、関数 $\boldsymbol{h}(\boldsymbol{x})$ の取る値の集合

$$C \quad \equiv \quad \{\boldsymbol{h} | \boldsymbol{h} = (h_1, h_2, \ldots, h_l)^T, h_k \in \mathbf{R}\} \tag{2.10}$$

をカテゴリ集合 と呼ぶことにする。このとき、S から C へのある関数はパターンをカテゴリに対応させる識別関数 と見ることができる。

いま、識別関数 $\boldsymbol{d}$ が固定されていて、その全体は分かっていないが S の有限部分集合 X における $\boldsymbol{d}$ の値、すなわち

$$\boldsymbol{d}(\boldsymbol{x}), \boldsymbol{x} \in X \tag{2.11}$$

は既知であるとする。このときパーセプトロンの重み $\boldsymbol{W}, \boldsymbol{V}$ をうまく決めて、$\boldsymbol{h}$ が $\boldsymbol{d}$ をできるだけ忠実にあらわすようにしたいという問題を考える。パーセプトロンを学習認識機械と見立てると、これは学習パターンから目標の識別関数を習得することだとも言える。この意味で $\boldsymbol{d}$ を目標関数 といい、X に属するベクトルを学習パターン、X を学習パターン集合 と呼ぶことにする。

目標関数 $\boldsymbol{d}$ とパーセプトロンの出力 $\boldsymbol{h}$ の近さを測るために、誤差を

$$E \equiv \frac{1}{2|X|} \sum_{\boldsymbol{x} \in X} ||\boldsymbol{h}(\boldsymbol{x}) - \boldsymbol{d}(\boldsymbol{x})||^2 \tag{2.12}$$

と定める。ここで $|X|$ は X に属するパターンの個数を表し、$||\boldsymbol{z}||$ はベクトル $\boldsymbol{z}$ のユークリッド ノルム

$$||\boldsymbol{z}|| \equiv \sqrt{z_1^2 + z_1^2 + \ldots + z_n^2} \tag{2.13}$$

を表す。

この時、誤差 E は重み $\boldsymbol{W}, \boldsymbol{V}$ の関数とみることができるので、この E を最小とする重みを求めればよいことになる。しかし、$\boldsymbol{h}$ の定義は相当複雑であり、E を最小化する $\boldsymbol{W}, \boldsymbol{V}$ を解析的に求めることは困難である。そこで、最急降下法 を用いて逐次近似的に E を最小化する $\boldsymbol{W}, \boldsymbol{V}$ を求めていく。

ここで、重み $\boldsymbol{W}, \boldsymbol{V}$ を時間の関数と考え、それらに関する次のような微分方程式系を用意する。

$$\frac{dv_{kj}}{dt} = -\frac{\partial E}{\partial v_{kj}} \tag{2.14}$$

$$\frac{dw_{ji}}{dt} = -\frac{\partial E}{\partial w_{ji}} \tag{2.15}$$

E は重みの関数であり、各重みは時間の関数と考えたので、E も時間の関数と考えることができる。このとき、この微分方程式系は 1 つの力学系をなす。この力学系に従い重みが動いているとき、誤差 E が時間に関してどのように変動していくかを調べる。そのために E を微分してみると、

$$\frac{dE}{dt} = \sum_{j=0}^{m} \sum_{k=1}^{l} \frac{\partial E}{\partial v_{kj}} \frac{dv_{kj}}{dt} + \sum_{i=0}^{n} \sum_{j=1}^{m} \frac{\partial E}{\partial w_{ji}} \frac{dw_{ji}}{dt} \tag{2.16}$$

となる。これに 式 (2.14) と式 (2.15) を代入すると、

$$\frac{dE}{dt} = -\sum_{j=0}^{m} \sum_{k=1}^{l} \left[\frac{\partial E}{\partial v_{kj}}\right]^2 - \sum_{i=0}^{n} \sum_{j=1}^{m} \left[\frac{\partial E}{\partial w_{ji}}\right]^2 \leq 0 \tag{2.17}$$

が得られ、誤差は時間 t に対して非増加であることがわかる。誤差が不変になるのは、この式の右辺から分かるように E の重みによる偏微分 $\frac{\partial E}{\partial v_{kj}}, \frac{\partial E}{\partial w_{ji}}$ が 0 になるときでしかもそのときに限る。

以上をまとめると、 式 (2.14) と式 (2.15) の力学系においては誤差 E は時間 t について非増加である。つまり、

$$\frac{dE}{dt} \leq 0 \tag{2.18}$$

が任意の t について成り立つ。また、等号が成り立つのは、任意の i, j, k について

$$\begin{cases} \frac{\partial E}{\partial v_{kj}} = 0 \\ \frac{\partial E}{\partial w_{ij}} = 0 \end{cases} \tag{2.19}$$

が成り立つときに限る。

つまり、重みを適当な初期値に設定してこの力学系を起動すると 式 (2.14) と 式 (2.15) の力学系の軌道は誤差 E の極小点に停留する。したがってその力学系をシミュレートすることができれば誤差を最小にする重みを求めることができる。

ここで 式 (2.14) と 式 (2.15) の右辺を 式 (2.12) から具体的に求めてみる。

$$\boldsymbol{d}(\boldsymbol{x}) = (d_1(\boldsymbol{x}), d_2(\boldsymbol{x}), \ldots, d_l(\boldsymbol{x})), \tag{2.20}$$

$$\boldsymbol{h}(\boldsymbol{x}) = (h_1(\boldsymbol{x}), h_2(\boldsymbol{x}), \ldots, d_l(\boldsymbol{x})) \tag{2.21}$$

として、まず E の v_{kj} による微分を計算すると、

$$\begin{aligned} \frac{\partial E}{\partial v_{kj}} &= \frac{1}{|X|} \sum_{\boldsymbol{x} \in X} \sum_{p=1}^{l} [h_p(\boldsymbol{x}) - d_p(\boldsymbol{x})] \frac{\partial h_p(\boldsymbol{x})}{\partial v_{kj}} \\ &= \frac{1}{|X|} \sum_{\boldsymbol{x} \in X} [h_k(\boldsymbol{x}) - d_k(\boldsymbol{x})] \frac{\partial}{\partial v_{kj}} \sigma(v_{k0} + v_{k1} f_1(\boldsymbol{x}) + \ldots + v_{km} f_m(\boldsymbol{x})) \end{aligned} \tag{2.22}$$

が得られるが、

$$\frac{d\sigma}{ds} = \sigma(s)(1 - \sigma(s)) \tag{2.23}$$

に注意すると、

$$\begin{aligned} \frac{\partial}{\partial v_{kj}} \sigma(v_{k0} + v_{k1} f_1(\boldsymbol{x}) + \ldots + v_{km} f_m(\boldsymbol{x})) &= h_k(\boldsymbol{x})(1 - h_k(\boldsymbol{x})) \frac{\partial}{\partial v_{kj}} (v_{k0} + v_{1k} f_1(\boldsymbol{x}) + \ldots + v_{km} f_m(\boldsymbol{x})) \\ &= h_k(\boldsymbol{x})(1 - h_k(\boldsymbol{x})) f_j(\boldsymbol{x}) \end{aligned} \tag{2.24}$$

となる。ただし。この式が $j = 0$ でも成り立つように便宜上 $f_0(\boldsymbol{x}) \equiv 1$ と決めておく。式 (2.24) を 式 (2.22) に代入すると、

$$\frac{\partial E}{\partial v_{kj}} = \frac{1}{|X|} \sum_{\boldsymbol{x} \in X} [h_k(\boldsymbol{x}) - d_k(\boldsymbol{x})] h_k(\boldsymbol{x})(1 - h_k(\boldsymbol{x})) f_j(\boldsymbol{x}) \tag{2.25}$$

となるが、ここで

$$\delta_{2k}(\boldsymbol{x}) \equiv [h_k(\boldsymbol{x}) - d_k(\boldsymbol{x})]h_k(\boldsymbol{x})(1 - h_k(\boldsymbol{x})) \tag{2.26}$$

とおくと、結局

$$\frac{\partial E}{\partial v_{kj}} = \frac{1}{|X|}\sum_{\boldsymbol{x}inX}\delta_{2k}(\boldsymbol{x})f_j(\boldsymbol{x}) \tag{2.27}$$

と書けることになる。

次に E の w_{ji} による偏微分を計算する。

$$\begin{aligned}
\frac{\partial E}{\partial w_{ji}} &= \frac{1}{|X|}\sum_{\boldsymbol{x}\in X}\sum_{p=1}^{l}[h_p(\boldsymbol{x}) - d_p(\boldsymbol{x})]\frac{\partial h_p(\boldsymbol{x})}{\partial w_{ji}} \\
&= \frac{1}{|X|}\sum_{\boldsymbol{x}\in X}\sum_{p=1}^{l}[h_p(\boldsymbol{x}) - d_p(\boldsymbol{x})]\frac{\partial}{\partial w_{ji}}\sigma(v_{p0} + v_{p1}f_1(\boldsymbol{x}) + \ldots + v_{pm}f_m(\boldsymbol{x})) \\
&= \frac{1}{|X|}\sum_{\boldsymbol{x}\in X}\sum_{p=1}^{l}[h_p(\boldsymbol{x}) - d_p(\boldsymbol{x})]h_p(\boldsymbol{x})(1 - h_p(\boldsymbol{x}))\frac{\partial}{\partial w_{ji}}(v_{p0} + v_{p1}f_1(\boldsymbol{x}) + \ldots + v_{pm}f_m(\boldsymbol{x})) \\
&= \frac{1}{|X|}\sum_{\boldsymbol{x}\in X}\sum_{p=1}^{l}\delta_{2p}(\boldsymbol{x})\frac{\partial}{\partial w_{ji}}(v_{pj}f_j(\boldsymbol{x})) \\
&= \frac{1}{|X|}\sum_{\boldsymbol{x}\in X}\sum_{p=1}^{l}\delta_{2p}(\boldsymbol{x})v_{pj}\frac{\partial}{\partial w_{ji}}f_j(\boldsymbol{x})
\end{aligned} \tag{2.28}$$

が得られるが、

$$\begin{aligned}
\frac{\partial}{\partial w_{ji}}f_j(\boldsymbol{x}) &= \frac{\partial}{\partial w_{ji}}\sigma(w_{j0} + w_{j1}x_1 + \ldots + w_{jn}x_n) \\
&= f_j(\boldsymbol{x})(1 - f_j(\boldsymbol{x}))\frac{\partial}{\partial w_{ji}}(w_{j0} + w_{j1}x_1 + \ldots + w_{jn}x_n) \\
&= f_j(\boldsymbol{x})(1 - f_j(\boldsymbol{x}))x_i
\end{aligned} \tag{2.29}$$

となる。ただし、この式が $i = 0$ でも成り立つように便宜上 $x_0 \equiv 1$ と決めておく。式 (2.29) を 式 (2.28) に代入すると、

$$\frac{\partial E}{\partial w_{ji}} = \frac{1}{|X|}\sum_{\boldsymbol{x}\in X}\left\{\sum_{p=1}^{l}\delta_{2p}(\boldsymbol{x})v_{pj}\right\}f_j(\boldsymbol{x})(1 - f_j(\boldsymbol{x}))x_i \tag{2.30}$$

となるが、ここで

$$\delta_{1j}(\boldsymbol{x}) = \left\{\sum_{p=1}^{l}\delta_{2p}(\boldsymbol{x})v_{pj}\right\}f_j(\boldsymbol{x})(1 - f_j(\boldsymbol{x})) \tag{2.31}$$

とおくと、

$$\frac{\partial E}{\partial w_{ji}} = \frac{1}{|X|}\sum_{\boldsymbol{x}\in X}\delta_{1j}(\boldsymbol{x})x_i \tag{2.32}$$

が得られる。

このようにして得られた微分の結果である 式 (2.32) と式 (2.27) を元の力学系である 式 (2.14) と 式 (2.15) に代入すると、

$$\frac{dv_{kj}}{dt} = -\frac{1}{|X|}\sum_{\boldsymbol{x}\in X}\delta_{2k}(\boldsymbol{x})f_j(\boldsymbol{x}) \tag{2.33}$$

$$\frac{dw_{ji}}{dt} = -\frac{1}{|X|}\sum_{\boldsymbol{x}\in X}\delta_{1j}(\boldsymbol{x})x_i \tag{2.34}$$

という具体的な力学系が得られる。

したがって微分の近似式

$$\frac{dv_{kj}}{dt} \simeq \frac{v_{kj}(t+\epsilon)-v_{kj}(t)}{\epsilon} \tag{2.35}$$

$$\frac{dw_{ij}}{dt} \simeq \frac{w_{ji}(t+\epsilon)-w_{ji}(t)}{\epsilon} \tag{2.36}$$

を式 (2.33) と式 (2.34) に用いると、重みを求める漸化式は、

$$v_{kj}(t+\epsilon) = v_{kj}(t) - \frac{\epsilon}{|X|}\sum_{\boldsymbol{x}\in X}\delta_{2k}(\boldsymbol{x})f_j(\boldsymbol{x}) \tag{2.37}$$

$$w_{ji}(t+\epsilon) = w_{ji}(t) - \frac{\epsilon}{|X|}\sum_{\boldsymbol{x}\in X}\delta_{1j}(\boldsymbol{x})x_i \tag{2.38}$$

と書くことができる。したがって現在の重み $v_{kj}^{old}, w_{ji}^{old}$ を新しい重み $v_{kj}^{new}, w_{ji}^{new}$ に修正する規則は、

$$v_{kj}^{new} := v_{kj}^{old} - \frac{\epsilon}{|X|}\sum_{\boldsymbol{x}\in X}\delta_{2k}(\boldsymbol{x})f_j(\boldsymbol{x}) \quad (j=0\ldots m, k=1\ldots l) \tag{2.39}$$

$$w_{ji}^{new} := w_{ji}^{old} - \frac{\epsilon}{|X|}\sum_{\boldsymbol{x}\in X}\delta_{1j}(\boldsymbol{x})x_i \quad (i=0\ldots n, j=1\ldots m) \tag{2.40}$$

となる。さらにこれを行列形式で書き直すと

$$\boldsymbol{V}^{new} := \boldsymbol{V}^{old} - \frac{\epsilon}{|X|}\Delta\boldsymbol{V} \tag{2.41}$$

$$\boldsymbol{W}^{new} := \boldsymbol{W}^{old} - \frac{\epsilon}{|X|}\Delta\boldsymbol{W} \tag{2.42}$$

のようになる。ここで、

$$\Delta \boldsymbol{V} := \sum_{\boldsymbol{x} \in X} \boldsymbol{\delta}_2(\boldsymbol{x}) \boldsymbol{f}(\boldsymbol{x})^T \tag{2.43}$$

$$(\text{ただし、} \boldsymbol{\delta}_2(\boldsymbol{x}) = (\delta_{21}(\boldsymbol{x}), \delta_{22}(\boldsymbol{x}), \ldots, \delta_{2l}(\boldsymbol{x}))^T)$$

$$\Delta \boldsymbol{W} := \sum_{\boldsymbol{x} \in X} \boldsymbol{\delta}_1(\boldsymbol{x}) \boldsymbol{x}^T \tag{2.44}$$

$$(\text{ただし、} \boldsymbol{\delta}_1(\boldsymbol{x}) = (\delta_{11}(\boldsymbol{x}), \delta_{12}(\boldsymbol{x}), \ldots, \delta_{1m}(\boldsymbol{x}))^T)$$

である。式 (2.31) および 式 (2.26) を見ると、$\delta_{2k}(\boldsymbol{x})$ や $\delta_{1j}(\boldsymbol{x})$ は目標関数 $\boldsymbol{d}$ とパーセプトロン $\boldsymbol{h}$ との誤差にほぼ相当する。また、式 (2.31) から分かるように、$\delta_{1j}(\boldsymbol{x})$ を求めるのに $\delta_{2k}(\boldsymbol{x})$ の値を用いている。つまり、入力側の重み w_{ji} の修正に出力側の誤差を利用していることがわかる。このように、重みの修正の際に誤差が、パターン情報がパーセプトロンの中を流れる向きとは逆に、出力側から入力側へと伝播していくので、この種の学習のしかたを誤差逆伝播学習法 (*Back Propagation*) と呼んでいる。

実際の学習においては、式 (2.39) と 式 (2.40) を繰り返し、誤差 E がある一定の値以下に収束した時に終了するという手順となる。図 2.2 に具体的な BP 学習アルゴリズムを示す。ここでイタレーションとは、すべての学習サンプルを提示して重みを更新する動作 1 回分のことを指す。

BP_method :
/*
n, m, l : 入力層, 中間層, 出力層のユニット数
X : 学習パターン集合
$\boldsymbol{d}$: 目標関数
$\boldsymbol{W} = [w_{ji}], i = 0, \ldots, n, \quad j = 1, \ldots, m$: 入力層と中間層を接続する重み行列
$\boldsymbol{V} = [v_{kj}], j = 0, \ldots, m, \quad k = 1, \ldots, l$: 中間層と出力層を接続する重み行列
$\boldsymbol{\Delta W}, \boldsymbol{\Delta V}$: $\boldsymbol{W}, \boldsymbol{V}$ の更新量
$\boldsymbol{\sigma}$: ベクトルの各要素にシグモイド関数をかける演算
$\boldsymbol{x} = (x_0, x_1, \ldots, x_n)^T$: 入力層の値 (ただし $x_0 = 1$)
$\boldsymbol{f} = (f_0, f_1, \ldots, f_m)^T$: 中間層の値 (ただし $f_0 = 1$)
$\boldsymbol{h} = (h_1, \ldots, h_l)^T$: 出力層の値
$\boldsymbol{\delta}_2 = (\delta_{21}, \delta_{22}, \ldots, \delta_{2l})^T$, $\boldsymbol{\delta}_1 = (\delta_{11}, \delta_{12}, \ldots, \delta_{1m})^T$: 修正誤差
*/
begin
$\boldsymbol{W} := \boldsymbol{W}^{init}$; $\boldsymbol{V} := \boldsymbol{V}^{init}$; /* $\boldsymbol{W}$ と $\boldsymbol{V}$ の初期値を設定 */
$iteration$:
$\Delta \boldsymbol{V} := 0; \quad \Delta \boldsymbol{W} := 0$; /* 重み更新量を初期化 */
for $\boldsymbol{x}$ in X do begin
$\boldsymbol{f}(\boldsymbol{x}) := \boldsymbol{\sigma W x}$; $\boldsymbol{h}(\boldsymbol{x}) := \boldsymbol{\sigma V f}(\boldsymbol{x})$; /* 中間層・出力層の計算 */
/* 修正誤差を求める “$*$” はベクトルの要素同士を乗じる演算子 */

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

$$\boldsymbol{\delta}_1(\boldsymbol{x}) := \begin{pmatrix} f_1(\boldsymbol{x})(1 - f_1(\boldsymbol{x})) \\ f_2(\boldsymbol{x})(1 - f_2(\boldsymbol{x})) \\ \vdots \\ f_m(\boldsymbol{x})(1 - f_m(\boldsymbol{x})) \end{pmatrix} * (\boldsymbol{V}^T \boldsymbol{\delta}_2);$$

/* 重みの修正量 $\Delta \boldsymbol{V}, \Delta \boldsymbol{W}$ を加える */
$\Delta \boldsymbol{V} := \Delta \boldsymbol{V} + \boldsymbol{\delta_2}(\boldsymbol{x}) \boldsymbol{f}(\boldsymbol{x})^T; \quad \Delta \boldsymbol{W} := \Delta \boldsymbol{W} + \boldsymbol{\delta_1}(\boldsymbol{x}) \boldsymbol{x}^T;$
end;
/* 重みの更新 */
$\boldsymbol{V} := \boldsymbol{V} - \frac{\epsilon}{|X|} \Delta \boldsymbol{V}$; $\boldsymbol{W} := \boldsymbol{W} - \frac{\epsilon}{|X|} \Delta \boldsymbol{W}$;
/* 誤差が閾値よりも大きい場合はイタレーションを続ける */
goto $iteration$ unless $E \leq E_{threshold}$
end;

図 2.2: BP 学習アルゴリズム

第3章

学習アルゴリズムの並列化

第 1 章で述べた通り、BP アルゴリズムには数種類の並列化手法がある。本章ではそれらの具体的な手法を述べ、WS クラスタに実装して性能評価を行なう。

3.1 学習セッション並列

BP アルゴリズムの過程は、ネットワークの規模 (n, m, l)、学習サンプル X、目標関数 $\boldsymbol{d}$、重みの初期値 $\boldsymbol{V}^{init}, \boldsymbol{W}^{init}$ により規定される。これらのパラメータで規定される BP の 1 つの過程をここでは 学習セッションと呼ぶ。学習セッションは、これらのパラメータが 1 個でも異なれば一般に全く違う過程となる。したがって各学習セッション同士は互いに全く独立であるため、異なる学習セッション毎の並列化は自明である。各プロセッサでそれぞれ異なるパラメータで学習セッションを開始すればよい。本論文では 1 つの学習セッション内部の並列化について論じ、これ以降は学習セッション並列について扱わない。

3.2 データ並列

式 (2.43), 式 (2.44) を見ると、重みの更新量 $\Delta\boldsymbol{V}, \Delta\boldsymbol{W}$ は学習サンプル集合 X の中のそれぞれの $\boldsymbol{x}$ について $\Delta\boldsymbol{V}(\boldsymbol{x}) = \boldsymbol{\delta}_2(\boldsymbol{x})\boldsymbol{f}(\boldsymbol{x})^T$ と $\Delta\boldsymbol{W}(\boldsymbol{x}) = \boldsymbol{\delta}_1(\boldsymbol{x})\boldsymbol{x}^T$ を求め、それらを合計していることがわかる。

データ並列方式では、まず学習サンプル集合 X をプロセッサの台数 (ここでは N とする) だけ分割し各プロセッサに割り当てる。

$$X = (\underbrace{X^{(1)}}_{\text{プロセッサ } 1} , \underbrace{X^{(2)}}_{\text{プロセッサ } 2} , \ldots , \underbrace{X^{(N)}}_{\text{プロセッサ } N}) \tag{3.1}$$

プロセッサ $p(p = 1, \ldots, N)$ は $X^{(p)}$ についての重み更新量の部分和

$$\Delta \boldsymbol{V}(X^{(p)}) = \sum_{\boldsymbol{x} \in X^{(p)}} \Delta \boldsymbol{V}(\boldsymbol{x}) \qquad \left(\Delta \boldsymbol{V}(\boldsymbol{x}) = \boldsymbol{\delta}_2(\boldsymbol{x}) \boldsymbol{f}(\boldsymbol{x})^T\right) \tag{3.2}$$

$$\Delta \boldsymbol{W}(X^{(p)}) = \sum_{\boldsymbol{x} \in X^{(p)}} \Delta \boldsymbol{W}(\boldsymbol{x}) \qquad \left(\Delta \boldsymbol{W}(\boldsymbol{x}) = \boldsymbol{\delta}_1(\boldsymbol{x}) \boldsymbol{x}^T\right) \tag{3.3}$$

を求め、それらを合計することで全学習サンプル X についての重み更新量 $\Delta \boldsymbol{V}(X), \Delta \boldsymbol{W}(X)$ を得る。

$$\Delta \boldsymbol{V}(X) := \sum_{p=1}^{N} \Delta \boldsymbol{V}(X^{(p)}) \tag{3.4}$$

$$\Delta \boldsymbol{W}(X) := \sum_{p=1}^{N} \Delta \boldsymbol{W}(X^{(p)}) \tag{3.5}$$

図 3.1 にデータ並列における処理の流れを示す。各プロセッサはそれぞれ同一の完全な重み行列を持っており、それぞれで部分和を計算する。その後に部分和を一つのプロセッサに集め合計し完全な重み更新量 $\Delta \boldsymbol{V}(X), \Delta \boldsymbol{W}(X)$ を求め、これを各プロセッサに配り直し、最後に各プロセッサが重み更新量を自分が持つ重み行列に加えることで、1 回のイタレーションが完了する。

この方式では、部分和を集めその合計を各プロセッサに配り直す部分を除けばプロセッサ間通信はまったく不要であり、各プロセッサは独立に部分和を計算できる。部分和を計算する時間は学習サンプルの数 s に比例するので、s が大きいほど、全計算時間のうちの部分和計算に占める時間の割合が増え、プロセッサの利用率が上がる。

3.3　ノード 並列

データ並列では、分割されるのは学習サンプル集合のみであり、各プロセッサは同一の完全な重み行列を保持していた。ノード 並列方式では、重み行列を分割してその各部分行列を各プロセッサを持たせることで、学習サンプル 1 個についての重み更新量 $\Delta \boldsymbol{V}(\boldsymbol{x}), \Delta \boldsymbol{W}(\boldsymbol{x})$ の計算を並列化する方式である。

図 3.2 はプロセッサ数が 3 の場合のノード 並列方式による中間層 $\boldsymbol{f}(\boldsymbol{x})$ および出力層 $\boldsymbol{h}(\boldsymbol{x})$ の計算の様子である。まず、入力層と中間層を接続する重み行列 $\boldsymbol{W}$ は行方向にプロセッサの数に等分しプロセッサ p にその部分行列 $\boldsymbol{W}^{(p)}$ を配置しておく。一方中間層と出力層を接続する重み行列 $\boldsymbol{V}$ は 列方向 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、各プロセッサは $\boldsymbol{f}$ が分割された部分ベクトル $\boldsymbol{f}^{(1)}, \boldsymbol{f}^{(2)}, \boldsymbol{f}^{(3)}$ を計

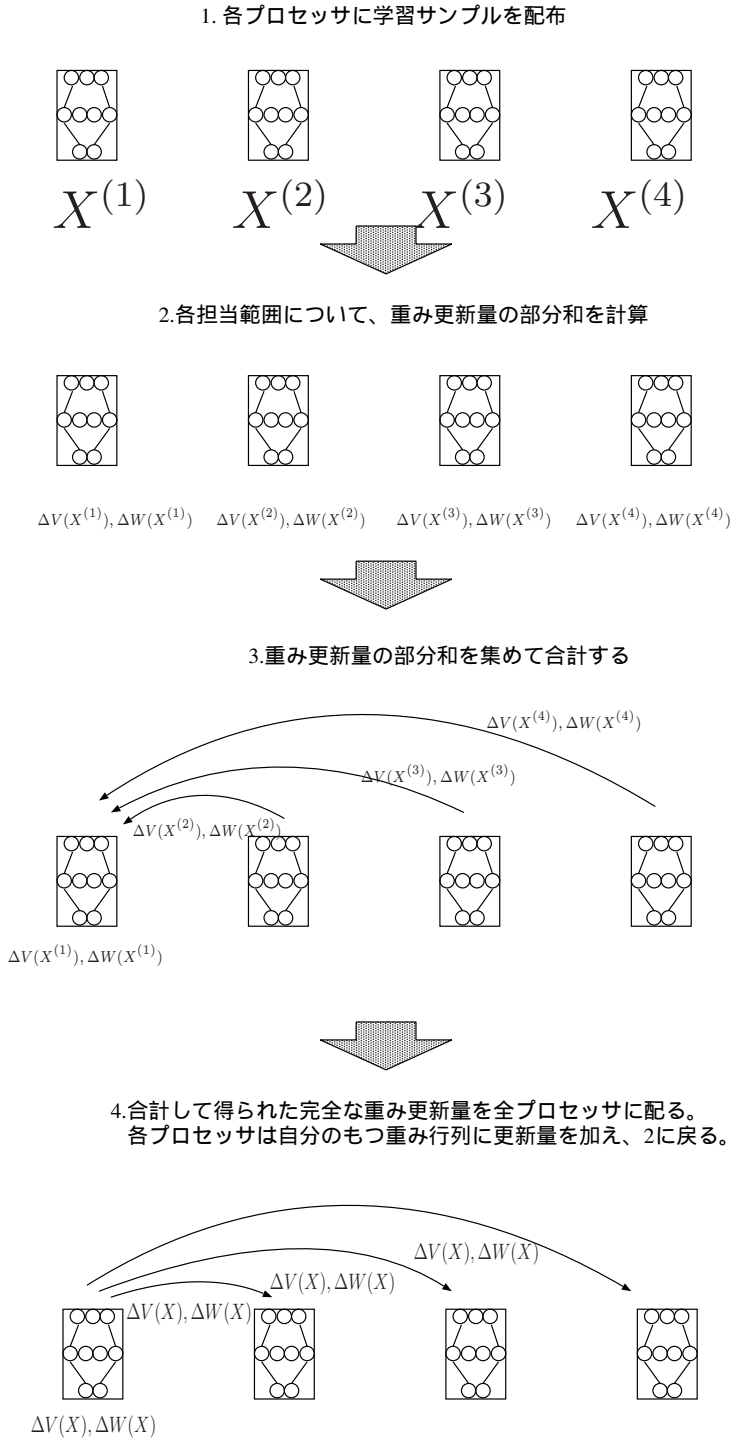

1. 各プロセッサに学習サンプルを配布
$X^{(1)}$ $X^{(2)}$ $X^{(3)}$ $X^{(4)}$
2.各担当範囲について、重み更新量の部分和を計算
$\Delta V(X^{(1)}), \Delta W(X^{(1)})$ $\Delta V(X^{(2)}), \Delta W(X^{(2)})$ $\Delta V(X^{(3)}), \Delta W(X^{(3)})$ $\Delta V(X^{(4)}), \Delta W(X^{(4)})$
3.重み更新量の部分和を集めて合計する
$\Delta V(X^{(4)}), \Delta W(X^{(4)})$
$\Delta V(X^{(3)}), \Delta W(X^{(3)})$
$\Delta V(X^{(2)}), \Delta W(X^{(2)})$
$\Delta V(X^{(1)}), \Delta W(X^{(1)})$
4.合計して得られた完全な重み更新量を全プロセッサに配る。
各プロセッサは自分のもつ重み行列に更新量を加え、2に戻る。
$\Delta V(X), \Delta W(X)$
$\Delta V(X), \Delta W(X)$
$\Delta V(X), \Delta W(X)$
$\Delta V(X), \Delta W(X)$

図 3.1: データ並列

算する。次のように各プロセッサがそれぞれ独立に計算できる。

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

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。

$$\boldsymbol{h} = \boldsymbol{\sigma}(\underbrace{\boldsymbol{V}^{(1)}\boldsymbol{f}^{(1)}}_{\text{プロセッサ } 1} + \underbrace{\boldsymbol{V}^{(2)}\boldsymbol{f}^{(2)}}_{\text{プロセッサ } 2} + \underbrace{\boldsymbol{V}^{(3)}\boldsymbol{f}^{(3)}}_{\text{プロセッサ } 3}) \tag{3.6}$$

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、これ以降示すようにプロセッサ間通信は全く不要となる。

次に $\boldsymbol{\delta_2}(\boldsymbol{x})$ の計算

$$\boldsymbol{\delta_2}(\boldsymbol{x}) = \begin{pmatrix} (h_1(\boldsymbol{x}) - d_1(\boldsymbol{x}))h_1(\boldsymbol{x})(1 - h_1(\boldsymbol{x})) \\ (h_2(\boldsymbol{x}) - d_2(\boldsymbol{x}))h_2(\boldsymbol{x})(1 - h_2(\boldsymbol{x})) \\ \vdots \\ (h_l(\boldsymbol{x}) - d_l(\boldsymbol{x}))h_l(\boldsymbol{x})(1 - h_l(\boldsymbol{x})) \end{pmatrix} \tag{3.7}$$

であるが、これは全プロセッサが完全な $\boldsymbol{h}$ を保持しているので、各プロセッサがそれぞれ式 (3.7) を計算すればよい (教師信号 $\boldsymbol{d}(\boldsymbol{x})$ も、$\boldsymbol{x}$ と同様に全プロセッサが完全な形で保持しているものとする)。さらに、図 3.3 にノード並列方式における $\boldsymbol{\delta_1}(\boldsymbol{x})$ の計算の様子を示す。ここで、$\tilde{\boldsymbol{f}} = (\tilde{f}_1(\boldsymbol{x}), \tilde{f}_2(\boldsymbol{x}), \ldots, \tilde{f}_m(\boldsymbol{x}))^T$ は、

$$\begin{pmatrix} \tilde{f}_1(\boldsymbol{x}) \\ \tilde{f}_2(\boldsymbol{x}) \\ \vdots \\ \tilde{f}_m(\boldsymbol{x}) \end{pmatrix} = \begin{pmatrix} f_1(\boldsymbol{x})(1 - f_1(\boldsymbol{x})) \\ f_2(\boldsymbol{x})(1 - f_2(\boldsymbol{x})) \\ \vdots \\ f_m(\boldsymbol{x})(1 - f_m(\boldsymbol{x})) \end{pmatrix} \tag{3.8}$$

というベクトルで、これをプロセッサ数で等分した各部分ベクトルが $\tilde{\boldsymbol{f}}^{(p)}$ であり、これは各プロセッサが独立に計算できる。“$*$” はベクトルの要素同士を乗じる演算なので、

$$\begin{aligned}\boldsymbol{\delta}_1^{(1)} &= \tilde{\boldsymbol{f}}^{(1)} * (\boldsymbol{V}^{(1)T}\boldsymbol{\delta}_2) \quad : \text{プロセッサ } 1\\ \boldsymbol{\delta}_1^{(2)} &= \tilde{\boldsymbol{f}}^{(2)} * (\boldsymbol{V}^{(2)T}\boldsymbol{\delta}_2) \quad : \text{プロセッサ } 2\\ \boldsymbol{\delta}_1^{(3)} &= \tilde{\boldsymbol{f}}^{(3)} * (\boldsymbol{V}^{(3)T}\boldsymbol{\delta}_2) \quad : \text{プロセッサ } 3\end{aligned}$$

とやはり各プロセッサがそれぞれ独立に計算できる。

最後に学習サンプル $\boldsymbol{x}$ に関する重み更新量 $\Delta\boldsymbol{V}(\boldsymbol{x}), \Delta\boldsymbol{W}(\boldsymbol{x})$ の計算の様子を図 3.4 示す。$\boldsymbol{f}$ 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。

$$
\begin{aligned}
\Delta \boldsymbol{V}^{(1)} &= \boldsymbol{\delta}_2 \boldsymbol{f}^{(1)T} : \text{プロセッサ } 1 \\
\Delta \boldsymbol{V}^{(2)} &= \boldsymbol{\delta}_2 \boldsymbol{f}^{(2)T} : \text{プロセッサ } 2 \\
\Delta \boldsymbol{V}^{(3)} &= \boldsymbol{\delta}_2 \boldsymbol{f}^{(3)T} : \text{プロセッサ } 3
\end{aligned}
$$

$$
\begin{aligned}
\Delta \boldsymbol{W}^{(1)} &= \boldsymbol{\delta}_1^{(1)} \boldsymbol{x}^T : \text{プロセッサ } 1 \\
\Delta \boldsymbol{W}^{(2)} &= \boldsymbol{\delta}_1^{(2)} \boldsymbol{x}^T : \text{プロセッサ } 2 \\
\Delta \boldsymbol{W}^{(3)} &= \boldsymbol{\delta}_1^{(3)} \boldsymbol{x}^T : \text{プロセッサ } 3
\end{aligned}
$$

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させている。このことにより $\boldsymbol{h}$ の計算以外の重み更新量の計算ではプロセッサ間の通信は必要でなくなり、各プロセッサはそれぞれ独立に重み更新量を計算可能となっている。

3.4　データセット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 (図 3.5)。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。

3.5　遅延重み更新

WS クラスタに限らず、多くの疎結合型並列計算機では、実際に計算を行なうプロセッサとプロセッサ間通信を行なうネットワークコントローラが独立しており、プロセッサ間通信を行なっている間に計算を実行することが可能である。この特性を利用して、データ並列における重み更新量の部分和の計算と、重み更新量の部分和をプロセッサ間通信により合計する部分を時間的にオーバーラップさせることで、プロセッサ間通信にかかる時間を「隠す」ことを考える。

i 回目のイタレーションにおいて計算された重み更新量を $\Delta \boldsymbol{V}_i, \Delta \boldsymbol{W}_i$ とする。通常の重み更新では、$\Delta \boldsymbol{V}_i, \Delta \boldsymbol{W}_i$ を計算した後に、それを用いて $\boldsymbol{V}, \boldsymbol{W}$ を更新する (図 3.6)。一方、

$$\left(\begin{array}{c} \boldsymbol{f}^{(1)} \\ \hline \boldsymbol{f}^{(2)} \\ \hline \boldsymbol{f}^{(3)} \end{array}\right) = \boldsymbol{\sigma} \left(\begin{array}{c} \boldsymbol{W}^{(1)} \\ \hline \boldsymbol{W}^{(2)} \\ \hline \boldsymbol{W}^{(3)} \end{array}\right) \left(\begin{array}{c} \\ \boldsymbol{x} \\ \\ \end{array}\right)$$

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

図 3.2: ノード 並列における中間層と出力層の計算

$$\left(\begin{array}{c} \boldsymbol{\delta}_1^{(1)} \\ \hline \boldsymbol{\delta}_1^{(2)} \\ \hline \boldsymbol{\delta}_1^{(3)} \end{array}\right) = \left(\begin{array}{c} \tilde{\boldsymbol{f}}^{(1)} \\ \hline \tilde{\boldsymbol{f}}^{(2)} \\ \hline \tilde{\boldsymbol{f}}^{(3)} \end{array}\right) * \left[\left(\begin{array}{c} \boldsymbol{V}^{(1)T} \\ \hline \boldsymbol{V}^{(2)T} \\ \hline \boldsymbol{V}^{(3)T} \end{array}\right) \left(\begin{array}{c} \\ \boldsymbol{\delta}_2 \\ \\ \end{array}\right) \right]$$

図 3.3: ノード 並列における $\boldsymbol{\delta}_1$ の計算

$$\left(\begin{array}{c|c|c} \Delta\boldsymbol{V}^{(1)} & \Delta\boldsymbol{V}^{(2)} & \Delta\boldsymbol{V}^{(3)} \end{array}\right) = \left(\begin{array}{c} \\ \boldsymbol{\delta_2} \\ \\ \end{array}\right) \left(\begin{array}{c|c|c} \boldsymbol{f}^{(1)T} & \boldsymbol{f}^{(2)T} & \boldsymbol{f}^{(3)T} \end{array}\right)$$

$$\left(\begin{array}{c} \Delta\boldsymbol{W}^{(1)} \\ \hline \Delta\boldsymbol{W}^{(2)} \\ \hline \Delta\boldsymbol{W}^{(3)} \end{array}\right) = \left(\begin{array}{c} \boldsymbol{\delta}_1^{(1)} \\ \hline \boldsymbol{\delta}_1^{(2)} \\ \hline \boldsymbol{\delta}_1^{(3)} \end{array}\right) \left(\begin{array}{c} \\ \boldsymbol{x}^T \\ \\ \end{array}\right)$$

図 3.4: ノード 並列における $\Delta\boldsymbol{V}(\boldsymbol{x}), \Delta\boldsymbol{W}(\boldsymbol{x})$ の計算

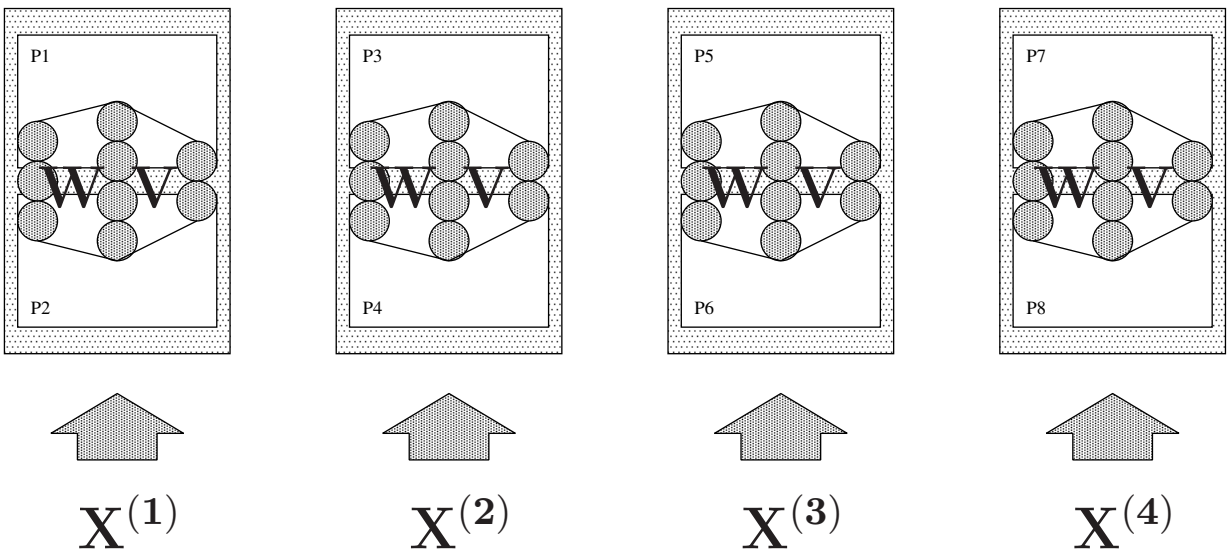


図 3.5: データセット並列とノード並列の組み合わせ

遅延重み更新では、1 つ前の重み更新量 $\Delta \boldsymbol{V}_{i-1}, \Delta \boldsymbol{W}_{i-1}$ を用いて $\boldsymbol{V}, \boldsymbol{W}$ を更新する。この遅延重み更新を用いると、図 3.7 のように、$\Delta \boldsymbol{V}_i, \Delta \boldsymbol{W}_i$ を計算している間に各プロセッサがもっている $\Delta \boldsymbol{V}_{i-1}(X^{(p)}), \Delta \boldsymbol{W}_{i-1}(X^{(p)})$ をプロセッサ間通信により合計するという処理が行なえる。$\Delta \boldsymbol{V}_i, \Delta \boldsymbol{W}_i$ の計算終了後に $\Delta \boldsymbol{V}_{i-1}, \Delta \boldsymbol{W}_{i-1}$ を用いて重みを更新する。これにより、通常の重み更新においてプロセッサ間通信のためにプロセッサが待機していた時間を重み更新量の計算にあてることができる。

遅延重み更新では、1 つ前の重み更新量を用いて重みを更新するので、第 2 章で述べた BP 法とは数学的には等価ではない。しかし、ほとんどの場合は通常の重み更新法と同様に誤差を収束させることが実験的に確かめられている。

3.6　性能評価

2 種類の並列 BP アルゴリズムを組合わせたものを WS クラスタ上に実装し、性能評価を行なった。実験に使用した WS クラスタの各 WS はそれぞれ 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[13] を使用した。実験条件は 表 3.1 に示す 3 種類を選んだ。条件 a では学習サンプル数は少なく、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。

3.6.1　遅延重み更新なしの場合

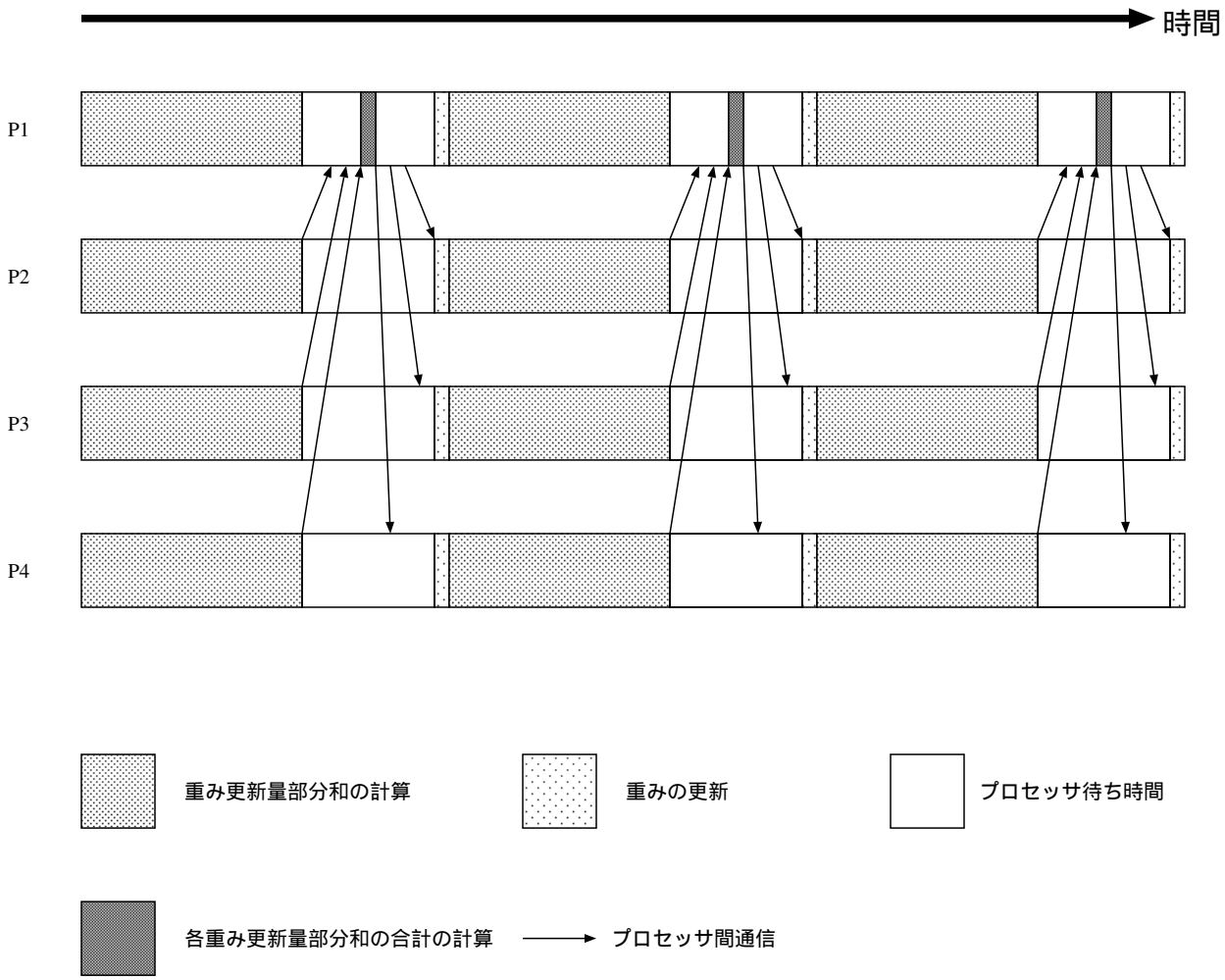
図 3.8 から図 3.10 に使用した WS の台数と各並列化法の並列度を変化させた場合の性能を示す。実験条件 a では全ての N において、ノード並列度最大 (データ並列度 1) の場合が最も高い性能を出している。条件 b では、逆にノード並列度 1 (データ並列度最大) の場合が常に最も高い性能を出している。一方、a と b の中間である条件 c では、ノード並列とデータ並列を組合わせた場合が最も高い性能を出している。

データ並列において、イタレーションの全実行時間のうち重み更新量 (の各部分和) を計算している部分では、各プロセッサが独立に実行できるため、この部分はプロセッサの個数に比例した高速化が可能である。一方、重み更新量の部分和をプロセッサ間通信により

表 3.1: 3 種類の実験条件

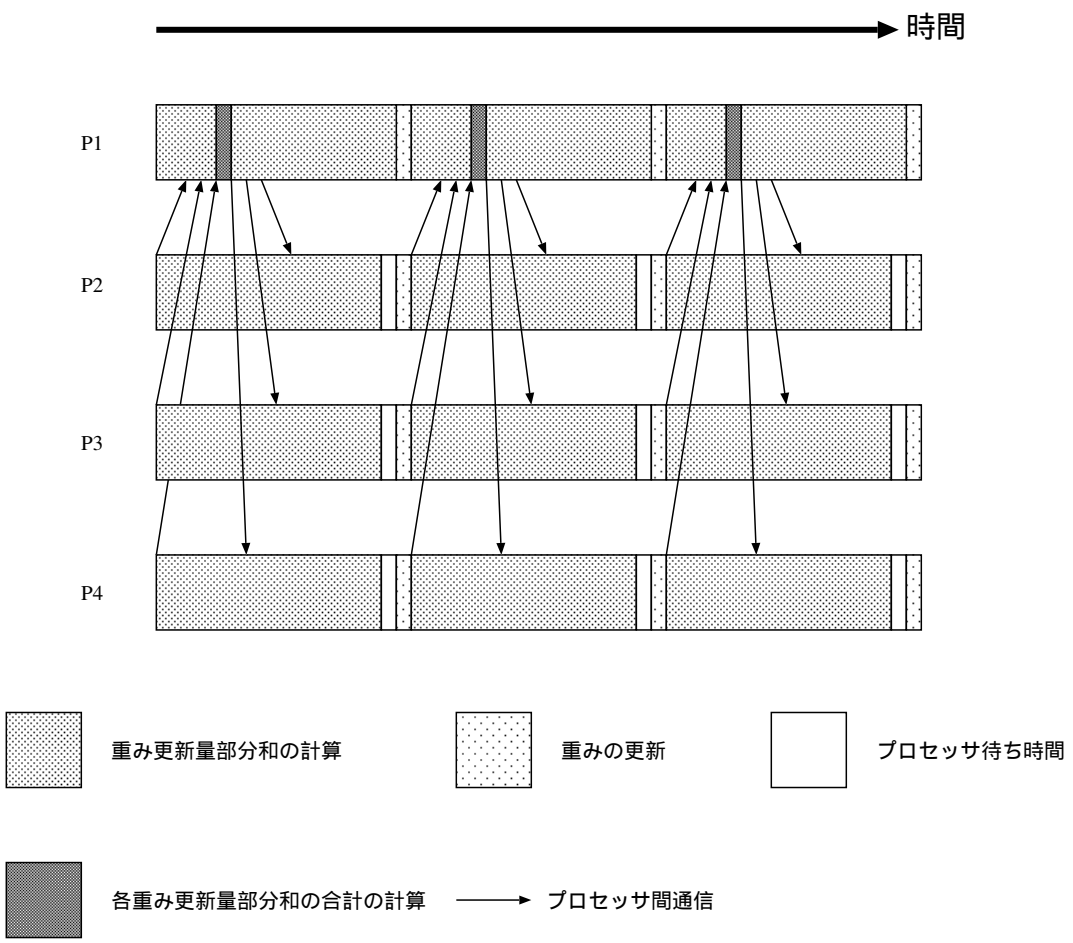
実験条件	n	m	l	s
a	50	1000	50	100
b	50	100	100	10000
c	50	100	50	1000

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



for $i = 1$ to N
  $calculate \quad \Delta \boldsymbol{V}_i, \Delta \boldsymbol{W}_i$;
  $V := V - \frac{\epsilon}{|X|} \Delta \boldsymbol{V}_i$;
  $W := W - \frac{\epsilon}{|X|} \Delta \boldsymbol{W}_i$;

図 3.6: 通常の重み更新処理



for $i = 1$ to N
  $calculate \quad \Delta \boldsymbol{V}_i, \Delta \boldsymbol{W}_i$;
  $\boldsymbol{V} := \boldsymbol{V} - \frac{\epsilon}{|X|} \Delta \boldsymbol{V}_{i-1}$;
  $\boldsymbol{W} := \boldsymbol{W} - \frac{\epsilon}{|X|} \Delta \boldsymbol{W}_{i-1}$;

図 3.7: 遅延重み更新

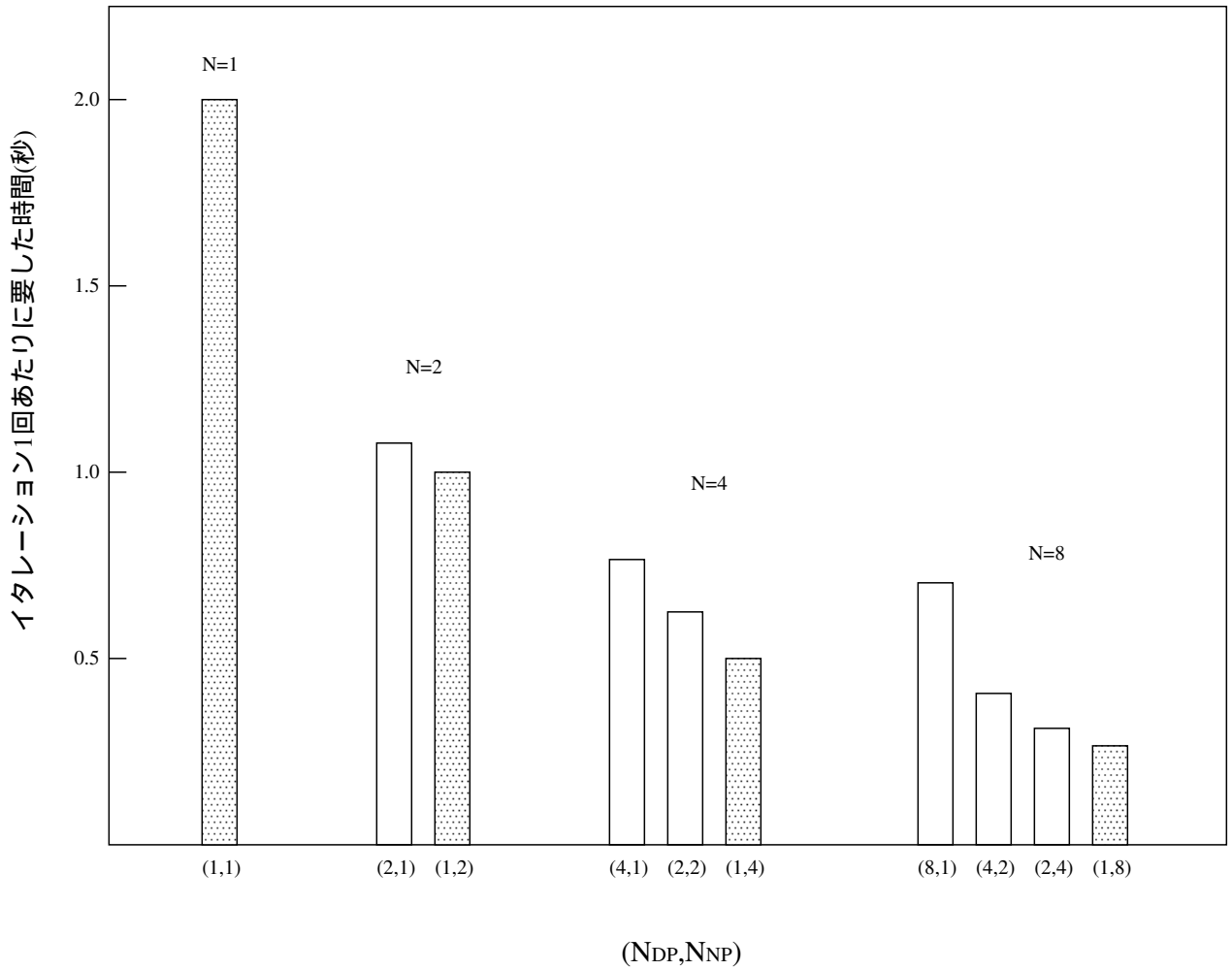


図 3.8: 実験条件 a でイタレーション 1 回あたりに要した時間 (遅延重み更新なし)

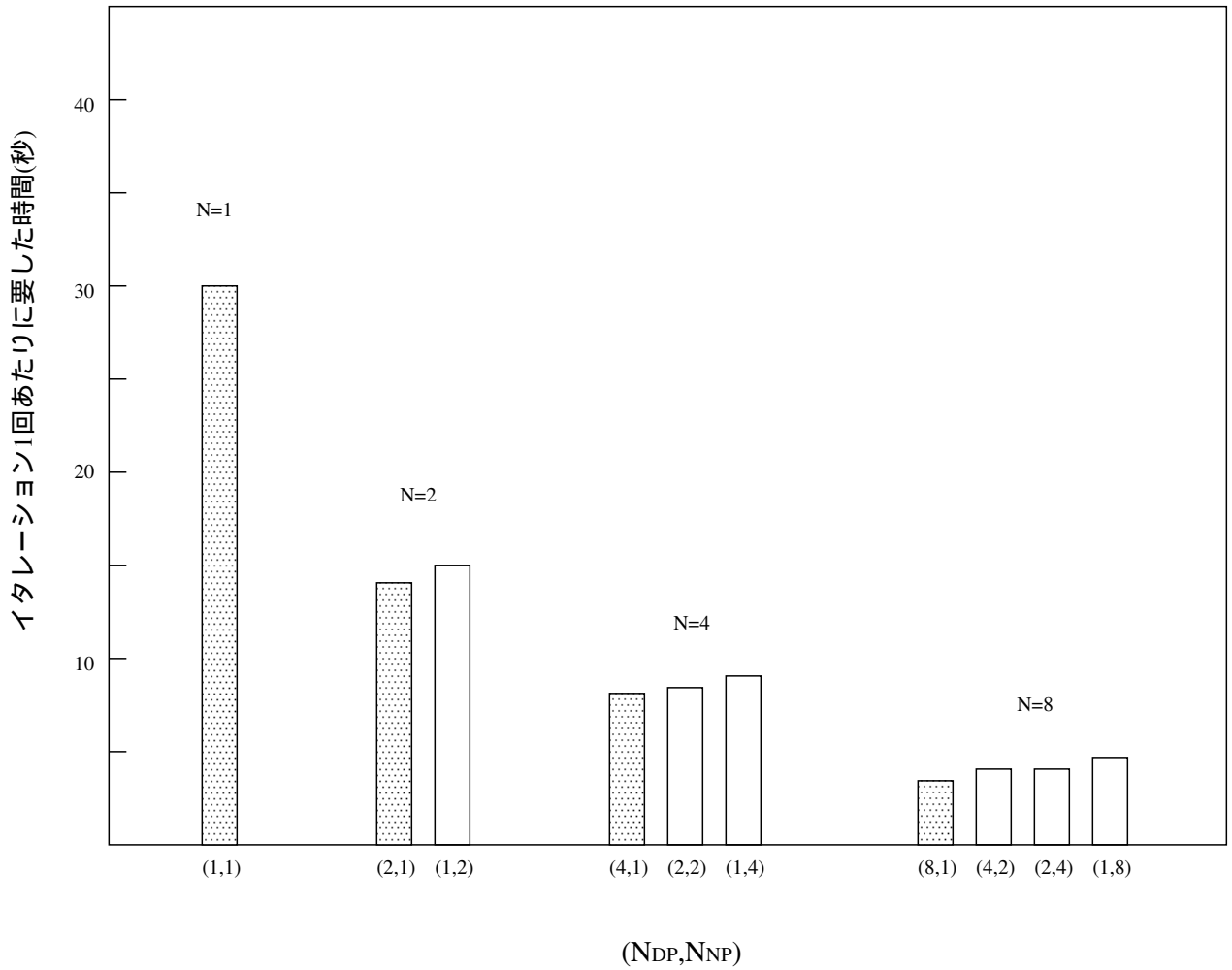


図 3.9: 実験条件 b でイタレーション 1 回あたりに要した時間 (遅延重み更新なし)

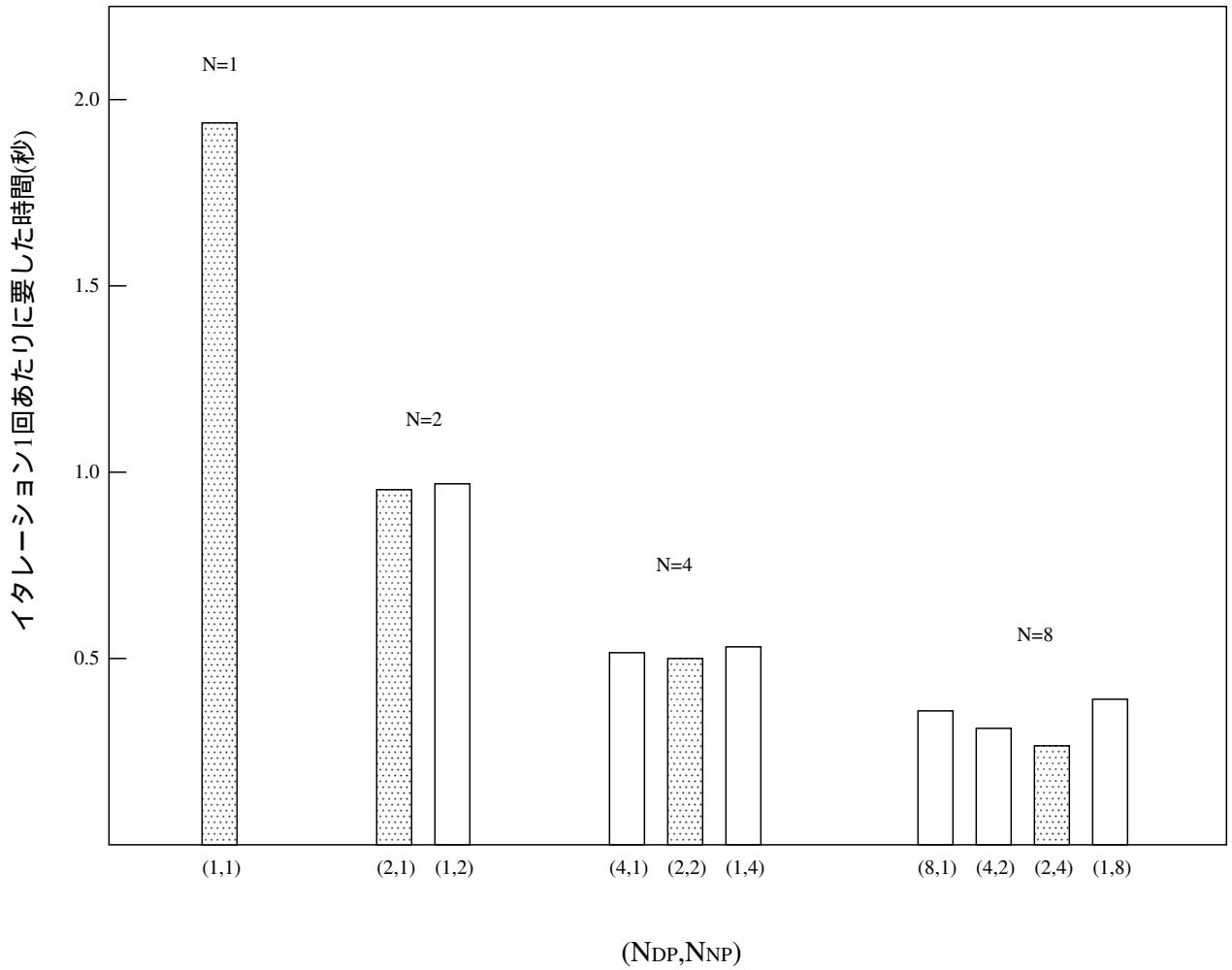


図 3.10: 実験条件 c でイタレーション 1 回あたりに要した時間 (遅延重み更新なし)

合計する部分では、重み行列の要素数やプロセッサの数が大きくなるとプロセッサ間通信の量が増大し、この部分の実行時間が大きくなる。したがって、学習サンプルの数が重み行列の要素数やプロセッサ数に対して大きい場合、イタレーションの全実行時間のうち重み更新量の計算にかかる時間の割合が大きくなり、データ並列が有利となる。しかし、学習サンプルが少ないと、逆に重み更新量の計算に要する時間の割合が小さくなり、重み更新量部分和の合計に要する時間の割合が大きくなる。したがってデータ並列が不利となってくる。一方、ノード並列では、重み行列を分割して各プロセッサに配置するので、重み部分和の合計の実行部分において、1 つのプロセッサが行なう通信量は小さくなり、結果としてこの部分の実行時間を減らすことになる。しかし、逆に学習サンプル 1 個あたりの重み更新量を計算する度にプロセッサ間通信を行なうので、これがボトルネックとなり台数効果が押えられてしまう。

このように、同じプロセッサ数でも、データ並列度とノード並列度の組合わせ (N_{DP}, N_{NP}) $(N = N_{DP}N_{NP})$ によりイタレーション 1 回あたりの計算時間は大幅に変わるということがわかる。

3.6.2　遅延重み更新ありの場合

前節と同じ条件で、遅延重み更新を行なった場合の性能を表 3.2 に示す。矢印 (→) の左側が遅延重み更新を行なわない場合、右側が行なった場合における、それぞれのイタレーション 1 回に要した時間 (秒) である。なお、矢印の左側の数値をグラフ化したものが図 3.8 から図 3.10 である。

遅延重み更新はデータ並列度が高い場合に効果があることがわかる。しかし、全般的に遅延重み更新による性能の改善はわずかであり、特にノード並列度が最大が高い性能を出せる場合はまったく効果がない。これは、ノード並列によるプロセッサ間通信と、重み更新量部分和の合計のためのプロセッサ間通信が衝突しているためと考えられる。

表 3.2: イタレーション 1 回あたりに要した時間 (遅延重み更新を行なった場合)

条件 a	N_{NP}=1	N_{NP}=2	N_{NP}=4	N_{NP}=8
$N = 1$	2.0 → 1.9	-	-	-
$N = 2$	1.1 → 1.0	1.0 → 0.96	-	-
$N = 4$	0.77 → 0.55	0.60 → 0.51	0.50 → 0.48	-
$N = 8$	0.72 → 0.50	0.43 → 0.32	0.34 → 0.29	0.27 → 0.27
条件 b	N_{NP}=1	N_{NP}=2	N_{NP}=4	N_{NP}=8
$N = 1$	29 → 29	-	-	-
$N = 2$	14 → 14	15 → 15	-	-
$N = 4$	7.8 → 7.7	7.9 → 7.9	8.3 → 8.3	-
$N = 8$	4.3 → 4.1	4.4 → 4.2	4.5 → 4.4	4.9 → 4.9
条件 c	N_{NP}=1	N_{NP}=2	N_{NP}=4	N_{NP}=8
$N = 1$	1.9 → 1.9	-	-	-
$N = 2$	0.95 → 0.94	0.96 → 0.96	-	-
$N = 4$	0.51 → 0.50	0.50 → 0.50	0.54 → 0.54	-
$N = 8$	0.29 → 0.28	0.28 → 0.28	0.27 → 0.27	0.30 → 0.30

N:使用した WS の台数,$N_{NP}(= N/N_{DP})$: ノード 並列度。矢印 (→) の左側が遅延重み更新を行なわない場合、右側が行なった場合。単位は秒。

第4章

並列度の自動選択と非均一環境への対応

第 3 章の実験結果から、使用するプロセッサの数 N 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}) が存在することが確かめられた。プロセッサ数と問題の規模が固定されていれば、様々な (N_{DP}, N_{NP}) で実際に学習を行ない、その中で最も実行時間が短くなる (N_{DP}, N_{NP}) を選択すればよい。しかし、プロセッサの数が大きくなれば (N_{DP}, N_{NP}) の組合わせ数も増大し、そのような発見的手法では最適な組合わせを高速に選択することは困難になってくる。さらに、様々な規模の問題を処理したい場合、各問題について最適な (N_{DP}, N_{NP}) が存在するため、その都度 (N_{DP}, N_{NP}) の探索が必要となる。したがって、最適な (N_{DP}, N_{NP}) を高速かつ確実に発見する手法が求められる。また、WS クラスタは、必ずしもすべての WS の性能が均一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必要がある。

4.1 並列度の自動選択

4.1.1 各並列度により実行時間が変化し得る部分

1 回分のイタレーションにおいて、各プロセッサが独立に計算を行なっている部分、つまり、

- 中間層 $\boldsymbol{f}^{(p)}$ の計算

- 出力層計算の一部。すなわち $\boldsymbol{V}^{(p)}\boldsymbol{f}^{(p)}$ の計算。
- 修正誤差 $\boldsymbol{\delta}_1^{(p)}$
- 重み更新量 $\Delta\boldsymbol{V}^{(p)}, \Delta\boldsymbol{W}^{(p)}$

に関しては、表 4.1 が示すように問題の規模とプロセッサの数 N が一定である限りプロセッサ 1 個あたりに割り当てられる計算量は (N_{DP}, N_{NP}) にかかわらず一定である。その他の部分、すなわち $\boldsymbol{h}$ のシグモイド関数の計算や、$\boldsymbol{\delta}_2$ の部分の実行時間は (N_{DP}, N_{NP}) によって変化し得るが、この部分の計算量は全体の計算量に占める割合がきわめて小さいので無視できる。したがって、(N_{DP}, N_{NP}) により実行時間が変化し得るのは主に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、ノード並列では出力層 $\boldsymbol{h}$ を計算している部分の中で、各プロセッサで計算した $\boldsymbol{V}^{(p)}\boldsymbol{f}^{(p)}$ を集めそれを合計し各プロセッサに配り直す部分である。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。

4.1.2 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、本質的にはすべて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 (図 4.1)。

ここで、使用する WS クラスタにおいて、長さ d 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 $T_{allreduce}(d, p)$ 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。

$$t_{DP} = T_{allreduce}\left(\frac{nm}{N_{NP}}, N_{DP}\right) + T_{allreduce}\left(\frac{ml}{N_{NP}}, N_{DP}\right) \tag{4.1}$$

$$t_{NP} = T_{allreduce}\left(\frac{sl}{N_{DP}}, N_{NP}\right) \tag{4.2}$$

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を行なうので (図 4.2)、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、

$$T_{allreduce}(d, p) = \alpha d\lceil\log_2 p\rceil + \beta\lceil\log_2 p\rceil \tag{4.3}$$

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

表 4.1: プロセッサ 1 個に割り当てられる計算量 (プロセッサ独立に計算できる部分)

中間層 $\boldsymbol{f}$	
加減算	mns/N
乗除算	mns/N
シグモイド関数	ms/N
出力層 $\boldsymbol{h}$	
加減算	mls/N
乗除算	mls/N
シグモイド関数	ls/N_{DP}
$\boldsymbol{\delta_2}$	
加減算	$2ls/N_{DP}$
乗除算	$2ls/N_{DP}$
シグモイド関数	0
$\boldsymbol{\delta_1}$	
加減算	$(m+1)ls/N$
乗除算	$(m+1)ls/N$
シグモイド関数	0
$\Delta \boldsymbol{V}$	
加減算	0
乗除算	mls/N
シグモイド関数	0
$\Delta \boldsymbol{W}$	
加減算	0
乗除算	nms/N
シグモイド関数	0

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

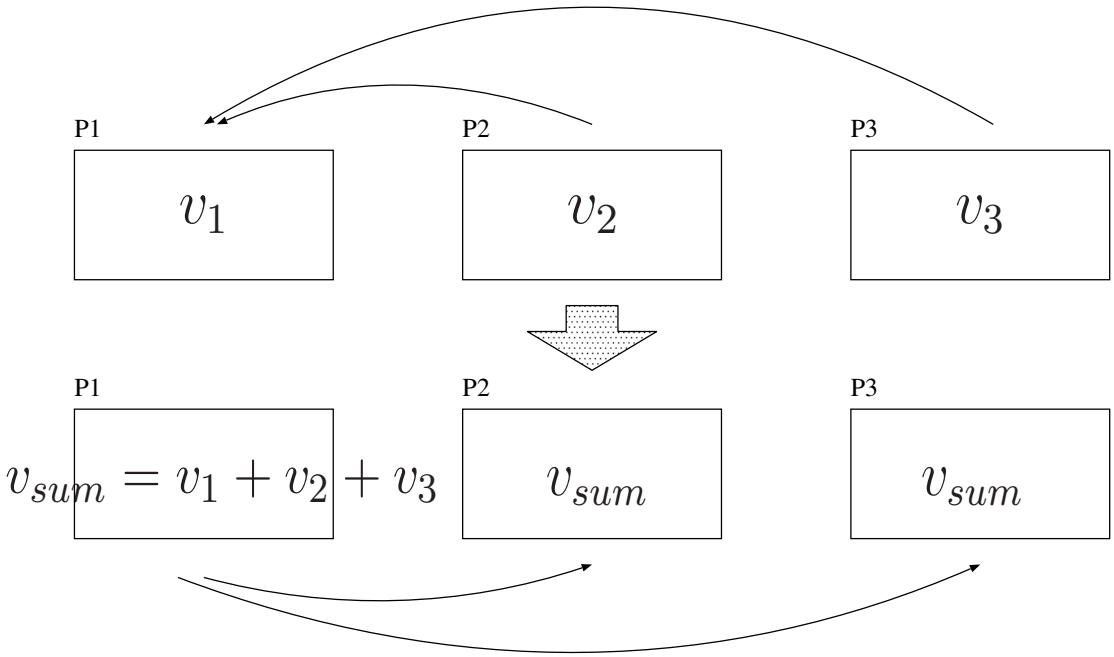


図 4.1: オールリデュース操作

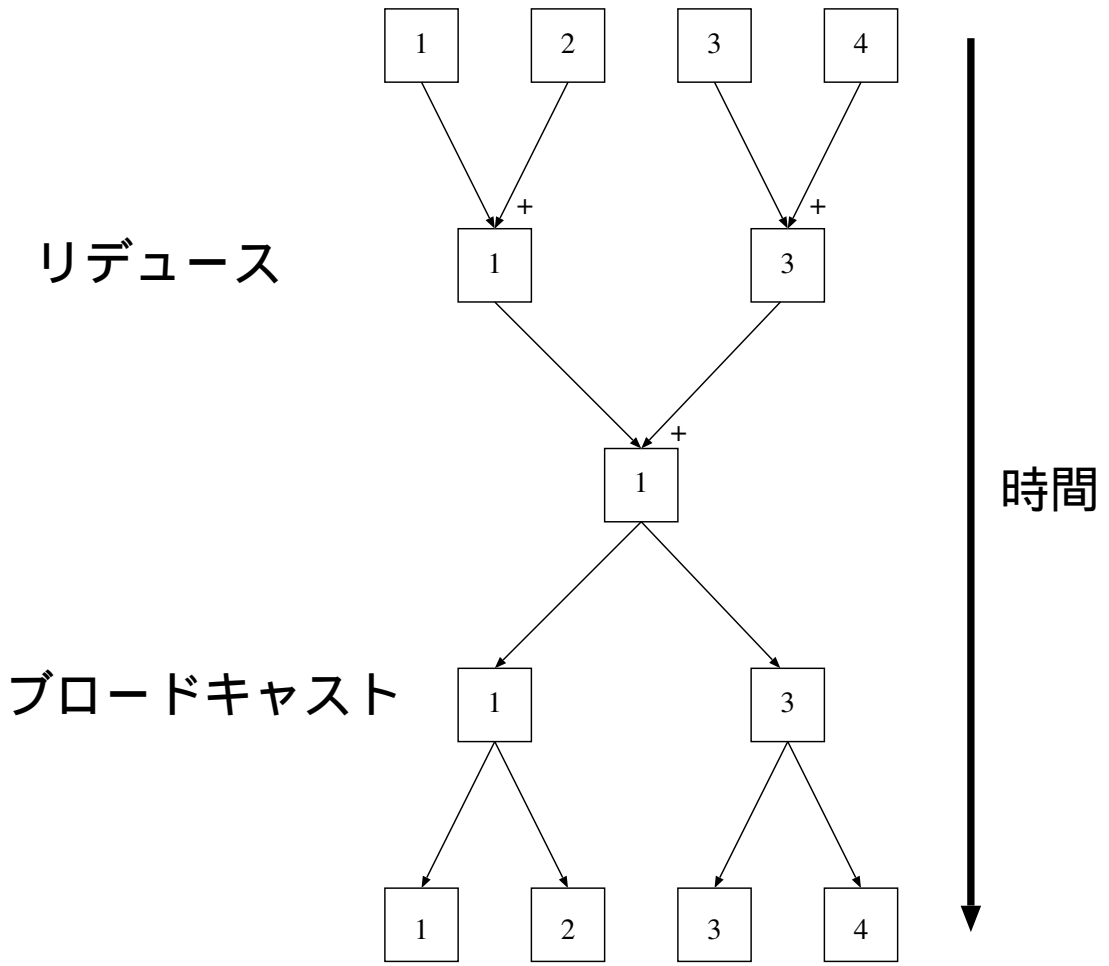


図 4.2: LAM における木構造オールリデュース操作

で、2 種類の適当な (d, 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 $T_{allreduce}(d, p)$ 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。

4.1.3 自動選択手法の評価

以上の並列度選択法を用いて、表 3.1 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 4.2 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となる場合は一致しなかった。これは 式 (4.3) のモデルが不十分なためと考えられる。

$d = 100000$ に固定し、p を 1 から 9 まで変化させて実際にオールリデュース操作を行なった場合に要した時間を図 4.3 に示す。式 (4.3) のモデルによれば、$p = 3$ の場合と $p = 4$ の場合は全く同じ実行時間になるはずであるが、$p = 4$ の場合のほうが実行時間がやや大きくなっている。同様に、$p = 5, 6, 7$ の場合もモデルでは実行時間が同一になるはずであるが、プロセッサ数が増えるにしたがって増加している。このようなモデルと実際との相違が何に起因するものかは現時点では不明である。

そこで、式 (4.3) のモデルの代りにあらかじめ様々な (d, p) でオールリデュース操作を行ないその時間を測定し、$T_{allreduce}(d, p)$ の表を作成しておく。なお、(p はともかく) あらゆる d で $T_{allreduce}(d, p)$ を測定するの不可能なので、n は適当な間隔で行ない、その間は線形補完した値を用いる。図 4.4 は p を固定して d を変化させた場合の $T_{allreduce}(d, p)$ に要した時間であるが、この結果からもわかるように $T_{allreduce}(d, p)$ は 式 (4.3) のモデル通り d に比例しており、線形補完が妥当であることがわかる。以上のようにして得られた $T_{allreduce}(d, p)$ の表を用いて、改めて並列度の選択を行なった結果が 表 4.3 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に一致する結果を得ることができる。

$T_{allreduce}(d, p)$ の表の作成には非常にコストがかかるが、使用する WS クラスタについて 1 回作ってしまえばあらゆる問題に対して適用可能である。

表 4.2: モデル $T_{allreduce}(d, p) = \alpha d\lceil \log_2 p \rceil + \beta \lceil \log_2 p \rceil$ を用いた並列度選択法で求めた N_{NP}

(○ が最適な並列度に一致した場合。× は一致しなかった場合。)

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

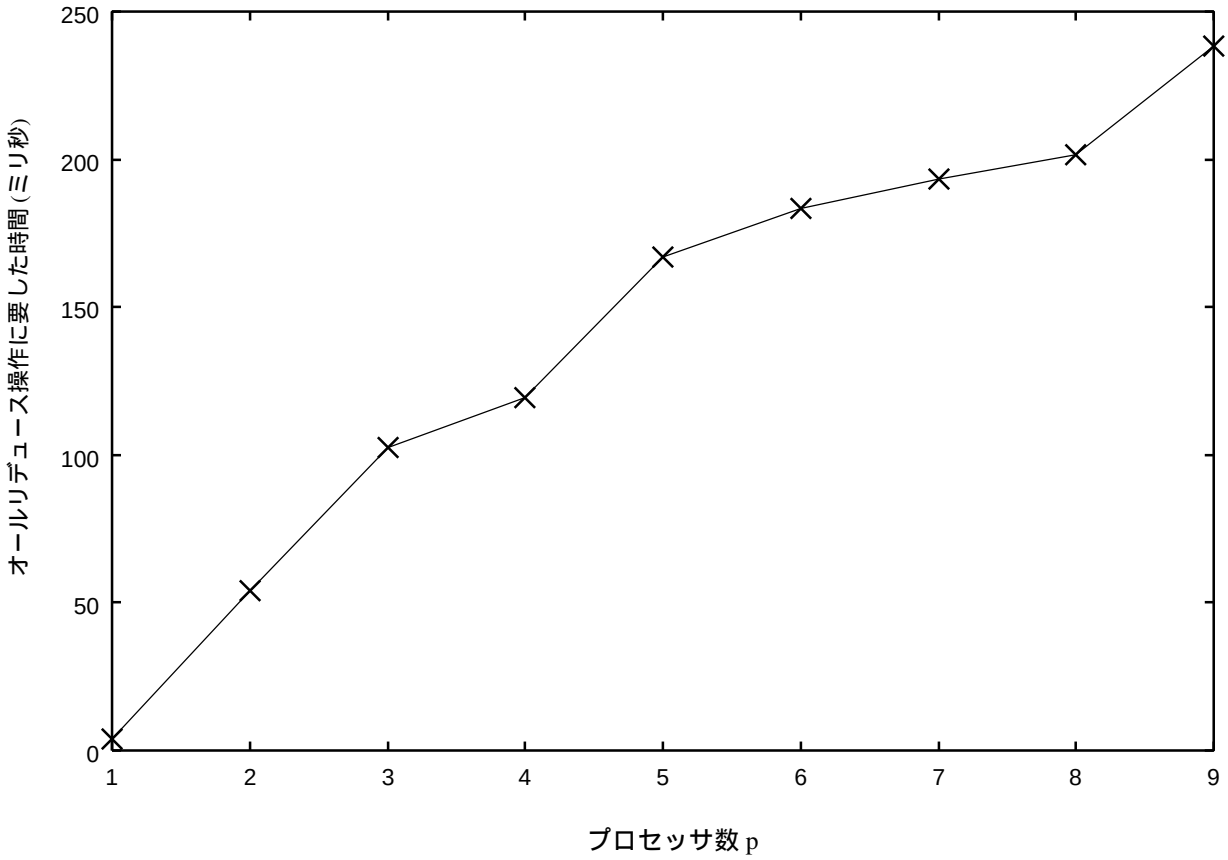


図 4.3: $d = 100000$ でオールリデュース操作に要した時間

表 4.3: $T_{allreduce}(d, p)$ 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

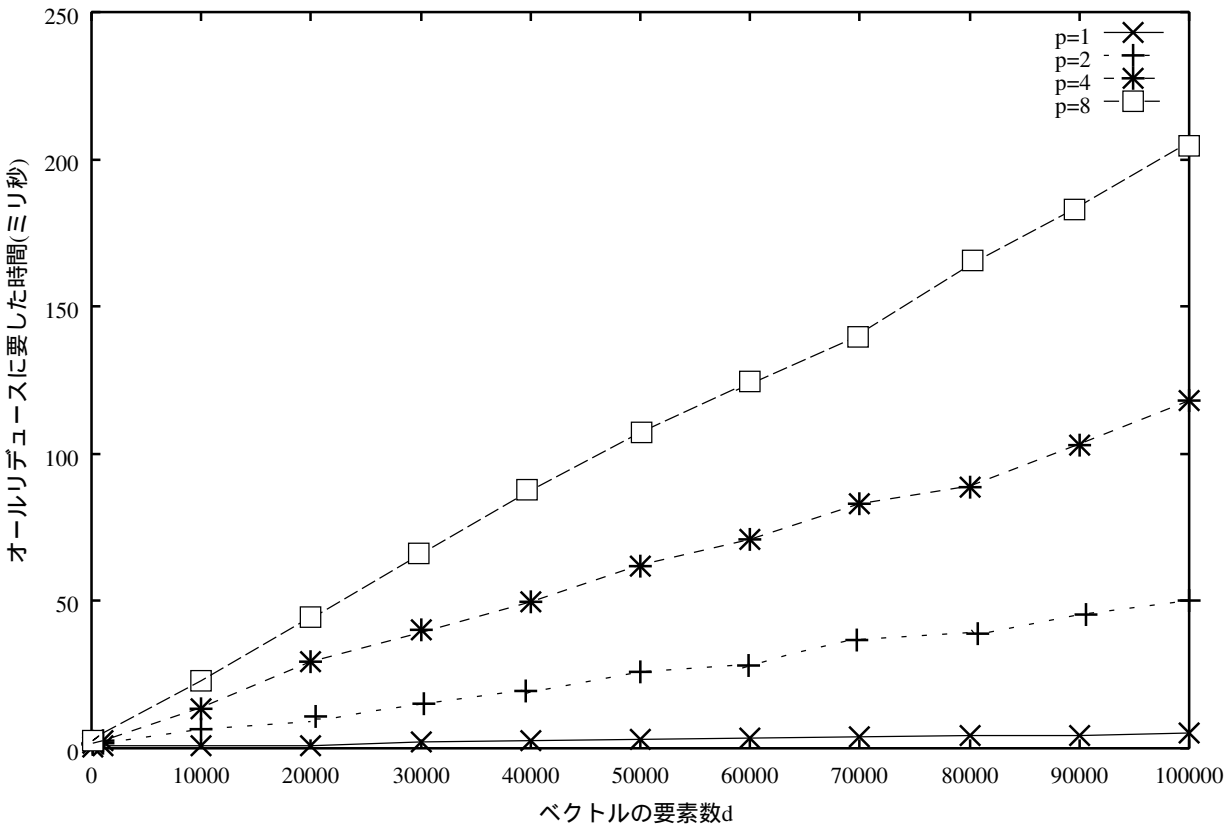


図 4.4: 様々な (d, p) においてオールリデュース操作に要した時間

4.2 非均一環境への対応

4.2.1 学習サンプルの非均一な割り当て

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 (本論文では単位時間に処理できる計算量とする) 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 3.4 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

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

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、各グループの中で最も性能の低いプロセッサの能力の割合、つまり $2.0 : 1.7 : 0.8 : 0.2$ の割合で学習サンプルを分配する (図 4.5)。

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。本手法では、ノード並列のグループ内においては各プロセッサに割り当てられる計算量は同一になるため、能力の高いプロセッサと能力の低いプロセッサを 1 つのグループ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で制限されてしまう。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 (図 4.6)

4.2.2 性能評価

以上の負荷分散手法を用いて、非均一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。

図 4.7 および図 4.8 に実験結果を示す。非均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 b は、均一環境においては 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、非均一環境でも 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 a は、

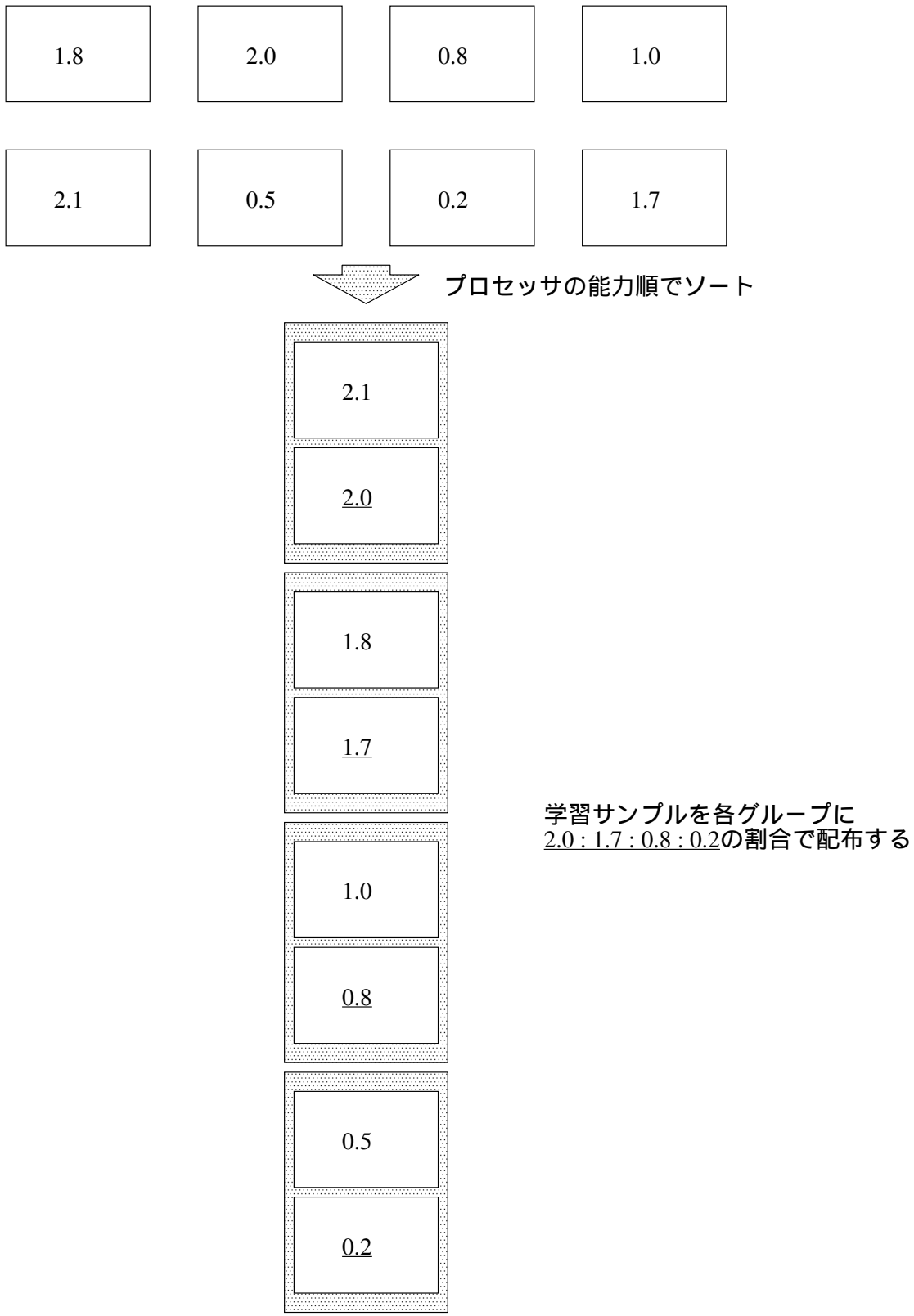


図 4.5: 学習サンプルの非均一な割り当て

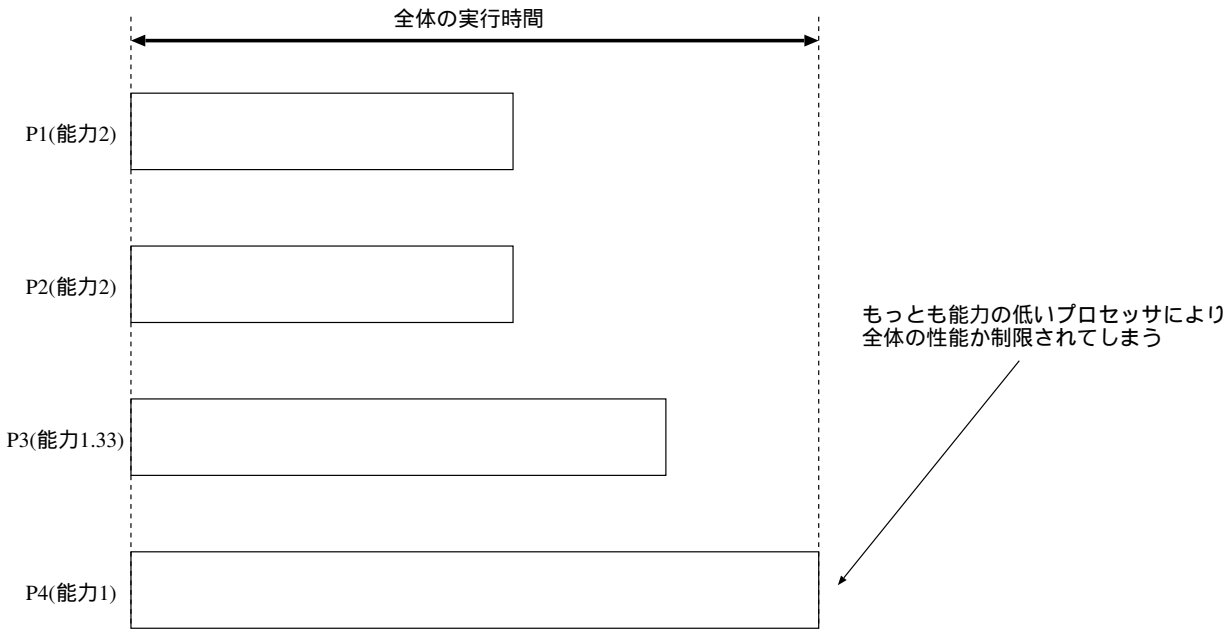


図 4.6: ノード並列においてそのグループの最も低い能力のプロセッサで全体の性能が制限されてしまう様子

均一環境においては 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、非均一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を得ている。すなわち、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。これは、本手法の非均一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。

4.2.3 非均一環境における並列度の自動選択

均一環境では、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る部分は (N_{DP}, N_{NP}) の組合わせによって実行時間はほぼ変化しないと仮定していた。しかし、非一様環境になると以下に示すように (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の実行時間 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。

イタレーション 1 回における、各プロセッサが独立に計算できる部分の全計算量を M とする。ある並列度の組合わせ (N_{DP}, N_{NP}) のもと、ノード並列グループ $i(i = 1, 2, \ldots, N_{DP})$ に割り当てられた計算量を $M^{(i)}(N_{DP}, N_{NP})$ とすれば、

$$M = \sum_{i=1}^{N_{DP}} M^{(i)}(N_{DP}, N_{NP}) \tag{4.4}$$

であるが、学習サンプルは各グループ i の最も低いプロセッサ能力 $a^{(i)}_{worst}(N_{DP}, N_{NP})$ の割合にしたがって割り当てられるので、計算量はそれに比例し、

$$\frac{M^{(1)}(N_{DP}, N_{NP})}{a^{(1)}_{worst}(N_{DP}, N_{NP})} = \frac{M^{(2)}(N_{DP}, N_{NP})}{a^{(2)}_{worst}(N_{DP}, N_{NP})} = \cdots = \frac{M^{(N_{DP})}(N_{DP}, N_{NP})}{a^{(N_{DP})}_{worst}(N_{DP}, N_{NP})} \tag{4.5}$$

となっている。

ノード並列においては、行列・ベクトルをプロセッサ数 (ノード並列度) で等分するのでグループ内部のプロセッサ $j(j = 1, 2, \ldots, N_{NP})$ に割り当てられる計算量 $M^{(i)}_j(N_{DP}, N_{NP})$ は

$$M^{(i)}_j(N_{DP}, N_{NP}) = \frac{M^{(i)}(N_{DP}, N_{NP})}{N_{NP}} \tag{4.6}$$

でグループ内の全プロセッサで等しい。グループ i 中のプロセッサ j の能力を $a^{(i)}_j$ とすれば、グループ i の実行時間 $t^{(i)}_{IC}(N_{DP}, N_{NP})$ は、

$$\begin{aligned} t^{(i)}_{IC}(N_{DP}, N_{NP}) &= \max_j \left\{ \frac{M^{(i)}_j(N_{DP}, N_{NP})}{a^{(i)}_j} \right\} \\ &= \max_j \left\{ \frac{M^{(i)}(N_{DP}, N_{NP})/N_{NP}}{a^{(i)}_j} \right\} \end{aligned} \tag{4.7}$$

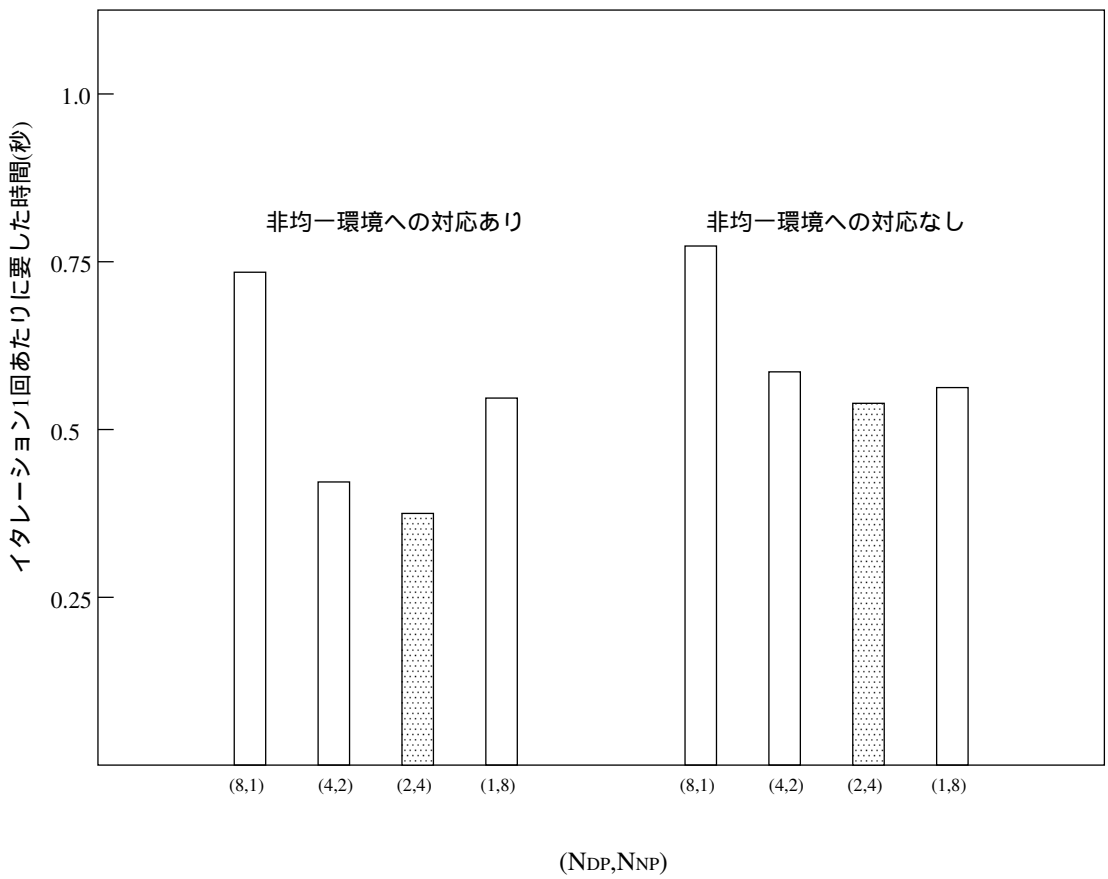


図 4.7: 非均一環境下での性能 (実験条件 a)

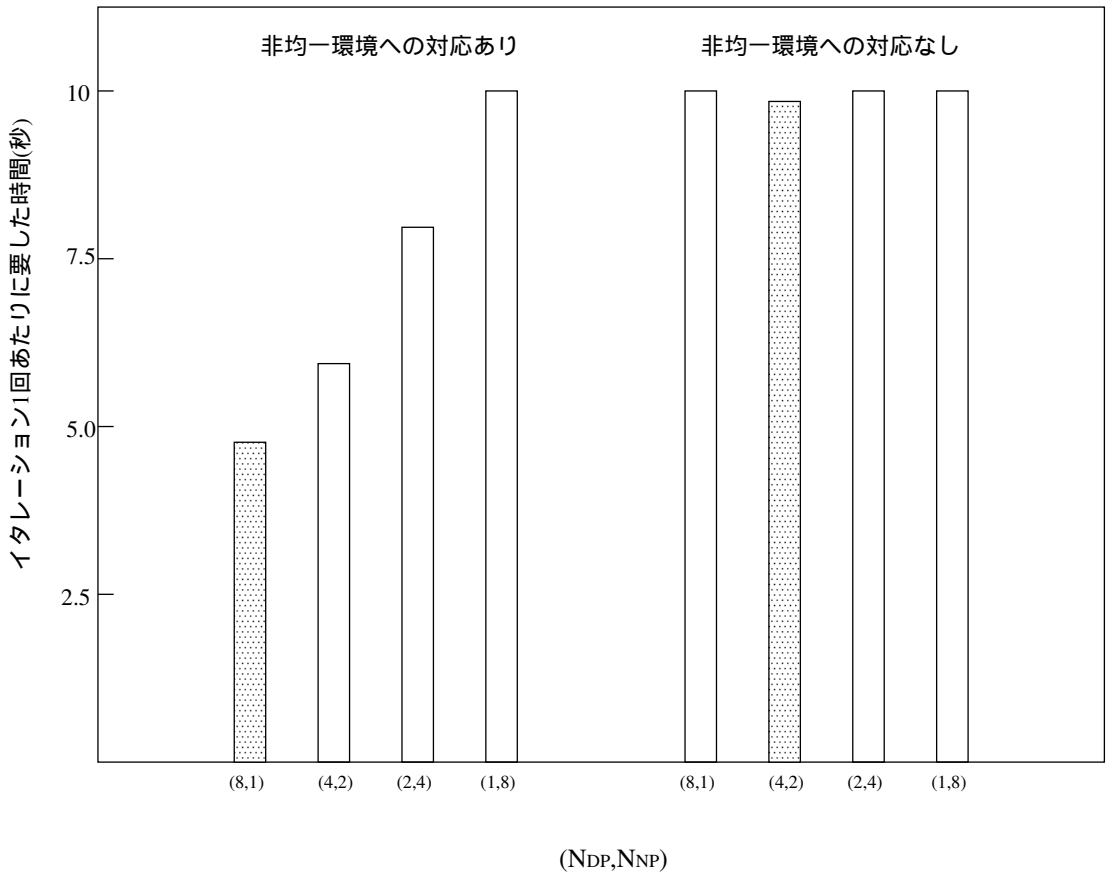


図 4.8: 非均一環境下での性能 (実験条件 b)

と書けるが、任意の j について $0 < a_{worst}^{(i)} \leq a_j^{(i)}$ であるので、

$$t_{IC}^{(i)}(N_{DP}, N_{NP}) = \frac{M^{(i)}(N_{DP}, N_{NP})}{a_{worst}^{(i)}(N_{DP}, N_{NP})N_{NP}} \tag{4.8}$$

である。したがって全グループでの実行時間 $t_{IC}(N_{DP}, N_{NP})$ は、

$$t_{IC}(N_{DP}, N_{NP}) = \min_i \left\{ t_{IC}^{(i)}(N_{DP}, N_{NP}) \right\} = \min_i \left\{ \frac{M^{(i)}(N_{DP}, N_{NP})}{a_{worst}^{(i)}(N_{DP}, N_{NP})N_{NP}} \right\} \tag{4.9}$$

であるが、式 (4.5) より i にかかわらず $\frac{M^{(i)}(N_{DP},N_{NP})}{a_{worst}^{(i)}(N_{DP},N_{NP})}$ は一定であるので、任意の i について

$$t_{IC}(N_{DP}, N_{NP}) = \frac{M^{(i)}(N_{DP}, N_{NP})}{a_{worst}^{(i)}(N_{DP}, N_{NP})N_{NP}} \tag{4.10}$$

が成り立つ。この両辺に $a_{worst}^{(i)}(N_{DP}, N_{NP})N_{NP}$ を乗じ、i を 1 から N_{DP} まで各辺を加えると

$$t_{IC}(N_{DP}, N_{NP}) \sum_{i=1}^{N_{DP}} a_{worst}^{(i)}(N_{DP}, N_{NP})N_{NP} = \sum_{i=1}^{N_{DP}} M^{(i)}(N_{DP}, N_{NP}) \tag{4.11}$$

が得られるが、式 (4.4) から

$$t_{IC}(N_{DP}, N_{NP}) = \frac{M}{\sum_{i=1}^{N_{DP}} a_{worst}^{(i)}(N_{DP}, N_{NP})N_{NP}} \tag{4.12}$$

となる。

$$a_{total}(N_{DP}, N_{NP}) = \sum_{i=1}^{N_{DP}} a_{worst}^{(i)}(N_{DP}, N_{NP})N_{NP} \tag{4.13}$$

と書くと、

$$t_{IC}(N_{DP}, N_{NP}) = \frac{M}{a_{total}(N_{DP}, N_{NP})} \tag{4.14}$$

と書けるが、ここで $a_{total}(N_{DP}, N_{NP})$ は、並列度 N_{DP}, N_{NP} において全プロセッサが発揮できる総合能力とみることができる。

この $a_{total}(N_{DP}, N_{NP})$ 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}) における総合能力を $a_{total}(N'_{DP}, N'_{NP})$ と書くと、やはり

$$t_{IC}(N'_{DP}, N'_{NP}) = \frac{M}{a_{total}(N'_{DP}, N'_{NP})} \tag{4.15}$$

であるが、式 (4.14) と 式 (4.15) から M を消去すると

$$t_{IC}(N'_{DP}, N'_{NP}) = \frac{a_{total}(N_{DP}, N_{NP})}{a_{total}(N'_{DP}, N'_{NP})} t_{IC}(N_{DP}, N_{NP}) \quad (4.16)$$

を得る。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、式 (4.16) 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。

以上の手法を用いて、4.2.2 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 4.4)。最適な (N_{DP}, N_{NP}) が求まっていることがわかる。

表 4.4: 非均一環境を考慮した並列度選択法の結果。○ が最適な並列度に一致した場合。× は一致しなかった場合。

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

第5章

結論

5.1 本研究の成果

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BP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した。特にノード並列方式では、全プロセッサに完全な出力層ベクトルをもたせるというデータの流れの工夫により、従来手法よりもプロセッサ間通信を大幅に減らしている。

さらに、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手法を提案した。$T_{allreduce}(d,p)$ を数式モデル化する手法では完全には正しく選択できなかったが、事前に様々な (d,p) で実際にオールリデュース操作に要した時間を測定して $T_{allreduce}(d,p)$ の表を作ることで、正しく選択できるようになった。この手法でも十分実用的である。

また、非均一な並列計算機環境を考慮した負荷分散および並列度選択法を提案した。非均一環境を考慮しない場合にくらべ、性能の悪化を押えることが可能である。

5.2 今後の課題

より高速な並列度自動選択のために、$T_{allreduce}(d,p)$ のより正確な数式モデルを作ることが必要である。ネットワークのスループットに関する項と遅延に関する項以外の項が存在することが推察されるが、具体的な推定には至っていない。

また、本手法における、非均一環境を考慮した負荷分散および並列度の自動選択法は、データ並列のみしか考慮していない。すなわち、ノード並列のグループ内においては各プロセッサの能力に関わらず割り当てられる計算量は同一であり、ノード並列の性能はそのグループの中で最も低い能力のプロセッサで制限されてしまう。したがって、ノード並列

グループ内部においても各プロセッサの能力に応じて重み行列を担当する割合を変化させるなどの方法により、ノード並列も考慮した非均一負荷分散を行なえばさらにプロセッサの利用率を向上させることが可能である。しかし、この方式では必然的にプロセッサ 1 個あたりの通信量がプロセッサによって変わることになるので、特に並列度の自動選択問題はきわめて複雑になると推測される。

謝辞

本研究を進めるにあたり、全般的なご指導を賜わりました東北大学大学院工学研究科阿曽弘具教授並びに大町真一郎助教授には心より感謝申し上げます。

また、本論文をまとめる際に、貴重なご意見を頂きました東北大学大学院工学研究科阿部健一教授並びに同情報科学研究科亀山充隆教授に深く感謝いたします。

本研究全般に渡り、親身に御指導と御助言を賜わった東北大学大学院工学研究科 菅谷至寛助手をはじめとする阿曽研究室並列処理グループの方々に心より感謝申し上げます。

参考文献

[1] A.Singer: Implementation of Artificial Neural Network on the Connection Machine, *Parallel Computing*, 14:305-315, Summer 1990.

[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] T.Nordstrom *et al*: Using and Designing Massivelely Parallel Computers for Artificial Neural Networks, *Journal of Parallel and Distributed Computing*, 14(3):260-285, March 1992.

[4] Xiru Zhang: The packpropagation algolithm on grid and hypercube architectures, *Parallel Computing*, 14:317-327, Summer 1990.

[5] 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.

[6] 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.

[7] F. Rosenblatt: Principle of Neurodynamics, *Spartan* , 1962.

[8] M.L.Minsky and S.A. Papert: Perceptrons, *The MIT Press*, 1969.

[9] D.E.Rumelhart : Learning Representation by Back-Propagating Errors, *Nature*, 323, pp.533-536, 1986.

[10] 高田直幸, 阿曽弘具, 三宅章吾: 多層パーセプトロンの幾何学的性質に基づく構造最適化, ニューロコンピューティング研究会, 2000.

[11] MPI Forum: Message Passing Interface (MPI) Forum Home Page, *http://www.mpi-forum.org*

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

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

[14] 渡辺勝正: 並列処理概論, コロナ社, 1991.

[15] 奥川峻史: 並列処理アーキテクチャ, コロナ社, 1991.

研究発表等

非一様な計算機環境おけるニューラルネットワークの学習,
　情報処理学会ハイパフォーマンス・アーキテクチャ研究会, 2002 年 3 月 (予定)