ハイパフォーマンス
コンピューティング 67－5
（1997. 8. 19）

# 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

吉田 英嗣　安永 守利

筑波大学 電子・情報工学系

ニューラルネットワーク学習の1つである Back-propagation(BP) 学習法は, 様々な分野で有力である半面, その学習計算には膨大な時間が必要である. 本研究では, 3次元 HXB を持つ超並列計算機 CP-PACS による BP 学習の高速化を図る. 実装には, パターンデータパラレルとニューロンパラレル, 更にそれらを組み合わせた並列化方法を用いた. 評価には, BP のベンチマークとして広く利用されている NETtalk(英単語を発音記号に変換する BP アプリケーション) を用いた.

評価の結果, 2つの並列化を組み合わせた方法では, 256 プロセッサで毎秒約 10 億回の結合重み更新が可能であることが分かった. 更に, パターンデータパラレルとニューロンパラレルそれぞれの結果から, それら 2 つの最適な組み合わせを見つけるための方法を提案し, 実験によりその有効性を示した.

# Neural network computation on a massively parallel computer: CP-PACS

EIJI YOSHIDA　MORITOSHI YASUNAGA

Institute of Information Sciences and Electronics,University of Tsukuba

Back-propagation(BP), which is one of artificial neural network algorithms, requires very long training time, though it has high performance for various applications. In this research, to overcome this difficulty, we implemented the BP algorithm on a massively parallel computer "CP-PACS" using pattern-data-parallelism, neuron-parallelism, and their combination. To evaluate this peformance, we used NETtalk, which is one of BP applications, as a BP benchmark.

We obtained high performance with the combination of the above two parallelisms, and achieved 10 billion weight-updates per second using 256 processors. We proposed a method to find the optimal combination of two parallelisms, and showed its efficiency experimentally.

## 1 はじめに

ニューラルネットワークは人間の脳をモデル化した情報処理方式であり, 近年, 様々な分野で応用されるようになっている. なかでも, バックプロパゲーション (Back-Propagation, 以下 BP) アルゴリズムは, 特に有力なもので, 文字認識や株価予測などへの応用にも利用されている. しかし, その学習計算には膨大な時間がかかるという問題がある. これを解決するために, ニューラルネットワークの動作をそのままハードウェア化したニューロコンピュータを用いて高速化する方法, 汎用の並列計算機上でニューラルネットワークをエミュレートすることにより高速化する方法が考えられている. 本研究では, 後者について超並列計算機 CP-PACS (Computational Physics by Parallel Array Computer System) に適した BP 計算の並列化法を検討し, 実際にプログラムを作成して実験を行う. 更にその結果から, BP 計算に対する CP-PACS の評価と, 最適な並列化方法について検討する.

## 2 CP-PACS

CP-PACS[2][3] は 2048 台の PU を持つ MIMD 方式の分散メモリ型並列計算機である. プロセッサには Hewlett Packard 社 PA-RISC1.1 アーキテクチャを基本とし, PVP-SW(Pseudo Vector Processor based on Slide Windowed registers) 機構を付加したものを使用している. クロック周波数は 150MHz, 2 命令同時発行可能スーパスカラ方式を採用している. 主記憶容量は 64MB で, キャッシュメモリ容量は 1 次データ・命令用共に 16KB, 2 次データ・命令用共に 512KB である.

PU 間相互結合網には, 3次元ハイパクロスバネットワーク (Hyper-Crossbar Network:HXB) を

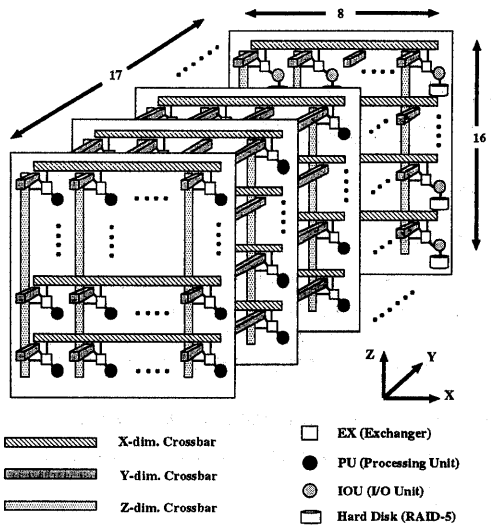


図 1: CP-PACS の構成図

採用しており, HXB と PU は NIA(Network Interface Adapter) によって接続されている. 3 次元 HXB は 3 次元直行座標上の各格子点に PU を並べ, その間をクロスバスイッチ (XB) で結合したネットワークである. XB によって結合されない PU 間での転送は, エクスチェンジャー (EX) と呼ばれる小規模のクロスバスイッチにより複数次元の XB を経由して行われる. また, CP-PACS ではメッセージ転送に wormhole ルーティング, 経路の選択は固定ルーティングを採用している. さらに,PU 間での通信はリモート DMA 転送と呼ばれる PU の NIA がユーザメモリ空間のデータを直接 DMA アクセスして送受信を行う方式になっている. 転送時に OS 領域のバッファを使用しないため, ネットワークのスループットを格段に上げることができる.

## 3 Back-Propagation:BP

### 3.1 Back-Propagation アルゴリズム

図 2 (右) に示すような階層型 2 層ネットワークを考えたとき, ある入力パターンから望まれる出力が得られるように, ニューロン間結合重みを調整するアルゴリズムを, Back-propagation 学習法 [1] という. 以下に,BP のアルゴリズムについて簡単に説明する (詳細については [1] を参照のこと).

BP アルゴリズムは forward 計算と backward 計算, modify 計算で構成される. まず forward 計算では, $\mu$ 番目の入力 $\xi_k^\mu$ が与えられたときの中間

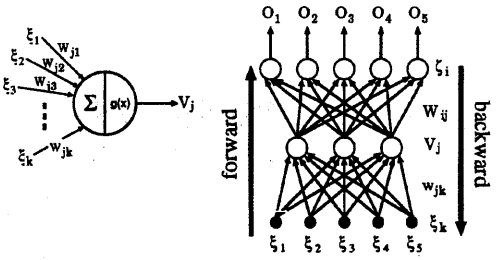


図 2: ニューロンモデル (左) と階層型 2 層ネットワーク (右)

層素子の出力 $V_j^\mu$, 出力層素子の出力 $O_i^\mu$ を以下の (1)(2) の順に計算する.

$$V_j^\mu = g(h_j^\mu) = g\left(\sum_k w_{jk}\xi_k^\mu\right) \tag{1}$$

$$O_i^\mu = g(h_i^\mu) = g\left(\sum_j W_{ij}V_j^\mu\right) \tag{2}$$

次の backward 計算では, forward 計算で得られた出力 $O_i^\mu$ と望まれる出力 $\zeta_k^\mu$ の誤差を求め, 誤差を小さくする結合重み変更量 $\Delta W_{ij}$,$\Delta w_{jk}$ を (3)(4) に従って計算する.

$$\Delta W_{ij} = -\frac{\partial E}{\partial W_{ij}} = \sum_\mu \delta_i^\mu V_j^\mu \tag{3}$$

$$\delta_i^\mu = g'(h_i^\mu)[\zeta_i^\mu - O_i^\mu]$$

$$\Delta w_{jk} = -\frac{\partial E}{\partial w_{jk}} = \sum_\mu \delta_j^\mu \xi_k^\mu \tag{4}$$

$$\delta_j^\mu = g'(h_j^\mu)\sum_i W_{ij}\delta_i^\mu$$

modify 計算では, ニューロン間結合重み $W_{ij}$,$w_{jk}$ を (5)(6) に従って変更する. ここでの $\eta(0 < \eta < 1)$ は学習係数と呼ばれ, 結合重み変更量を調整する定数である.

$$w_{jk}^{\text{new}} = w_{jk}^{\text{old}} + \eta\Delta w_{jk} \tag{5}$$

$$W_{ij}^{\text{new}} = W_{ij}^{\text{old}} + \eta\Delta W_{ij} \tag{6}$$

上記計算は更に, 逐次修正法と一括修正法の 2 種類の方法に分けられる. 逐次修正法は, 各学習パターンデータ毎に forward 計算, backward 計算, modify 計算を順に行ない, それを繰り返す方法である. 一括修正法は, 各学習パターンデータの forward 計算と backward 計算の後毎回 modify 計算を行わず, 全ての学習パターンデータについて forward 計算, backward 計算を行った後に, modify 計算を行う方法である. 本研究ではこれら 2 つの方法について CP-PACS での高速化を図る.

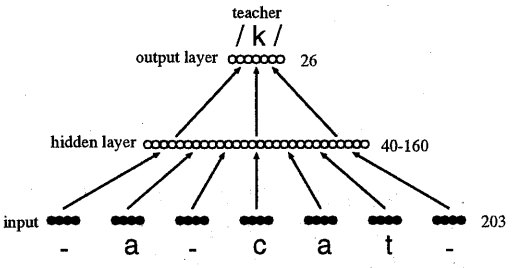


図 3: NETtalk ネットワークの構成

## 3.2 NETtalk

NETtalk は, ある英単語入力に対して正しい音素 (発音記号) に出力するようなネットワークを, BP により学習させるニューラルネットワークの 1 つのアプリケーションである. 図3のように,NETtalk ネットワークは入力層素子数 203 個, 中間層素子数 40 〜 160 個, 出力層素子数 26 個で構成される. 英単語を 1 文字ずつ左へずらして作る 7 文字を 1 パターンとし, それをネットワークへの入力とする. そして, ネットワークには真中の文字に対応する発音記号を出力するように学習させる.

T. J. Sejnowski 等によると, 1024 語の学習を行った結果, 10 回学習後に十分理解可能な発音となり, 50 回学習後には 95 %正確な発音になり, 面白いことに学習の途中の発音はまるで子供が話すような発音になっていたという. しかしながら, 20008 語の学習に当時のミニコンで 1,2 週間の時間を要したと報告されている [4]. NETtalk は, このようにニューラルネットワークの能力の高さを示す応用がある反面, 非常に長い学習時間が必要であることから, 1980 年代後半よりニューロコンピュータやニューロシミュレータの事実上のベンチマークとされてきた. 近年の計算機の性能向上により学習時間が短くなったとはいえ, いまだ数時間の学習時間が必要であり, 並列計算機による高速化が期待される.

# 4 BP 計算の並列化

## 4.1 パターンデータパラレル

パターンデータパラレル (以下 PDP) は図 4に表すように, 各 PU それぞれに完全な 2 層ネットワークを持たせ, 全学習パターンデータを PU 数で分割したものを各 PU に割り当てる. 例えば図 4では, 40 個の学習パターンデータを 4 つの PU にそれぞれ 10 個ずつ割り当てている. それぞれの PU は割り当てられたパターンすべてについて forward 計算, backward 計算を行ない, 結合重み更新量を求めその和をとる. modify 計算では, 他の PU で求めた結合重み更新量が必要となるため, ここで結合重み更新量の PU 間通信が必要となる. 通信手段は図 5に示す Butterfly Summation 転送を行い, 転送後全 PU が同じ重み更新量を保持できるようにする. 使用する PU 数を $P$ とすると, Butterfly Summation 転送では総転送回数は $\log_2 P$ となる (図では $P = 8$).

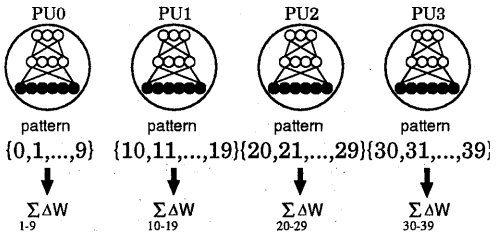


図 4: パターンデータパラレル (PDP)

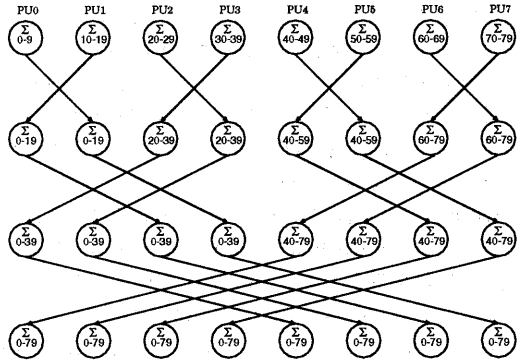


図 5: Butterfly Summation 転送

PDP は, 1 パターンデータ毎に modify 計算をする逐次修正法には利用できないため, 本研究では一括修正法にのみ適用する. PDP は, 与えられるパターンデータ数が多くなればなるほど通信のオーバヘッドの割合が小さくなり有利になる.

## 4.2 ニューロンパラレル

ニューロンパラレル (NP) では図 6に表すように, ネットワークを縦に PU 数で分割し, それぞれの PU が割り当てられた素子の計算をする。図では, PU0 に中間層素子 0-1, 出力層素子 0, PU1 に中間層素子

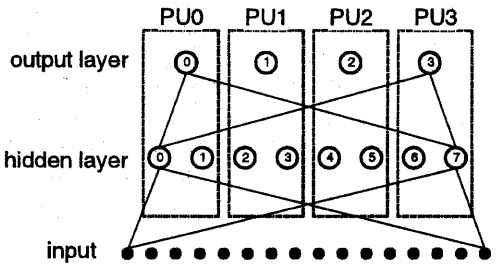


図 6: ニューロンパラレル (NP)

2-3, 出力層素子 1 が割り当てられている. 学習パターンデータに関してはすべての PU が同じものを初期値として持つ. このとき, 各 PU が行うべき計算はそれぞれに割り当てられた中間層・出力層素子の forward 計算と backward 計算, それら素子につながる結合重みの modify 計算である. また, forward 計算時には中間層素子の出力を他の PU に送信し, backward 計算時には出力層素子の誤差を他の PU に送信する必要がある. このどちらの送信もすべての PU から他のすべての PU への転送, 全対全 broadcast となる. さらに,PDP でのように和を計算をしながらの転送ではなく, すべての値を集める転送が必要となる. したがって, 全 PU からの $P$ 回のマルチキャストや $P-1$ 回のリレー転送のなどが考えられるが, 広域の転送でも高速な転送が望める CP-PACS では, PDP と同様に転送回数が $\log_2 P$ で済む Butterfly 転送方式をとる. PDP での Butterfly Summation 転送とは, 転送時のステップが進むごとにデータ転送量が増加していくという点で異なることに注意されたい.

以上の 2 つの並列化法を組み合わせたものもある. つまり, 8 台の PU を用いる場合, それを 2 個ずつの 4 グループに分け, PU2 台で NP を行ない, その 4 グループで PDP を行なうというような方法である. この方法を PDP+NP と記述する. これについては, 後に詳しく述べる.

## 5 結果と考察

一括修正法には,PDP と NP の 2 つの並列化を行い, 逐次修正法には NP のみの並列化を行った. すべて以下の条件のもとで実験を行った.

- 学習パターン数 $p = 1024$ (約 150 語).
- 学習反復回数 $N = 500$.
- 学習係数 $\mu = 0.1$.
- NETtalk ネットワークサイズ 203-80-26.
- CP-PACS プロセッサの PVP-SW 機能は利用しない.

なお, 時間の実測値は結合重み初期化・学習パターンの読み込みが終った時点から 500 回の学習が終るまでの時間を計測している.

まず一括修正法と逐次修正法おいて, 使用する PU 数を 1, 2, 4, 8, 16, 32, 64, 128, 256 としたときの速度向上率グラフを図 7に示す (ネットワーク分割法では中間素子数の制限により使用できる 64 台までである). また, BP の速度性能指標として広く利用されている MCUPS [1] (Million Connection Updates Per Second) 値による速度性能を図 8に示す. MCUPS は 1 秒間での結合重みの更新回数を示している. ただし, 一括修正法の場合は modify フェイズでの結合重み更新回数ではなく, 結合重み更新量を求めた回数で MCUPS 計算を行うのが普通となっている. よって, 一括修正法と逐次修正法を MCUPS 値によって直接比較できないことに注意されたい.

一括修正法・逐次修正法共に,NP を用いたとき PU32 台でピークが現れている。この主な原因には以下の 2 つがある。

1. 中間・出力素子数の関係上, PU を増やしたときそれぞれが受け持つ計算の均等な負荷分散が行えない. そのため, PU 数の割に計算時間は減らない.
2. 不均等な負荷分散と頻繁に起こる通信のため, 通信に伴う同期待ち時間が積み重なり, 大きなオーバーヘッドとなっている.

したがってこれらを解決するには, ネットワークを単純に縦割する方法だけではなく, より計算負荷が均等に分散でき, 通信がより少ない方法を考える必要がある.

PDP が利用できない逐次修正法は仕方がないとして, 一括修正法だけを考えると,CP-PACS のような MIMD 型並列計算機には NP より PDP が適していることは明らかである. しかし, 実験結果から CP-PACS ではその高速な通信機能により NP でも少ない PU 数では高いスケーラビリティが得られている. これを考慮すると上述した PDP と NP を組み合わせた方法 PDP+NP により, 更に高速化できることが予測できる. ここで問題となるのは, 例えば $P$ 台の PU を使用するとき何台の PU で

[1] $\text{MCUPS} = \frac{(\text{総結合重み数}) \times (\text{パターンデータ数}) \times (\text{学習回数})}{\text{実行時間 (sec)}}$

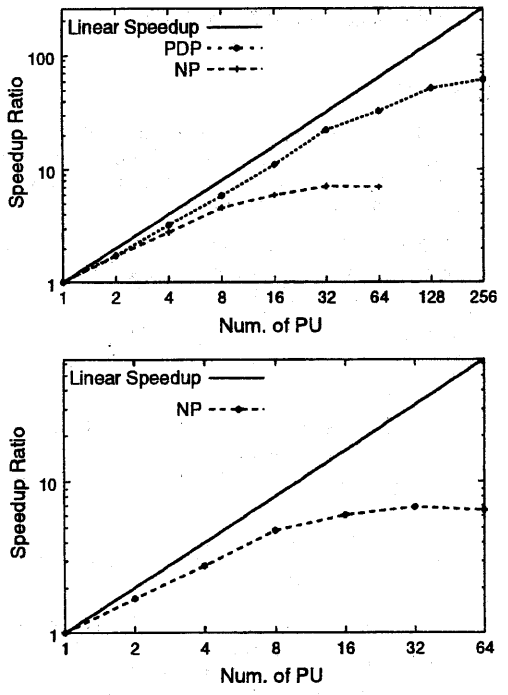


図 7: 逐次修正法 (上), 一括修正法 (下) における速度向上率

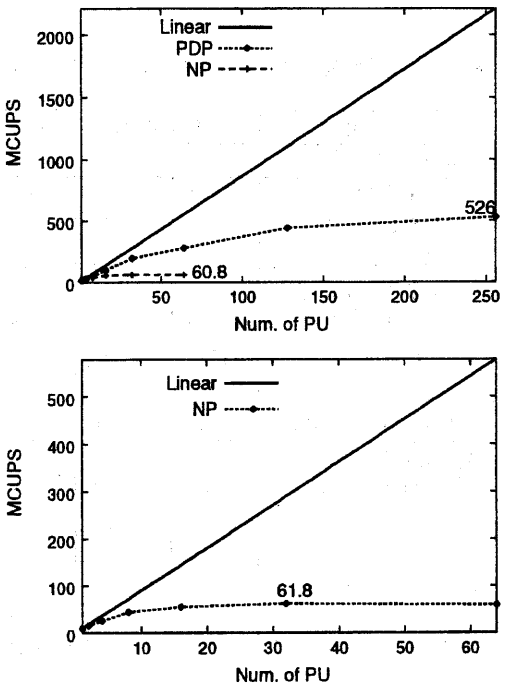


図 8: 逐次修正法 (上), 一括修正法 (下) における MCUPS 値

NP を行ない, それを何グループ作り PDP を行なうかである. 今, $n$ 台の PU で NP を行ない, それを $m$ グループ作るとする. 全 PU 数を $P$ とすると, $P = m \times n$ の条件のもとで性能が最大となる最適な $(m,n)$ の組を求めなければならない. そこで, 最適な $(m,n)$ を求める方法として, PDP の実験結果から $m$ 台のときの速度向上率と NP の実験結果から $n$ 台のときの速度向上率の積をとり, その値が最大になる $(m,n)$ を求めることを提案する. 実際に結果から $P = 256$ のとき速度向上率の積が最大になる $(m,n)$ を求めると, $m = 32, n = 8$ のときであった. つまり,8PU で NP を行ない, その 32 グループで PDP を行なうときの性能が最大であると予測できる.

実際に PDP と NP を組み合わせたプログラムを作成し, CP-PACS 上で実行した. この結果を, 図 9に示す。予測したとおり $m = 32, n = 8$ のときが最大性能となっている. この予測方法は, CP-PACS に限らず他の並列計算機においても有効であると思われる.

既に報告されている並列計算機やニューロコンピュータの MCUPS 性能との比較を表 5に示す. すべて NETtalk をベンチマークとして用いており, "中間素子" は使用した NETtalk ネットワークの中間素子数を示している. 現段階で CP-PACS は, 一括修正法・逐次修正法共に最高性能をマークできた.

## 6 まとめ

ニューラルネットワークの 1 つである Back Propagation 学習を, 超並列計算機 CP-PACS を利用することにより高速化を図った. 実装には, パターンデータパラレル (PDP) とニューロンパラレ

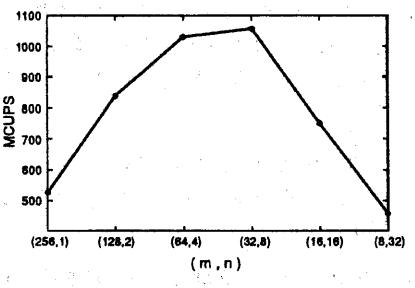


図 9: PDP+NP の MCUPS 性能値

表 1: 他の計算機・ニューロコンピュータとの比較

| マシン名 | PU | 中間素子 | MCUPS |
|---|---|---|---|
| 一括修正法 | | | |
| CP-PACS* | 256 | 80 | 1056 |
| CP-PACS** | 256 | 80 | 526 |
| AP-1000 [5] | 512 | 80 | 82 |
| MP-1216 [6] | 16384 | 120 | 41 |
| CM-5 [7] | 32 | 80 | 18.3 |
| CM-2 [8] | 65536 | 80 | 40 |
| iPSC/860 [9] | 32 | 80 | 11 |
| Transputers [10] | 53 | 60 | 1.54 |
| 逐次修正法 | | | |
| CP-PACS | 64 | 80 | 61.8 |
| VP-2400/10 [11] | 1 | 60 | 60 |
| Sandy [12] | 32 | 60 | 42 |
| MY-NEUPOWER [13] | 106 | 80 | 27 |
| AP-1000 [5] | 64 | 80 | 3.0 |

* PDP+NP($m = 32, n = 8$) の場合 ** PDP のみの場合

ル (NP), 更にそれらを組み合わせた並列化方法を用いた. CP-PACS に最適だと考えられる実装を行った結果, PDP と NP を組み合わせた並列化方法では,1055MCUPS という高い性能を得ることができ, PDP も十分な台数効果が得られた. しかし, NP では計算の不均衡な負荷分散や頻繁に起こる通信のため, 通信同期待ち時間がかさみ, 比較的台数効果が得られなかった.

また, 一括修正法では PDP と NP の最適な組み合わせ方法を提案し, 実験によりこの方法が有効であることを示した. 本最適組み合わせ方法は, CP-PACS に限らず他の並列計算機にも適用できると思われる.

今後の課題を以下に記す.

- CP-PACS プロセッサの PVP-SW 機能を利用し, 更に高速化を図る. BP 計算の詳細な解析を行い, PVP-SW 機能の効果について考察を行う.
- NETtalk だけではなく, 他のニューラルネットワークアプリケーションについても性能比較を行う.
- 実行時間のより詳細なうちわけを測定し, 不均等な負荷分散が起こる原因や, 通信時の同期待ちによるロスなどについて, 詳しい考察を行なう.
- NP の最適な並列化について検討する.

## 参考文献


[1] Hertz, J., Krough, A. and Palmer, R.G.: Introduction to the Theory of Neural Computation. Addison-Wesley Publishing Company: Redwood City, California, 1991

[2] 岩崎洋一, 中澤喜三郎ほか : 計算物理学と超並列計算機 - CP-PACS 計画 -, 情報処理, Vol.37, No.1, pp.10-42 (1996)

[3] 板倉憲一, 朴泰祐ほか : 超並列計算機 CP-PACS の基本性能評価, 情報処理学会研究報告 ARC123-4, pp.19-24(1997).

[4] Terrence J. Sejnowski and Charles R. Rosenberg : Parallel networks that learn to pronounce English text. Complex Systems, 1:145-168,1987

[5] J. Torresen : Parallelization of Backpropagation training for feed-forward neural networks. Doctoral thesis HTH(1996)

[6] Andreas Zell et al.: Problems of massive parallelism in neural network simulation. In Proc. of IEEE Int. Conference on Neural Networks, pages 1890-1895, 1993

[7] J.M. Adamo and D. Anguita : Object oriented design of a BP neural network simulator and implementation on the Connection Machine (CM-5). Technical report, International Computer Science Institute, September 1994. TR-94-46

[8] Xiru Zhang : The backpropagation algorithm on grid and hypercube architectures. Parallel Computing, 14:317-327, Summer 1990

[9] Darin Jackson and Dan Hammerstrom : Distributing back propagation networks over the Intel iPSC/860 hypercube. In Proc. of IJCNN, volume I, pages 569-574, 1991

[10] Shou King Foo, P. Saratchandran and N. Sundararajan : Analysis of training set parallelism for backpropagation neural networks. Int. Journal of Neural Systems, 6(1):61-78, March 1995

[11] E. Sanchez, S. Barro and C.V. Regueiro : Artificial neural networks implementation on vectorial supercomputers. In Proc. of IEEE Int. Conference on Neural Networks, pages 3938-3943, Orlando, FL, June 28 - July 2, 1994

[12] Hideki Yoshizawa and Kazuo Asakawa : Highly parallel architecture for backpropagation using a ring register data path. Fujitsu Sci. Tech. J, pages 227-233, September 1993

[13] M. Yasunga, T. Ochiai :"Performance Evaluation of Neural Network Hardware Using Time-Shared Bus and Integer Representation Architecture" IEICE TRANS. INF. & SYST., VOL. E79-D, NO.6, pp. 888-896, 1996