人工知能システムの疎結合型並列コンピュータによる高速化の研究

湯川 高志

梗概

本論文では，ルールベースシステム，ニューラルネットシステム，概念ベースシステムといった広範な応用が期待されている人工知能システムを対象として，疎結合型並列コンピュータを用いてそれらを高速化する並列化方式を提案し，それらが達成する速度向上を評価する．

人工知能システムでの処理は、処理順序の非決定性という点で、一般的な手続き型処理とは大きく異なっている．この相違点ゆえにシステムが知的振る舞いをするように見えるが，その一方で，処理量が非常に大きくシステムの動作が低速となってしまう．多くの人工知能システムでは，非決定的に処理のために，システムに入力あるいは保持された状況とあらかじめ蓄積された知識との照合処理を行っている．また，完全一致による照合のみならず，類別に基づく類似的一致を扱う場合も多い．したがって，照合と類別の各処理を高速化することにより，多くの人工知能システムにおいて全体の動作速度の向上が期待できる．本論文では，完全一致による照合処理が処理時間を支配する典型としてのルールベースシステム (プロダクションシステム;PS)，パターン情報を帰納的学習によって類別処理を行うニューラルネット (NN) システム，記号情報の中でも特に自然言語情報の類似判別処理を行う概念ベースシステムを対象として，その高速化手法を論ずる．

高速化にあたっては，ヒューリスティックスによるアプローチ，専用プロセッサによるアプローチ，並列処理によるアプローチが考えられる．多くの人工知能システムではヒューリスティクスはすでに導入されており，その上での更なる高速化が要求されている．人工知能システムにおける処理の多様性を考慮すると，処理内容を限定し特化した設計を行う専用プロセッサよりも，汎用のプロセッサを多数利用する並列処理の方が，柔軟性，拡張性，経済性において優れていると考えられる．並列処理を実現する並列コンピュータは，密結合型並列コンピュータと疎結合型並列コンピュータとに大きく分類されるが，10 以下の並列度を主眼においた密結合型よりも，中〜高並列でのスケーラブルな性能を期待できる疎結合型並列コンピュータの方が，大幅な高速化を期待する本論文の目的に適していると言える．

本論文では，以上の議論に基づき，ルールベースシステム，ニューラルネットシステム，概念ベースシステムのそれぞれに対し，疎結合型並列コンピュータ上で並列処理することにより処理を高速化する手法を提案し，それらの速度向上度を評価している．

ルールベースシステムに対しては，条件照合に用いられる RETE マッチ・アルゴリズムについて，その特性を解析的に分析し，種々の並列化軸について高速化の可能性を議論する．その議論に基づいてバス結合型並列コンピュータに好適な並列化方式である “TWIN 方式” を提案する．TWIN 方式を効率的に処理する機能をバスに備えた並列コンピュータ “Presto” の設計・製作とシステムの実装を行い，実験評価の結果を示す．

ニューラルネットシステムに対しては，最も多くの処理時間を占めるバックプロパゲーション (BP) 学習について，その処理を連続行列演算として定式化し，並列化に際しての速度向上の上限値を理論的に導出する．疎結合型並列コンピュータに対する並列化方式を提案し，それら方式が理論的な上限値を達成していること，すわなち最適並列化方式であること示す．

概念ベースシステムでは，類似概念検索処理に対して，探索空間を絞り込みつつバス結合型並列コンピュータにより並列処理する方式を提案する．更に，自然言語処理に好適な意味ネット向け並列コンピュータ SNAP(Semantic Network Array Processor) を用いた並列処理方式を提案し，処理がスケーラブルでありプロセッサ数に応じてリニアな速度向上が得られることを示す．

近年の状況として，インタネット上での溢れんばかりの情報・知識の洪水から知識を抽出・構成するために高速な人工知能システムが必要とされる事，コモディティ・ハードウェアを用いた疎結合型並列コンピュータが実現されコストパフォーマンスが大幅に上昇している事について述べ，本研究が，情報処理技術の今後の発展に資するであろうことをもって結言とする．

目 次

図 目 次

表 目 次

第1章　序論

1.1　研究の目的

情報処理技術が浸透し社会を構成するに欠くべからず要素になるとともに，データ通信ネットワーク技術の発展により地球規模での高速・高度な情報流通が実現しつつある．このような高度情報流通社会においては，人間が快適かつ容易に情報を扱うために，また，大量に流通する情報からユーザにとって必要な情報を的確に捕捉し処理するために，知的な情報処理が求められる．このような背景から，人間の思考過程や認識過程をコンピュータで模擬することにより知的情報処理を実現する人工知能システムが多く研究されている．

人工知能システムには，その動作原理上，従来のコンピュータで実行するとその処理速度が非常に低速になってしまうものが多く存在する．しかしながら，従来のコンピュータの柔軟性，これまでの技術的蓄積，市場規模を背景とした今後の性能的発展を鑑みると，まったく動作原理の異なる人工知能システム専用のコンピュータを新たに創造して従来のコンピュータを上回る性能を実現するのは容易なことではない．従来のノイマン型コンピュータの動作原理を踏襲しつつ，人工知能システムを高速化するためには，多数のプロセッサを用いて処理を並列化することが考えられる．

本研究の目的は，並列コンピュータを用いて人工知能システムの高速化を図ることにある．

並列化のアプローチは，更に，専用プロセッサと専用の相互結合網を用いるアプローチ，汎用プロセッサに専用相互結合網を用いるアプローチ，汎用プロセッサを用い相互結合網も汎用部品を流用するアプローチなどに細分される．このうち汎用プロセッサを用いるアプローチでは，従来型のコンピュータに対する技術的発展のほとんどをそのまま享受することが可能である．相互結合網は，並列化に際してはその性能の支配的要因となり得るため，アプリケーションとなる人工知能システムに適した形態とする必要があることは当然として，更にプロセッサの

性能向上に追従したデータ転送性能が必要とされる．相互結合網に汎用部品を流用するアプローチでは，多くの汎用部品がプロセッサの発展と共に性能が向上しているため，その性能向上をそのまま享受できる．一方，専用型では常に開発投資が必要となり，プロセッサ技術の進歩に追従できるとは限らないが，構成が比較的単純な直接網に限れば，LSI 技術や回路技術の進歩に関しては少なくともプロセッサと同様にその恩恵を享受することができる．以上のことから，汎用プロセッサを，汎用部品を流用した相互結合網または専用設計の直接網により相互接続した疎結合型並列コンピュータを用いることにより，一般のコンピュータ技術の発展に追従することができ，性能的にも常に優位性を保持できると期待される．そこで，本研究では，人工知能システムを疎結合型並列コンピュータによって並列処理し，実用的な処理速度を得ることをねらう．

1.2 研究の動機と主な成果

本論文では，広範な応用がなされており，今後も更なる応用が期待される人工知能システムに対し，疎結合並列コンピュータによる並列化方式を提案する．本論文が対象としたシステムは以下の通りである．

1) ルールベースステム

ルールベースシステムは，人間が外界の状況を認知し，認知結果に適合する行動を起こす過程に基づいたシステムであり，これまで，専門家の判断を模擬するエキスパート・システム (ES) の推論メカニズムとして多く用いられてきた．近年では，人間が協調して知的作業を行う過程を模擬したマルチエージェント・システムへの適用も期待されている．

ルールベースシステムに用いられる推論メカニズムがプロダクション・システム (PS) である．PS は条件照合・競合解消・実行の 3 フェーズから成る認知–実行サイクルを繰り返すことにより推論を行う．PS において実行すべき命令は，サイクルごとの条件照合および競合解消によって決定される．すなわち非手続き型処理の典型である．このうち条件照合フェーズでは，すべてのルール条件部に対し，その時点でのすべてのデータの照合が行われるため，処理量が非常に大きく，システムとしての処理速度が低速となる．

本研究では，まず処理時間を支配する条件照合フェーズの並列化方式を検討し，

次にそれをベースとして，認知–実行サイクル全体を通じた並列化方式について検討する．

2) ニューラルネット (NN) システム

NN システムは生物の神経回路網を模擬したシステムであり，図形認識や音声認識をはじめ様々なパターン識別に多く用いられている．これまでの統計的なアプローチや記号処理的アプローチでは十分な識別能力が得られない対象に対しても，NN システムを用いることにより良い識別結果が得られる場合も多い．NN システムでは，パターン識別を正しく行うために，入力と正答との組を多数提示することで学習を行う．この例示学習のためのアルゴリズムとしてバックプロパゲーション (BP) がもっとも一般的に用いられている．BP は，入力から計算した結果と提示された正答との差分を NN システム内で伝播させつつ，NN の結合係数を微少変化させることにより学習を行う．この例示とそれに基づく結合係数の微少変化を多数回繰り返すことにより，結合係数が次第に収束し，識別能力の高いシステムとなる．BP では，例示を多数 (数万から時には数百万) 回繰り返すことにより学習を行うため，学習に非常に多くの時間がかかる．

BP の処理は，行列演算の繰り返しとそれに基づく行列要素への操作として定式化できる．行列演算であることから並列化に際しては，負荷を均等とすることは容易であると考えられる．となれば，総計算量の増加 (分割損) と通信オーバヘッドがその性能を左右することになろう．本研究では，BP 並列化における総計算量と通信量とを定式化し，その理論的下限値とそれを得るための条件とを検討する．また，それに基づいて並列化方式を考案する．

3) 概念ベースシステム

概念ベースシステムは，人間が持つ日常的な単語に関する知識ベースであり，単語間の包含関係や類似関係を判別するシステムである．World Wide Web(WWW) の検索システムとして，テキスト内容に指定した検索語が含まれているか否かに基づいてスコアリングする全文検索システムが発展してきているが，今後は，指定した語そのものだけではなく，それに類似した語もスコアの対象とする類似検索が必須となって来よう．このような検索システムには，類似語の判別のために概念ベースが必須となる．また，検索のみならず，文章の要約や知識が欠落していた場合に類似知識から補って推論を行う補完推論等，概念ベースは広範な応用

が期待されている．概念ベースの基本的な機能である類似概念検索は，ある概念が与えられた際に，概念ベースに含まれる全概念の中からもっとも類似した概念を探しだす機能である．概念ベースには，日常的に用いられる数万〜数十万の概念が含まれているため，類似概念検索では，これらにすべてに対して類似度を計算する必要があり，非常に時間がかかる処理となっている．

本研究では，並列処理による類似概念検索の高速化を検討する．上述のとおり概念数が非常に大きいため，実用的な速度を得るには，全探索という原始的な方法に対して数千〜数万倍の高速化が必要とされる．本研究では，探索空間を絞り込みつつ中並列の並列化でこれを達成するアプローチと，スケーラブルな並列化方式を考案して高並列コンピュータによりこれを達成するアプローチの両者を検討する．

以下，それぞれについて，本研究での成果を述べる．

1.2.1 プロダクションシステムの並列化

バス結合型並列コンピュータに向けたPSの並列化方式を提案し，解析的に性能を評価して他の方式と比較した．提案した並列化方式 “TWIN 方式” は，条件照合に用いられる RETE マッチ・アルゴリズムの持つ制御流れの並列性 (構造的並列性) とデータ処理の並列性を巧妙に組み合せることにより，通信オーバヘッドと負荷ばらつきとの両者を小さく抑えた方式である．本方式により，低廉に構築可能なバス結合型並列コンピュータにおいても安定的に性能が得られる．

性能評価においては，実際にプログラムを実装することなく並列化時の性能を推定できれば，多くの並列化方式を同一条件下で検討・比較できる．このために解析的性能評価手法を提案している．本評価手法では，RETE マッチ・アルゴリズムに用いられる RETE ネットワークをモデル化し，そのモデルに基づいて，並列化時の速度向上度を計算によって求める速度向上推定式を構築した．本手法を用いれば，PS のアプリケーション・プログラムにあたるエキスパート・システム (ES) の特徴パラメータと並列化方式とから，速度向上度を計算により推定することができる．この速度向上推定手法によって TWIN 方式と既存の代表的な並列化方式を評価し比較した．

さらに，提案した TWIN 方式を実装するのに適したバス結合型並列コンピュータ “Presto” と，PS の推論インタプリタ全体を通じた並列化方式を提案している．

Prestoは，低廉に構築できるバス結合型並列コンピュータであるが，TWIN方式をより効率的に実装するために，放送，同期，プロセッサ間割込を実現する機構をバスに持たせている．PSでは照合が処理時間の大部分を占めるとはいえ，10倍を越える速度向上を得るには，推論の認知–実行サイクルを構成する他のフェーズ，すなわち競合解消フェーズと実行フェーズも並列化する必要がある．このために，TWIN方式による照合並列化を基本とした，PS推論全体の並列化方式を提案している．更に，NTTで開発されたKBMS推論インタプリタをこの並列化方式に基づいてPresto上に実装し，性能を評価している．

1.2.2 ニューラルネットシステムの並列化

BP学習の並列処理について論じ，バス結合型並列コンピュータと2次元アレイ型並列コンピュータとに対する並列化方式を提案した．また，BP並列化に際しての本質的な処理量および通信量について考察し，理論的な下限値とそれを実現するための条件(最適並列化条件)とを明確にした．更に，提案方式のうち2次元アレイ型並列コンピュータに対する方式が，ここで明らかにした最適並列化条件を満足していること，すなわち計算量・通信量が本方式よりも小さい並列化方式は存在しないことを示した．また，バス結合型並列コンピュータする方式では，従来方式の通信量がプロセッサ数に比例していたのに対し，それをプロセッサ数の平方根オーダに抑えることができた．

1.2.3 類似概念検索の並列化

類似概念検索に対する並列化方式としてバス結合型向けの並列化方式と意味ネット並列マシン向けの並列化方式とを提案した．バス結合型向けの方式は，類似度計算法の特性を利用して見込みのない概念に対する類似度計算を省くとともに，バス結合型並列コンピュータによって処理を並列化する．これにより，原理的な検索方式に対して1万倍台の速度向上が得られる．意味ネットマシン向け方式は，意味ネット並列マシンSNAP(Semantic Network Array Processor)上に概念ベースをマッピングして，類似度計算を高並列化できるスケーラブルな並列化方式である．

1.3 本論への準備

1.3.1 並列コンピュータとその分類

コンピュータに用いられるプロセッサの処理速度は，LSI 技術，特に微細加工技術の進歩に支えられて，これまで長足の進展を遂げてきた．しかしながら，現在主流である CMOS プロセスも，速度的な面では，さほど遠くないうちに限界を迎るであろうと予測されている．また，それに代る GaAS やジョセフソン素子等は，消費電力，集積度，歩留りなどの面で未だに多くの問題を抱えており，近い将来に CMOS に取って代わる技術にはなりそうにない．

単一のプロセッサでの速度の限界を突破するために，多数のプロセッサを用いてひとつのアプリケーションを並列に処理する技術が古くから研究されてきた．アプリケーション・プログラムを適切に複数の部分 (処理単位) に分割し，複数のプロセッサでそれぞれの処理単位を実行すれば，単一プロセッサで実行する場合よりも短い時間で結果を得ることができる．このような並列処理を行うことを目的として構成されたコンピュータが並列コンピュータである．しかしながら，並列コンピュータと一口に言っても，扱う処理単位の細かさ (粒度)，命令とデータの流れとの関係，プロセッサ間のデータ交換の方法などによって，その構成には非常に広範なバリエーションがある．そこで，本節では，並列コンピュータの分類 [92] を示し，そのうち MIMD 型マルチコンピュータを更に共有メモリ型 (密結合型) と分散メモリ型 (疎結合型) とに分類する．

コンピュータ・アーキテクチャは，機械命令の起動形態によってコントロール駆動型，データ駆動型，要求駆動型に分類できる．また命令とデータの流れにより，SISD(single instruction single data stream) 方式，SIMD(single instruction multi data stream) 方式，VLIW(very long instruction word) 方式，演算パイプライン (arithmetic pipeline) 方式，MIMD(multi instruction multi data stream) 方式に分類できる．データ駆動型と要求駆動型では必然的に MIMD 方式になる．これらは非ノイマン型並列コンピュータと呼ばれる．コントロール駆動型はあらゆる方式を取り得るが，SISD 方式が従来の単一プロセッサのコンピュータに相当する．他の方式をとるものはノイマン型並列コンピュータと呼ばれる．コントロール駆動型での各方式について，以下にその概要と特徴を述べる．

SISD 方式

SISD 方式では，演算装置を一つ持ち，一つの機械命令の実行によって一つの

演算結果が得られるもので，並列コンピュータとしての分類には含まれない．汎用コンピュータがSISD方式の代表である．また近年，高級言語コンピュータが開発されているがこれもSISD方式に分類される．高級言語コンピュータは，高級言語のソースコードを直接，あるいは抽象度の高い中間形式に変換して，ハードウェアにより処理するもので，1ステップあたりの処理が複雑な人工知能向け言語を対象としたものが多く開発されて来た[30, 36, 49]．しかしながら，市場規模の圧倒的に大きな汎用プロセッサへの開発投資の集中によって，現在では，高級言語向け専用コンピュータの性能的優位性は失われている．

SIMD方式

SIMD方式は，演算装置を多数持ち，それらが一つの制御部から発せられた演算命令を同時に実行する．演算装置は結合網を介して互いに接続されている．すべての演算装置が同一の命令により駆動されるため，処理方法が単一なデータが多数ある場合にその真価を発揮する．ベクトルや配列等に対する演算を必要とする分野に適しており，科学技術計算，画像処理，信号処理，データ・サーチ，パターン・マッチング，連想処理などに利用できる．

人工知能を応用対象としたSIMD方式並列コンピュータの代表的なものとして，Connection Machine[33]があげられる．Connection Machineは1ビットの演算器と4kビットのメモリから構成されるプロセッシング・エレメント(PE:processing element; セルとも呼ぶ）を2進n-キューブ網で結合したものであり，集合演算，木構造演算，バタフライ演算，文字列演算，サーチ，配列演算などを高速に処理できるとされている．

VLIW方式

VLIW方式は，水平型マイクロプログラムから派生した方式であり，語長の非常に長い機械命令を多数のフィールドに分割し，個々のフィールドで演算器やメモリを独立に制御する方式である．長形式の命令に同時実行可能な演算が埋め込まれてその性能を発揮することができる．この演算の埋め込みはコンパイラによって行われるので，コンパイラ技術が重要となる．並列性の抽出をコンパイラで行う分，実行時の並列制御は簡単となるため，ハードウェア構成は単純となり高速化を達成できる．

演算パイプライン方式

演算パイプライン方式では，演算装置が多数のステージから構成されており，データを流れ作業の原理で処理する．最初のデータが入力されてから最初の結果がでるまでは時間がかかるが，それ以降は連続的に結果が得られる．各ステージでの処理時間が等しく，データが十分に大量であればステージ段数に等しい速度向上が得られる．ベクトルや配列に対する演算を高速に実行するスーパ・コンピュータでは主流の方式である．

MIMD 方式 (マルチプロセッサ方式)

コントロール駆動型の MIMD 方式は，プロセッサを相互結合網により複合した形態である．プロセッサは，それぞれが演算器や制御部を持ち，自律的に動作できる．このため，マルチプロセッサ方式とも呼ばれる．

次節で述べるように，主記憶の位置により，それを共有する共有メモリ型 (密結合型)，個々のプロセッサが独自に主記憶を持ちメッセージ交換によりデータをやり取りする分散メモリ型 (疎結合型) に分られる．また，相互結合網も，直接網と間接網に分類でき，更に同一分類中にも多くのトポロジーが考えられる．この相互結合網は，並列コンピュータのコストと性能とを左右する．また，適用分野もこれに依存する．

このようなことから，適用分野が比較的明確で限定的な他の方式に比べて，マルチプロセッサ方式は，様々な適用分野が考えられ，また，それぞれの分野で様々な相互結合網の形態に対する並列化アルゴリズムが研究されている．人工知能向けとしても多く開発されており，DADO[21]，Non Von などが代表的なものとしてあげられる．DADO は，木構造の分散メモリ型並列コンピュータであり，この上で PS 等の人工知能システムが実装されている．次節では，本方式についてより詳しく述べる．

密結合型並列コンピュータと疎結合型並列コンピュータ

MIMD 方式 (マルチプロセッサ方式) は，演算装置と制御部からなるプロセッサを相互結合網により多数結合した方式である．プロセッサに命令やデータを供給する主記憶の配置によって，密結合型 (共有メモリ型) と疎結合型 (分散メモリ型) に分類される．

密結合型並列コンピュータでは，各プロセッサが主記憶装置を共有して動作す

る．プロセッサからは共有されている主記憶領域がメモリ空間上に直接見えることになり，これを命令の記憶領域としてもデータの記憶領域としても利用することができる．主記憶がすべてのプロセッサから共有されるため，主記憶装置に対するアクセス・トラフィックが大きくなり，プロセッサ数が増加するとここにボトルネックが生じて性能が低下する．これを避けるために，他のプロセッサとはアクセスが競合しないローカル・メモリを備えているものも多い．また，プロセッサやそれの群ごとにキャッシュ・メモリを装備する場合もある．この場合にはキャッシュ・コンシステンシの制御が重要となる．

プロセッサと主記憶装置の結合は，単一バス結合が最も一般的であるが，階層バス結合やクロスバ結合を採用したものもある．いずれの場合も，共有メモリへのアクセス競合によってその性能が抑えられるため，一般にプロセッサ数が少ない領域で使われる．汎用大型機の双頭プロセッサ，UNIX サーバや PC サーバにおける SMP(symmetric multi processor) 等が代表的である．

疎結合型並列コンピュータは，個々のプロセッサが自身の主記憶 (ローカルメモリ) を持ち独立した 1 台のコンピュータとして動作する．このため，これをマルチコンピュータ方式とも呼ぶ．プロセッサ間のデータ交換は入出力ポート等を通じ，相互結合網を介してメッセージ通信の形式で行われる．後述するようにプロセッサを結合する相互結合網には様々な方式が考案されている．相互結合網のトポロジにも依存するが，一般に多数のプロセッサによって構成することが可能であり，数 10 台〜数万台規模の並列コンピュータが開発されている．

密結合型並列コンピュータでは，自プロセッサ内でのデータへのアクセスも，プロセッサ間でのデータの交換も，プロセッサから見ればともにメモリアクセスであるため，プログラムは容易である反面，メモリ・アクセス競合等のオーバヘッドが予測しにくい．一方，疎結合型並列コンピュータでは，自プロセッサ内に閉じたアクセスと他のプロセッサとのデータ交換は，プロセッサにとって全く異なった動作であり，プログラマが明確に意識することになる．このため，アルゴリズムの並列化は，分割による処理量の増加，通信オーバヘッド，プロセッサの負荷ばらつき等を十分に考慮し注意深く行う必要がある．その反面，性能があらかじめある程度予測可能であり，並列化にあたっても実際にプログラムを実装することなく様々な方式を検討することができる．また，良い並列化方式を見つけることができれば高性能を安定に達成できる．このようなことから，本論文では，疎結合型並列コンピュータを用いた人工知能システムの並列化方式を論ずる．

次節では，疎結合型並列コンピュータを相互結合網に着目して分類し，本論文が対象とするアーキテクチャについて述べる．

疎結合型並列コンピュータの分類

疎結合型並列コンピュータにおいてプロセッサを結合する相互結合網は大きく間接網と直接網とに分けられる．間接網は，プログラム実行時あるいは動作中に接続を動的に変更できる結合網を言い，直接網は，接続形態がハードウェアの製作時点で固定的に決っている結合網を言う．間接網は必然的にプロセッサ間にスイッチ群を設置し，これらを制御することによって結合経路を構成する形態となる．スイッチの接続形態により，クロスバ網，Beneš 網，オメガ網，バンヤン網等がある．いずれの網でも，スイッチは比較的単純な構成をしているが，ピン数の制約から LSI 化しにくいという問題がある．このため，間接網では結合網部のハードウェア構成が大掛かりとなってしまい，コスト高を招く．

直接網は，間接網よりも単純で，各プロセッサの通信用ポートを配線により固定的に接続したものである．結合のトポロジとしては以下のようなものがある．

完全網

すべてのプロセッサが互いに接続された網である．プロセッサ数を N とするとプロセッサあたりのポート数は $N-1$ 必要で，全結線数は $N(N-1)/2$ となる．このように多数のポートと結線が必要となるため，大規模システムの実現は困難である．

スター網

中心となる通信制御部に対しプロセッサが放射状に接続されたトポロジである．プロセッサあたりのポート数は 1 であり最も少ない．ただし，中心に位置する制御部のポート数は N であり，ここがボトルネックとなる．また，プロセッサ間の距離は，どのプロセッサをとっても 2 である．バスを通信制御部ととらえるとバス結合型もスター網の特殊な場合と言える．ただし，バスはプロセッサ間のデータ交換に対して能動的な役割を果たさないので，プロセッサ間距離は 1 と考えられる．

通信制御部は多数のプロセッサからのデータ転送要求が集中するため，それらが衝突しないように調停を行う必要がある．プライオリティ・エンコー

ダなどで構成されたアービタによる調停やポーリングにより制御部が調停機能を提供する集中制御方式は，バックプレーンなどを利用した同一筐体内のバスに利用される．集中制御方式では，プロセッサ番号の識別のために多数の制御線が必要となるが，バックプレーン利用のバスでは一般にデータ線やアドレス線も多数並列しているため，多少の制御線は問題とならない．CSMA/CD(Carrier Sense Multiple Access/Collision Detect) に代表されるような，プロセッサが自律的に調停を行うのが分散制御方式である．これは制御信号をデータ信号と同一の線上に載せることができるため，1 本の同軸ケーブルや 1 組のより対線をバスとして利用してシリアル伝送する結合形態に適している．

鎖網とリング網

隣接するプロセッサ同士を結合した網が鎖網である．両端のプロセッサ同士も接続すればリング網となる．プロセッサあたりのポート数は 2 であり，プロセッサ間距離の平均は $N/4$ である．

格子網 (ラティス，メッシュ，トーラス)

プロセッサを格子の交点に配置し，辺を通信路としたものが格子網である．格子網には正方格子，三角格子，六角格子等が考えられるが，正方格子 (メッシュ) が最も良く用いられる．正方格子の場合，プロセッサは 4 つのポートを持ち隣接する 4 つのプロセッサと通信できる．最左端と最右端，最上端と最下端同士を結ぶとトーラス網となる．プロセッサ間距離の平均は $\sqrt{N}/2$ である．

メッシュ網，トーラス網により結合された並列コンピュータは 2 次元アレイ型とも呼ばれる．2 次元アレイ型並列コンピュータは，プロセッサのポート数が実現容易な範囲にあり，様々なアルゴリズムの並列化に適しているので，この結合網によるシステムが多数開発されている．

木状網 (ツリー)

ツリーのノードにプロセッサを配置し，枝を通信路とした網である．2 進木の場合，プロセッサあたりのポート数は 3，プロセッサ間の平均距離は $2\log_2 N$ である．根に近いプロセッサほど処理が重くなるため，葉ノードのプロセッサのみが演算処理を行い，他のノードはデータの中継のみを行う方式もある．

単純な木状網はLSIチップ内に効率よく埋め込むことができ，網の拡張も容易である．しかし，通信の迂回路がないため，ノードがひとつでも故障するとシステム全体が機能不全に陥るという問題がある．

超立方体網 (2進 n-キューブ，ハイパーキューブ)

$N = 2^n$ 個のプロセッサがそれぞれ n の通信ポートを持ち，プロセッサ番号を2進数で表した時にハミング距離が1のプロセッサ同士が結合している網が超立方体網である．プロセッサ間の平均距離は $\frac{1}{2}\log_2 N$ である．二つの2進 n-キューブがある時に，各プロセッサに通信ポートを一つ増設し，同一番号のプロセッサ同士を結合することにより，容易に2進 $n+1$-キューブを構成することができる．このような数学的な美しさと，リング網，トーラス網，ツリー網などを包含していることから，汎用並列処理向きの相互結合網として注目されている．ただし，接続ポート数が他の結合網に比べて多いため，プロセッサのコストは高くなる傾向にある．

1.3.2 本研究が対象とする並列コンピュータ

本論文では，実用的な領域で高コストパフォーマンスが得られるような人工知能システムの並列化をねらっている．この目的のために，まず，プロセッサあたりの通信ポート数が最も小さく低廉で容易に構成できるバス結合型を対象とする．バス結合型並列コンピュータは，VMEやPCI等の汎用バスに汎用のプロセッサボードを実装しただけも構成可能である．更に近年では，パーソナルコンピュータをローカルエリアネットワーク用のネットワークインタフェース によって接続した形態のBeowulf型並列コンピュータも多く製作されており，多くの人が容易にこのタイプの並列コンピュータを利用することが可能となって来ている．更に，汎用のマイクロプロセッサやその周辺部品を用いているため，これらの部品の進歩にも容易に追随でき，常に最新の部品性能を享受できる．このようなアーキテクチャにおいて，良好な性能が得られる並列化方式を考案できれば，その意義は大きいと考える．

更に，用途によっては絶対性能の追求も必要であるため，それぞれの応用システムに最も適すると考えられる結合網も対象とし，これらに対する並列化方式も検討する．PSに対しては，すでに多くの形態の結合網について研究されていることと，本論文で提案しているバス結合でも十分に良好な性能が得られることから，

他の結合網は取り上げない，BP に対しては，その演算に最適な結合網としてトーラス網を用いる．概念の類似判別では，概念の表現である意味ネットを直接マッピングするのに適したトポロジの網を対象とする．

第2章　プロダクションシステム条件照合の並列化

ルールベースシステムは，人間の認知行動過程に基づいたシステムであり，エキスパートシステム (ES:expert system) はその代表的応用である．また，近年では，複数の能動的なシステムが知識を交換しながら協調して問題解決を行うマルチエージェント・システムへの応用も期待されている．

ルールベースシステムの推論機構としても最も一般的に用いられているプロダクションシステム (PS:production system)[72] は，条件照合・競合解消・実行の 3 フェーズからなる認知-実行サイクルにより推論を行う．この認知-実行サイクルの中で，条件照合フェーズは，ES を構成する多数のルール条件部と多数のワーキングメモリエレメントとの照合を行うため，その処理量が推論全体の中で大きな比率を占める．特に大規模な ES では，条件照合フェーズでの処理に消費される時間が非常に大きなものとなり，推論時間の大幅な増大を招くことになる．条件照合フェーズを高速に処理するために，RETE マッチ [17] や TREAT[65] といった条件照合処理の特性を利用して処理量を削減するアルゴリズムが開発されているが，それらをもってしても認知-実行サイクルの 95%以上を条件照合フェーズが占めており [23]，更なる高速化が必要とされている．

本章では，PS の推論を高速化するために，条件照合フェーズに最も多く用いられる RETE マッチ・アルゴリズムを並列処理する手法について論ずる．まず，PS と，その PS を成功に導く要因になったともいえる条件照合アルゴリズム RETE マッチについて説明する．次に，RETE マッチ・アルゴリズムで用いられている RETE ネットワークのモデル化を行い，そのモデルに基づいて並列度及び並列化時の速度向上度を計算する，RETE マッチ・アルゴリズム並列化の解析的評価法を提案する．続いて，RETE マッチ・アルゴリズムの並列化軸を抽出し，単純にこの並列化軸に沿った基本的な並列化方式を示す．これら基本並列化方式及び既存の並列化方式を，前述の解析的評価法に基づいて評価する．評価結果から，RETE

マッチ・アルゴリズムは本質的には数十倍の高速化の可能性があることがわかる．しかし，従来方式では，十分に並列性を利用できていないか，または，通信オーバヘッドの影響を受けやすいものであるため，現実的な通信バンド幅に制限した場合には 10 倍程度の速度しか得られないことを示す．

上記の並列化軸の抽出と評価により得た知見に基づき，RETE マッチ・アルゴリズムの並列性を引出し，実現容易な構成の並列コンピュータにおいても 10 数倍以上の速度向上度を実現できる TWIN(top two-input node distribution) 方式を提案する．TWIN 方式は，RETE ネットワークの構造的な並列性と，RETE ネットワークを構成するノードでの処理におけるデータ的な並列性を，通信の必要性が少なくなるように組合わせた方式である．このため，RETE マッチ・アルゴリズムが内包する並列性を十分に引出すと同時に，通信頻度も小さく抑えることが可能となり，実現が容易なバス結合型並列コンピュータでも 10 数倍から数十倍の速度向上比が期待できる方式である．従来方式を評価したと同一の解析的評価手法により TWIN 方式の性能を評価するとともに，実際の ES を用いて実験を行い，本方式の有用性を実証する．

2.1 プロダクションシステムと RETE マッチ・アルゴリズム

本節では，PS の構成及び推論動作である認知–実行サイクルについて述べるとともに，認知–実行サイクル中で最も処理量が多い条件照合フェーズに用いられている，RETE マッチ・アルゴリズムについて説明する，

2.1.1 プロダクションシステム

1960 年代から 70 年代にかけて化学構造同定システム DENDRAL[16] や医学診断・治療支援システム MYCIN[85] など，専門家と同様に高度な判断ができる ES が開発された．70 年代後半になると，これら ES から知識表現技法及び推論制御技法を抽出し応用独立化を図った ES 構築用ツール [63] が開発されるに至る．これらに採用された代表的な推論制御技法が `IF—THEN`—型のルールに基づいて推論を行う PS である．PS を用いた ES 構築用ツールは，推論の高速化を実現する RETE マッチ・アルゴリズムを採用した OPS5[9] が開発されるに至り，ほぼ実用の域に

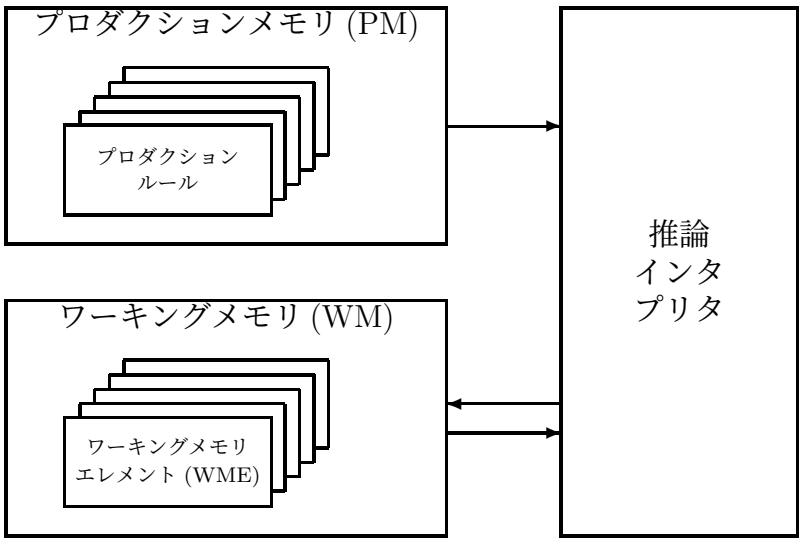


図 2.1: プロダクションシステムの構成

達した．その後，この OPS5 と同様の推論制御技法と RETE マッチ・アルゴリズムを採用した ES 構築ツールが多く開発されている [45].

PS は，*ルール*を格納するプロダクションメモリ (PM:production memory) と，事実データや中間データを格納するワーキングメモリ (WM:working memory)，そして，PM，WM の双方を参照し解釈実行する推論インタプリタから構成される [72]. 図 2.1 に PS の構成を示す．PM には `if`—`then`—型のルールが格納される．個々のルールは，条件部または LHS(left-hand side) と呼ばれる条件要素の連接と，アクション部または RHS(right-hand side) と呼ばれる動作の集合とから構成される．WM には，事実，状況，推論の途中結果などの宣言的なデータが格納される．これらの個々のデータはワーキングメモリエレメント (WME:working memory element) と呼ばれる．PM は長期記憶 (long-term memory)，WM は短期記憶 (short-term memory) と呼ばれることもある．

WME 及びプロダクションルールは具体的には以下のように記述される (OPS5[9] での文法を例としている).

WME

WME は，その WME が属するクラス名と，0 個以上の属性名–属性値の組によって記述される．クラスはその WME が含む属性を定義しており，以下のように書く．

```
(literalize Person name father mother age
            city-of-residence)
(literalize Move name type from-city to-city)
```

これは Person というクラスが属性 name, father, mother, age, そして city-of-residence を持つことを表す．また，Move というクラスは，属性 name,from そして to を持つことを表している．

WME は，必ず上述のクラスのいずれかに属し，以下のように記述される．

```
(Person ^name Teresa
        ^mother Helen
        ^father Cleve
        ^age 21
        ^city-of-residence Swannanoa)
```

リストの先頭，すなわち上の例では Person，がその WME の属するクラスを表している．以後の要素は，属性名とそれに対応する値の組の羅列である．この例では属性 name の値が Teresa であり，属性 mother の値が Helen，属性 father の値が Cleve，属性 age の値が 21，そして属性 city-of-residence の値が Swannanoa であることを表している．

実際は，WME には上述のクラスと属性名–属性値に加えタイムタグ (time tag) が付与されて WM に格納される．タイムタグは，WME を一意に識別するための番号で生成順にふられる．

プロダクションルール

プロダクションルールの例を以下に示す．

```
(p new-residence
   (Person ^name <name1> ^city-of-residence <city1>)
   (Move   ^name <name1> ^type permanent
           ^from-city <city1> ^to-city <city2>)
```

```
-->
(modify 1 ^city-of-residence <city2>)
(remove 2))
```

プロダクションルールであることを表すpに続き，ルール名new-residenceが記述される．次に条件部が記述され，-->以降にアクション部が書かれる．条件部は，複数の条件要素の連接として構成される．この例では，

```
(Person ^name <name1> ^city-of-residence <city1>)
```

及び

```
(Move   ^name <name1> ^type permanent
        ^from-city <city1> ^to-city <city2>)
```

がそれぞれ条件要素である．

条件要素には，その条件要素にマッチするためにWMEが満足すべき条件としての属性名と値の組が羅列される．値としては，定数値や計算によって得られる値 (実現値) だけでなく，変数 (パターン変数) も指定できる．上の例では，^type permanentにおいて実現値が指定されている．これは，属性typeの値としてpermanentを持つWMEにのみマッチする．一方，^name <name1>，^from-city <city1>などはパターン変数が指定されている．この場合は，同一の変数に対して同一の値が束縛できるようなWMEの組が存在した場合に，そのWMEの組にマッチする．次の二つのWME,

```
(Person ^name Teresa
        ^mother Helen
        ^father Cleve
        ^age 21
        ^city-of-residence Swannanoa)

(Move   ^name Teresa
```

```
        ^type permanent
        ^from-city Swannanoa
        ^to-city Pittsburgh)
```

がある場合に，第 1，第 2 の両方の条件要素に含まれるパターン変数<name>，<city1>を，それぞれ Teresa，Swannanoa に束縛することが可能なため，この二つの WME はマッチする．また，同時に，第 2 の条件要素に含まれる変数 city2 は Pittsburgh に束縛される．

アクション部には，条件部を満足する WME の組 (インスタンシエーション) があった場合に実行されるアクションを記述する．アクションには，WME に対する生成 (make)，更新 (modify)，削除 (remove) といった操作や，情報の表示，システムの制御，他のプログラムの呼出しなどが記述される．WME 操作のアクションにおいては，操作対象の WME をそれがマッチする条件要素の番号で指定することができる (make の場合は新たな WME を生成するので当然ながら操作対象の指定はない)．例では，modify の対象は第 1 の条件要素にマッチした WME であり，remove の対象は第 2 の条件要素にマッチした WME である．ただし，ここでは説明を簡潔とするためこの記述法を用いているが，実際のプログラミングでは保守性を高めるためにパターン変数と同様の変数記述を用いて以下のように書くことが多い．

```
(p new-residence
   { (Person ^name <name1> ^city-of-residence <city1>) <p1> }
   { (Move   ^name <name1> ^type permanent
             ^from-city <city1> ^to-city <city2>) <m1> }
   -->
   (modify <p1> ^city-of-residence <city2>)
   (remove <m1>))
```

2.1.2 認知–実行サイクル

ES構築ツールのほとんどで用いられている，やり直しなしの前向きPS(irrevocable forward chaining production system)では，次の3つのフェーズからなる認知–実行サイクル(recognize-act cycle)を繰り返すことによって推論を行う．

1. 条件照合 (match)

すべてのルールについて，条件部とその時点のWMとの照合を行う．条件部にマッチするWMEの組が見つかった場合には，これらマッチしたWMEの組とルールそれ自体に関する情報(これをインスタンシエーションと呼ぶ)を競合集合(conflict set)に追加する．

2. 競合解消 (conflict resolution)

条件照合が完了した時点で複数のインスタンシエーションが競合集合にある場合に，定められた戦略に従って実行すべきインスタンシエーションを一つだけ選び出す．この戦略を競合解消戦略(conflict resolution strategy)と呼ぶ．代表的な競合解消戦略であるLEXとMEAについて以下に述べる．

LEX

LEX戦略は，より新しいWMEを含み，より厳しい条件を満足するインスタンシエーションを選択する戦略である．以下の手順で選択する．

1. 以前選択したインスタンシエーションは選択しない(同一処理を永久に繰り返すことがないようにするため)
2. タイムタグが最も新しいWMEを含むインスタンシエーションを選択する．
3. 条件要素数の多いルールのインスタンシエーションを選択する．条件要素数が同じ場合には，条件部で行われるテスト数が多いインスタンシエーションを選択する．
4. 上記を順に適用してもインスタンシエーションを一つだけに絞り込めない場合は，残ったインスタンシエーションから任意に選択する．

MEA

MEA戦略はLEX戦略を拡張したもので，以下の手順でインスタンシエーションを選択する．

1. 以前選択したインスタンシエーションは選択しない (同一処理を永久に繰り返すことがないようにするため)
2. インスタンシエーションを構成する最初の WME(条件部の最初の条件要素にマッチする WME) のタイムタグが最も新しいものを選択する.
3. タイムタグが最も新しい WME を含むインスタンシエーションを選択する.
4. 条件要素数の多いルールのインスタンシエーションを選択する．条件要素数が同じ場合には，条件部で行われるテスト数が多いインスタンシエーションを選択する.
5. 上記を順に適用してもインスタンシエーションを一つだけに絞り込めない場合は，残ったインスタンシエーションから任意に選択する.

MEA 戦略では，インスタンシエーションを構成する最初の WME のタイムタグを選択の基準とする条件が追加されている．これにより，第 1 条件要素に副目標や制御の切替えを記述して，推論動作の見通しを良くすることができる.

3. 実行 (act)

競合解消によって選びだされたインスタンシエーションに対して，そのルールのアクション部を実行する．これをルールの発火 (fire) と呼ぶ．一般に，アクション部には WME の追加・更新・削除といった操作が書かれるため，この実行フェーズにより WM の内容が変化する.

実行フェーズでの WME の追加・更新・削除によって，WM が変化し条件照合フェーズにおいて前回とは異る WME とルールの組合せがマッチすることになる．これにより競合解消フェーズにおいて新たなインスタンシエーションが選択され，このインスタンシエーションのルールが実行フェーズで発火し，WM の新たな変化を引起す．このような動作の繰返しにより推論が進んで行くことになる．この認知–実行サイクルが停止するのは次のような場合である.

- 条件照合フェーズが完了した時点で競合集合にインスタンシエーションが一つもない場合
- アクション部に明示的な停止命令が記述されたルールが発火した場合

これら停止条件が満足されると，推論インタプリタは停止し，推論完了となる．

2.1.3 RETE マッチ・アルゴリズム

条件照合フェーズでは，最も原始的には，PM に含まれる個々のルールに対し，WM に含まれる WME のあらゆる組合せがそのルールの条件部にマッチするかを調べることになる．WM に m 個の WME があったとすると，n 個の条件要素を持つルールに対しては，m^n の組合せに対する条件部の照合が行われることになる．実際には，条件要素には照合対象とする WME のクラスを指定するため，これよりも少ない組合せとなるが，オーダとしては変わらない．このことから，原始的な照合処理では，ルール数や WME 数の増大とともに条件照合フェーズでの処理量が急激に増大し，推論動作が非常に低速なものとなってしまう．

条件照合フェーズを効率化する方法は多く研究されており，そのなかで最も広く用いられているのが OPS5 を成功に導いた RETE マッチ・アルゴリズム [17] である．RETE マッチ・アルゴリズムは，前サイクルでの条件照合結果を記憶することによって，WME の変化分に対してのみ照合処理を行うことで毎回の条件照合フェーズでの照合処理を削減する．RETE マッチ・アルゴリズムでは，すべてのルールの条件部は RETE ネットワークと呼ばれる一種のデータフローグラフに展開され，変化した WME (トークンと呼ばれる) がこれを流れる事により照合が行われる．以下のようなルールに対する RETE ネットワークの概念図を図 2.2 に示す．

```
(p R1
   (C1 ^a1 100 ^a2 <x>)
   (C2 ^b1 ABC ^b2 <x>)
   -->
   (make C3 ^c1 <x> ^c3 ABC))
```

RETE ネットワークは次のようなノードから構成されている．

ルートノード (root node)

RETE ネットワーク全体の根となるノード．すべてのトークンはここから RETE ネットワークに注入される．

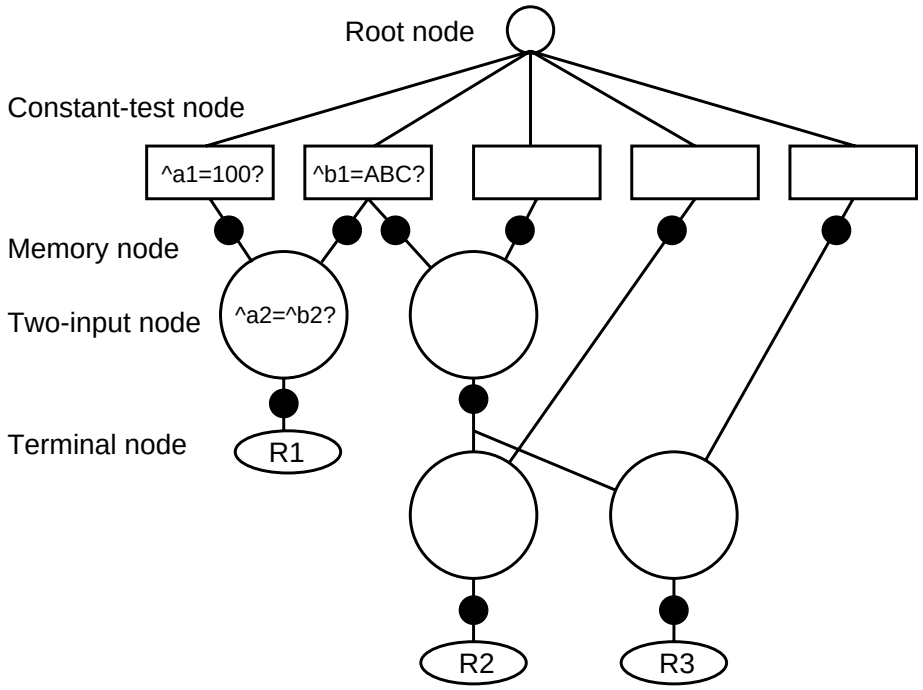


図 2.2: RETE ネットワークの概念図

定数テストノード (constant-test node)

クラスが適合しているか，属性の値が与えられた定数値と等しいかといった条件要素内に閉じたテスト (条件内テストと呼ぶ) を行うノード．1 入力ノード (one-input node) とも呼ばれる．

2 入力ノード (two-input node)

複数の条件要素に書かれた同名のパターン変数が同一の束縛値を持つかどうかをテスト (条件間テストと呼ぶ) するノード．

終端ノード (terminal node)

ルールの条件部に記述されたすべての条件要素にマッチしたトークンが到達するノード．終端ノードに到達したトークンはルールの情報とともにインスタンシエーションとして競合集合に追加される．

メモリノード (memory node)

定数テストノード及び 2 入力ノードを通過したトークンを記憶するノード．定数テストノード直下のメモリノードを α メモリ，2 入力ノード直下のものを β メモリと呼ぶ．

ルートノードから定数テストノードまでは条件要素内に閉じたテストを行うネットワークであり判別ネットまたは弁別ネットと呼ばれる．また，2 入力ノードで構成されるネットワークは条件要素間のテストを行うもので結合ネットと呼ばれる．

RETE マッチ・アルゴリズムでの条件照合処理は以下のような手順で行われる．

1. 実行フェーズでの WM 更新によって WME が変化すると，その変化分がトークンとして RETE ネットワークに注入される．
2. ルートノードから伝播したトークンは，定数テストノードにおいて条件要素内の定数値との比較が行われる．このノードでの比較が成功すると，その下つながる枝に伝播する．
3. 定数テストノードを通過したトークンは，そのノードと接続された 2 入力ノードへ伝播するとともに，その枝上のメモリノード (α メモリ) に記憶される．トークンが伝播してきた 2 入力ノードは活性化 (activation) される．2 入力ノードでは，条件要素にまたがる変数の照合を行うために，入力となる 2 本の枝のメモリノードに記憶されたすべてのトークンとの突合せが行われる．この処理は結合演算 (join) とも呼ばれる．ここでの比較が成功すると，それらマッチした二つのトークンの組を新たなトークンとして，その下の枝へと伝播させる (図 2.3)．
4. 更に従属に 2 入力ノードが接続されている場合には，伝播してきたトークンはその 2 入力ノードに入力されるとともに，枝上のメモリノード (β メモリ) に記憶される．2 入力ノードでは上のステップと同様の突合せが行われる．従属につながった 2 入力ノードがある限り，同様の処理が繰返される．
5. 2 入力ノードをすべて通過して終端ノードに到達した場合，そのトークンは照合に成功したことになる．ルールの情報とともにインスタンシエーションとして競合集合に登録される．

上記の動作をする RETE マッチ・アルゴリズムが条件照合フェーズを効率的に処理できるのは，以下の二つの特徴による．

中間結果の保存

RETE ネットワークの α メモリ，β メモリには，以前のサイクルでの条件照合の途中結果が保存されることになる．このため，サイクルごとにすべての WME に対して条件テストを繰り返す必要がなくなる．すなわち，各サイクルでの条件照合はルールの発火によって変化した WME に対してのみ行えば

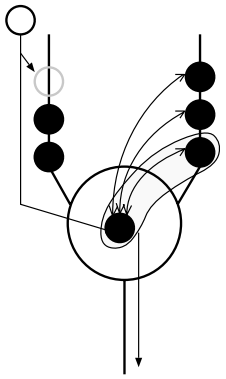


図 2.3: 2 入力ノードでの処理

良いことになる．実際の応用システムの測定によれば，サイクルごとに条件照合が必要となるのは RETE ネットワークのごく一部に過ぎない [23]．また，ES の規模が増大しても，サイクルごとに活性化される 2 入力ノード数は，それほど大きくならない．このことが，RETE マッチ・アルゴリズムを用いることにより大規模 ES でも条件照合を実用的な時間で処理できる大きな要因となっている．

ネットワークの共有

RETE ネットワークでは，異なるルール間でも，同一の照合条件のものは一つのノードとして共有する．このノードの共有により定数テストノードの数は 1/4〜1/9 に，2 入力ノードの数が 2〜3 割削減され，実行速度が 1.4 倍向上すると報告されている [23].

上述の通り，中間結果の保存による再計算の大幅な削減と，ネットワークの構造による同一条件照合の共有により，RETE マッチ・アルゴリズムは大幅な速度向上を実現している．しかしながら，これを用いてさえも，条件照合フェーズが推論時間のうち 90%以上を占める報告されている [23].

2.2 プロダクションシステム高速化手法

RETE マッチ・アルゴリズムは多くの ES 構築ツールで採用されているが，これを用いてさえも条件照合フェーズが推論の 90%以上もの処理時間を占めている．計画問題向けで認知–実行サイクルを多く繰り返すタイプの ES や，探索型でルー

ル発火により多数の WME が変化する ES では，条件照合フェーズの処理時間が原因で推論に非常に長い時間を要することになる．このため，条件照合フェーズを更に高速化し，推論時間をより短縮することが望まれている．

RETE マッチ・アルゴリズムの高速化法としては，RETE ネットワークの構造や個々のノードでの処理を最適化して処理量を減らす方法 [5, 43] と，複数のプロセッサで分割処理する並列化による方法 [21, 75] とがある．本論文では並列化による方法に焦点をあてており以降の節で詳述するので，本節では RETE ネットワークの最適化について概観する．

RETE マッチ・アルゴリズムを用いた推論インタプリタでは，α メモリ，β メモリを単純なリストとして実装しているものが多い．この場合には，条件間テストが 2 重ループで計算され，処理時間はトークン数の自乗に比例する．これを高速化するために，α メモリ，β メモリへのトークンの記憶を単純なリストとせずハッシュ表で実現する方法が提案されている [24]．ハッシュ表による実装を行うと，条件間テストの多い ES に対しては 2.5 倍〜3.5 倍程度の性能向上が得られると報告されている．

更に高度な手法として，RETE ネットワークをより効率的な構造に変形することが考えられている．ART[12] や YES/OPS[82] 等では，条件間テストの実行順序や 2 入力ノードの結合構造を ES 設計者が指定する方式を提案している．効率の良い構造を指定するためのヒューリスティックスとして次のようなものが知られている．

- トークン数の少ない条件要素から条件テストを実行する
- 変更量の少ない条件要素から条件テストを実行する
- RETE ネットワークの結合構造を共有する

しかしながら，ES 設計者にとって，これら 3 種類のヒューリスティックの相互作用を考慮しながら最適な RETE ネットワーク構造となるようなルールを記述することは容易ではない．この問題を解決するために，ES の動作特性に基づいて RETE ネットワークの構造を自動的に最適化する手法が提案されている [43]．この手法は，RETE ネットワークのコストモデルを構築し，ES の実行を測定して得られたパラメータに基づいてコストが最小となるように RETE ネットワークの結合を最適化する．これにより ES 設計者の試行錯誤によることなく，また，保守性

を損なわずに最適化が行われることになる．更に，人手の試行錯誤による最適化によりも良い性能が得られると報告されている．

2.3 条件照合処理の並列性と従来の並列化方式

本節では，RETE マッチ・アルゴリズムの処理の並列性について論じ，従来提案されている方式がどのような並列性をどの程度の粒度で利用しているか，すなわち従来方式の並列化軸における位置付けを示す．

RETE ネットワークはデータフローグラフととらえることができる．トークンがこのグラフを流れるデータであり，グラフの各ノードでは条件内テストや条件間テストなど，トークンあるいはトークンの組に対する比較演算が行われる．特に 2 入力ノードでは，それぞれの入力枝上のメモリノードに記憶されたトークンをペアとして比較が行われるため，比較演算を多数回繰り返すことになる．したがって，複数のトークンが同時に伝播する経路としての構造的な並列性，ノードにおいて多数のテストが同時に行われるノード処理の並列性の二つの軸が考えられる．以下では，この二つの並列性について詳述する．

RETE ネットワーク構造の並列性

RETE ネットワークには，1 個のルートノードがあり，ここがトークンの入口となる．ルートノードには，定数テストノードから構成される弁別ネットへの枝が接続されている．弁別ネットでは，すべてのルールのすべての条件要素に対する定数テストノードが含まれるため，ルートノードから弁別ネットへの枝は PM に含まれる全条件要素に相当する数だけに分岐していることになる．

弁別ネットの下段には 2 入力ノードと終端ノードから構成される結合ネットが接続されている．RETE ネットワークの 2 入力ノードは，二つの入力のうち一方は必ず弁別ネット内の定数テストノードに接続される．したがって，結合ネット最上段に位置する 2 入力ノードは二つの入力がともに弁別ネットに接続され，それ以下の段では，例えば左側の入力は 1 段上の 2 入力ノード，右側の入力は弁別ネットに接続されることになる (図 2.4)．RETE マッチ・アルゴリズムの特徴であるルール間でのノード共有のために，一つの定数テストノードが複数の 2 入力ノードへ接続されている場合がある．同様に，ある 2 入力ノードの下段には一つ以上の 2 入力ノードが接続される場合がある．

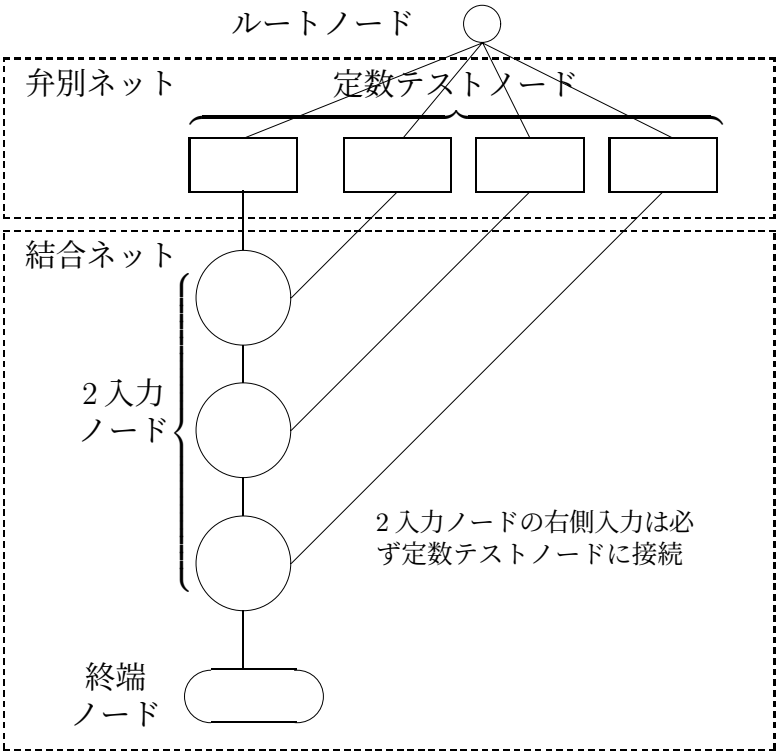


図 2.4: 2 入力ノードの連鎖

ルートノードからトークンが入力された後は，RETE ネットワークをデータフローグラフとしてトークンが流れることになる．このため，弁別ネットでは，ルートノードから分岐した枝の数だけ並列に処理が可能である．結合ネットでは，従属に接続されている 2 入力ノードの連鎖は逐次に処理される必要があるが，ある連鎖と別の連鎖は並列に処理できる．なお，2 入力ノードは二つの入力を持つため，両入力からのデータ到着を待つ同期処理が必要なように見えるが，次に述べるように，RETE マッチ・アルゴリズムの場合にはそのような同期は不要である．2 入力ノードでは，一方の入力よりトークンが入来し，そのトークンと他方の入力上のメモリノードに記憶されたトークンとの比較が行われる．すなわち，トークンが入来した際の比較相手となるトークンは既にメモリノードに記憶されているものだけである．したがって，トークンが入来し次第処理を開始することができ，同期は不要なのである．

以上のように，RETE ネットワークの構造に着目した場合，弁別ネットでは定数テストノードごとに，結合ネットでは 2 入力ノードの連鎖ごとに並列処理が可能である．ただし，このような並列処理可能な単位に基づいて単純に処理を分割

しプロセッサにマッピングしても，良い結果が得られるとは限らない．RETE ネットワークを分割するとノード間をつなぐ枝が切断される．この場合には，分割した処理の単位の間で通信が必要となり，通信オーバヘッドが生じるため，枝を多く切断するほど処理速度が低下する．また，一つのトークンのみに注目すれば 2 入力ノードの連鎖は逐次にしか処理できないが，認知–実行サイクルで複数トークンが発生したり上段の 2 入力ノードでの照合によって複数のトークンが発生する場合があり，この場合には従属に接続されたノードはパイプライン的に並列処理できる．したがって，このような場合には，2 入力ノードの連鎖も分割した方が，より多くの並列性を利用できる．

ノードでの処理の並列性

弁別ネットの各ノードでは，一つのトークンに対して，単純な比較が行われるだけである．このため，個々の定数テストノードでの処理には並列性はない．もっとも，その処理の単純さゆえ，弁別ネットでの処理時間が RETE マッチ・アルゴリズム全体の処理時間に占める割合は小さい．

弁別ネットを通過したトークンは、結合ネットの 2 入力ノードのどちらか一方の入力に伝播して来る．そして，もう一方の入力の直上にあるメモリノードに記憶されたすべてのトークンとの条件間テストが行われる．ここで，テストに成功するものがあれば，成功したトークンのペアの情報を持つ新たなトークンが生成され，より下段の 2 入力ノードへと伝播させられる．テストに成功するトークンがある限りペア情報を持つ新たなトークンの生成と伝播が続けられ，トークンが終端ノードに達するとインスタンシエーションとして競合集合に追加される．

上述のように，2 入力ノードではメモリノードに記憶されたトークンの数だけテストが繰り返されるため，これを並列に処理することが可能である．並列化に際しては，入来したトークンは，その 2 入力ノードを処理すべきすべてのプロセッサに複写される必要がある．一方，メモリノードに記憶されたトークンはどれかのプロセッサ上にあれば良く，また，テストも他のプロセッサからは独立に処理することができる．2 入力ノード処理の並列化では，テスト対象となるトークンは異なるものの，テストする内容は同一であるため，処理量のばらつきは小さいと考えられる．また，プロセッサ間の通信量は，メモリノードに記憶されるべきトークンのプロセッサへの蓄積方法及び配布方法に依存する．

従来の並列化方式

上記の考察から，それぞれの並列性を利用したいくつかの方式を容易に考えることができるし，実際にそれらの多くは既に提案されている．以下に，そのような並列化方式を列挙する (図 2.5).

ルール並列化方式

個々のルールを並列処理の単位とする自明な方式である．RETE ネットワークの構造的並列性の着目した方式の一種と位置付けることができるが，ネットワークの分割と同時に，複数ルールで共有されていたノードがそれぞれの処理単位に複写されることになる．これは，並列化に伴う処理量の増加，すなわち分割損をもたらす.

トップノード並列化方式

結合ネット最上段の 2 入力ノード (トップノードと呼ぶことにする) とそれに接続されたすべてのノードを並列処理の単位とする方式である [100]．トップノードに基づいて RETE ネットワークを分割することにより，結合ネットでは枝を切断することがなく，通信の必要がない．一方で，粒度はルール並列化方式よりも更に大きくなる．また，分割した結合ネットの複数に共有されている定数テストノードは，複写してそれぞれの処理単位に含ませる．これによって，ルール並列化方式同様に分割損が生ずる．ただし，弁別ネットの処理量は結合ネットでの処理量にくらべて圧倒的に小さいため，この分割損による処理時間の増加は小さいと予測される.

ノード並列化方式

個々のノードを処理単位とする方式である [26]．上記考察で述べたように，1 個のトークンだけの動作に着目すると，従属に接続された 2 入力ノードは逐次的に処理しなければならず，ノードにまで処理を分割する利点はほとんどないが，RETE ネットワーク内を複数のトークンが流れる場合には，このような 2 入力ノードの連鎖でもパイプラインとしての並列処理が可能である．ノードの入力と出力の枝がすべて切断されるため，トークンの伝播があるごとにプロセッサ間通信が必要となる.

ジョインスライス並列化方式

2 入力ノードにおけるトークンのテストのそれぞれを処理単位とする方式で

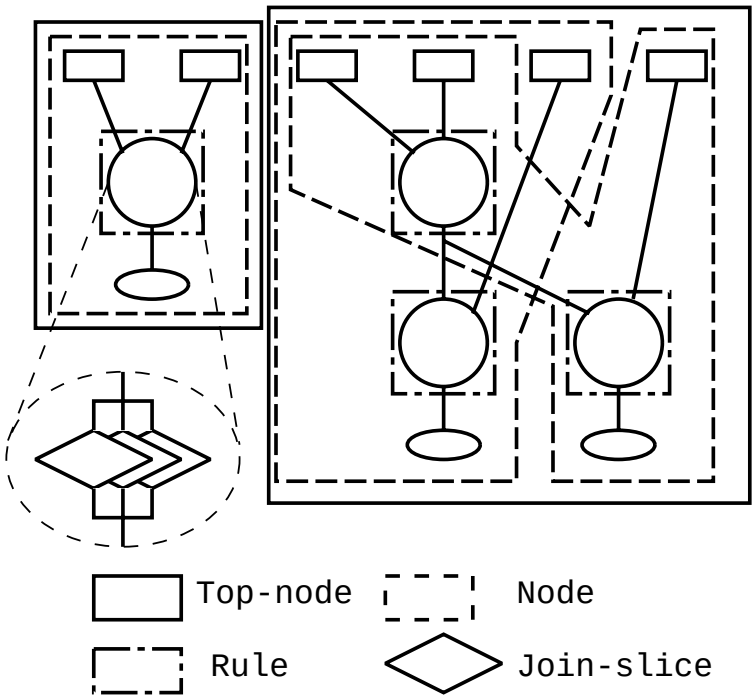


図 2.5: RETE マッチ・アルゴリズムの並列化方式

ある [53]．ノード処理における並列性を最大限に利用する方式で，メモリノードに記憶された個々のトークンに対して 1 個ずつのプロセッサを割り付ける．トークンが伝播して来た場合には，これが複写されて各プロセッサに入力される．各プロセッサは，入来したトークンと自分が持つトークンとのペアに対するテストを 1 回だけ行い，成功すればトークンのペアの情報を含む新たなトークンを生成して下段のノードが割り付けられたプロセッサに流す．構造的にはノード並列化方式と同様に入力と出力の枝がすべて切断されており，トークンの伝播がプロセッサ間通信の機会となるが，枝そのものが照合対象となるトークンの数だけ分岐することになるため，通信量は更に多くなる．

2.4 RETE ネットワークのモデル化による並列化方式の評価

前節で列挙した RETE マッチ・アルゴリズム並列化方式のなかには，既に提案されその性能が報告されている方式も多い．しかし従来の報告での評価手法は，特定の ES を対象としたシミュレーション評価によっており，報告ごとにサンプル

ESが異なるため，並列化方式相互の比較が困難であった．RETEマッチ・アルゴリズムの並列化による速度向上度は，対象となるESの問題特性に大きく依存するため，並列化方式を相互に比較するためにはこの問題特性を一致させる必要がある．また，問題への依存性も比較するために，広範囲のES特性にわたる評価が望まれる．

本節では，RETEマッチ・アルゴリズム並列化方式の速度向上度を解析的に評価する手法を提案する．本評価法では，RETEネットワークをモデル化し，このモデルに基づいて並列度や処理量をESの問題特性で表現する．これを用い，与えられたES 問題特性に対する速度向上度を計算する推定式を導出する．この推定式を用いて並列化方式の比較を行い，各方式の適用領域，ハードウェアの要求条件を明確化する．

2.4.1 エキスパートシステムの問題特性

ここでは，並列化方式の速度向上度に影響を与えるES問題特性を抽出する．並列化方式はRETEネットワークの構造とノードでのデータ処理に基づいて処理を分割しているため，これらに影響するパラメータが必要である．ただし，パラメータから構造や処理内容が完全に特定できる必要はなく，速度向上度の推定が可能となるような統計的な数値が得られれば十分である．

このような観点からES問題特性として表2.1及び表2.2に示すパラメータを抽出した．表2.1はRETEネットワーク構造の推定に必要なパラメータ，表2.2はノードでの処理特性の推定に必要なパラメータである．

構造を支配する問題特性パラメータ

個々の条件要素に対し定数テストノードが生成されるため，ルール数とルールあたりの条件要素数の平均値がわかれば弁別ネットでのノード数が推定できる．また，個々のルールに対し従属接続される2入力ノードの数は，条件要素数に依存する．異なるルールに同一の条件テストがある場合，それに対応するノードは共有される．したがって，ルール数Rと1ルールあたりの条件要素数E，そして，同一条件内テストの比率S_cがわかれば，ノード数や接続枝数などのRETEネットワークの構造に関する値を推定できる．

動作を支配する問題特性パラメータ

ルールあたりのアクション数Aは，一つのルールに含まれるWME操作アク

表 2.1: RETE ネットワークの構造を支配する問題特性パラメータ

パラメータ	記号
ルール数	R
ルールあたりの条件要素数	E
同一定数テストの比率	S_c

ションの数の平均値である．また，アクション部更新動作比率 β_m は，WME 操作アクション中の WME 更新動作の割合である．WME の生成操作及び削除操作はそれぞれトークンを 1 個しか生成しないが，WME 更新操作では，トークンが 2 個 (旧い内容の WME の削除と新たな内容の WME の生成) 発生するため，1 認知–実行サイクルあたりのトークン発生数を推定するために β_m は必要である．クラス数 L は ES に含まれるクラスの種類である．また，WME の数は，生成，削除動作により推論中に変化するが，ここではその平均値を用いる．これらは，ES ソースプログラムの分析，ES 設計段階での見積によって得られる．

条件内テストの成功確率 α_c 及び条件間テストの成功確率 α_j は，個々のノードで異なるはずであるが，ここでは RETE ネットワークの全体的な動作特性が推定できれば十分なため，一定値と仮定する．α_c，α_j は ES の設計に大きく依存し，正確な値を得るには実際に ES を動作させ測定する必要があるが，設計時に見当がつく認知–実行サイクルごとの照合成功ルール数から概算する事もできる．一般には，条件内テストを緩くし条件間テストで最終的に絞り込むので，α_c は 10〜数十%，α_j は 1%以下といわれている [23].

以上のパラメータから，RETE ネットワーク内で活性化されるノード数，活性化されたノードにおける条件テスト回数などを推定することができる．

2.4.2 RETE ネットワークのモデル化

RETE ネットワークを構造的にも動作的にも特徴づけているのは 2 入力ノードである．2 入力ノードは，定数テストノードを通過したトークンとメモリノードに蓄積されたすべてのトークンとの条件間テストを行うため，処理量が他ノードに比べ非常に大きい．更に，あらゆる種類のノードは必ず 2 入力ノードに接続され

表 2.2: RETE ネットワークの動作を支配する問題特性パラメータ

パラメータ	記号
ルールあたりのアクション数	A
実行部の更新動作比率	β_m
WME 数	W
クラス数	L
条件内テストの成功確率	α_c
条件間テストの成功確率	α_j

るため，このノードの特性が推定できれば他ノードの特性も付随的に求まる．そこで，ここでは 2 入力ノードのふるまいに着目し，RETE ネットワークのモデルを構築する．

代表的な ES についての分析 [23] に基づけば，以下の事実が導き出される．

事実

- 2 入力ノードの出力枝の分岐数は比較的少数
- ルールあたりの条件要素数の度数分布は条件要素数の増大に伴い指数関数的に減少

この事実より，2 入力ノードを以下のようモデル化する．

2 入力ノードのモデル

- 出力枝の一定の分岐率 B を持つ
- 一つのノードは確率 P で終端する

このモデルによれば，図 2.6 のように，出力が平均 B 本あり，そのノードから出る枝の 1 本が一定確率 P で終端ノードに接続されることになる．したがって，従属接続される 2 入力ノードは一段毎に $(B-P)$ の割合で減少する．すなわち，2 入力ノードの従属段数 (条件要素数－ 1 に等しい) の度数分布は指数関数的に減少し，上記事実に良くあう．このモデルを規定するパラメータ B 及び P と問題特性との関係を以下に示す.

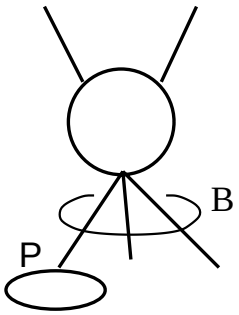


図 2.6: 2 入力ノードのモデル

E と P の関係

1 段目の 2 入力ノード数を d_o としたときに，n 段目で終端するノードの数 d_n はこのモデルより $d_n = d_o P(B-P)^{n-1}$ となる．これより，全終端ノード数 D_t は，

$$D_t = \sum_n d_n = \sum_n d_o P(B-P)^{n-1} = d_o \frac{P}{1-(B-P)} \tag{2.1}$$

となり，その ES における従属段数の平均値 e は，

$$e = \sum_n n \frac{d_n}{D_t} = \sum_n n \frac{d_o P(B-P)^{n-1}}{d_o \frac{P}{1-(B-P)}} = \frac{1}{1-(B-P)} \tag{2.2}$$

で計算できる．ところで，$E = e + 1$ であるから，

$$P = (B-1) + \frac{1}{E-1}. \tag{2.3}$$

ルールあたりの条件要素数の平均 E は ES のプログラムを見れば容易に得られるから，残る分岐数 B が決まれば RETE ネットワークの特性は一義的に決定できる．B は ES のルール記述を分析すれば正確に求まるが，一般には，共通部分が少ないもので 1.1，共通部分が多いものでも 1.9 程度と報告されている [23].

2.4.3 RETE ネットワークの諸特性

ここでは，並列化による速度向上推定のために必要となる RETE ネットワークの構造的特性及び処理量特性を導出する．構造的な特性としては以下のものがある．

定数テストノードの総数 D_1: 弁別ネットには定数テストノードが含まれるため，これが弁別ネットでのノード総数となる．

2 入力ノードの総数 D_2**:** 結合ネットは，2 入力ノードと終端ノードから構成されている．終端ノードはルール数に等しいことは自明であるし，最下段の 2 入力ノードに付随するものと考えることができる．したがって，2 入力ノードの数が結合ネットの構造を支配することになる．

定数テストノードから 2 入力ノードへの接続枝数 S**:** 弁別ネットと結合ネットとを接続する枝の数である．粒度が大きく結合ネットの枝を切らない分割をするルール並列化方式やトップノード並列化方式でも，弁別ネットと結合ネットとを結ぶ枝は切断される．したがって，通信オーバヘッドを推定するために，この枝数の推定が必要となる．

また，処理量特性としては以下のものがある．これらは，並列化のために RETE ネットワークを分割した際のそれぞれの処理単位の処理量を支配する．処理量ばらつきが大きいと，プロセッサ間で負荷の不均衡が生じて速度向上を阻害するし，分割損によって総処理量が増加しても速度が低下する．したがって，並列化による速度向上を推定するために，これら特性が必要となる．

活性化される定数テストノード数 O_1**:** 認知–実行サイクルの実行フェーズにおいて発生したトークンは，条件照合フェーズにおいて RETE ネットワークのルートノード入力される．ルートノードからは弁別ネットへトークンが流れ込み，そのトークンが持つクラスと同一のクラスの条件内テストを行う定数テストノードが活性化される．O_1 は，1 個のトークンによって活性化される定数テストノードの延べ数を表す．

弁別ネットを通過するトークン数 T_t**:** 活性化された定数テストノードにおいて条件内テストが行われ，テストに成功したトークンが結合ネットへと流される．T_t は，この定数テストノードでテストに成功するトークンの数を示す．

活性化される 2 入力ノード数 O_2**:** 定数テストノードを通過したトークンは，それが接続された 2 入力ノードを活性化する．また，2 入力ノードでテストに成功すると，新たなトークンが生成され，更に下段の 2 入力ノードを活性化する．O_2 は，1 個のトークンの入力によって活性化される 2 入力ノードの延べ数を表す．

2 入力ノードでの平均テスト回数 j**:** 2 入力ノードでは，入力されたトークンと，対向する入力上のメモリノードに記憶されたトークンそれぞれとをペアとして

条件間テストを行う．したがって，2 入力ノードでは，条件間テストをメモリノードに記憶されたトークンの数だけ繰り返すことになる．j は，この条件間テストの回数を示す．

認知–実行サイクルあたりの発生トークン数 M**:** 上記の特性値は，1 個のトークンによって活性化されるノード数やテストの回数であった．1 回の認知–実行サイクルでは，複数のトークンが発生する場合が多い．1 個の WME に対して更新が行われるとトークンが 2 個発生 (旧い内容の WME 削除と新たな内容の WME 生成を行ったと等価のため) するし，ルールのアクション部には任意数のアクションを記述できるため，1 回のルール発火でいくつもの WME が生成・更新・削除される場合もある．M は，1 回のルール発火あたりのトークンの発生個数を表す．

これまで述べてきた ES 問題特性と 2 入力ノードのモデルとを用いて，これらの RETE ネットワーク特性を導出する．ただし，速度向上の計算には，処理量の平均，分散など統計量がわかれば十分なため，導出に際し以下のような近似を行った．

近似

- 各クラスに対する条件要素の数は均一とする．また，トークンもこれに対し均等に発生することとする．
- 各ノードの照合成功率は一定とする．

RETE ネットワーク特性は以下のように導出される．

定数テストノードの総数 D_1

ルール間で共通の定数テストがなければ，条件要素の総数に等しいので $R \cdot E$ である．実際には S_c だけ共通化されるので，

$$D_1 = R \cdot E(1 - Sc) \tag{2.4}$$

となる．

2 入力ノードの総数 D_2

まず，m 段目の 2 入力ノードのノード数を $D_{2(m)}$ とすると，この 2 入力ノー

ドのうち終端ノードに接続されている数 b_m は，ノード終端の確率が P であることから，

$$b_m = D_{2(m)}P \tag{2.5}$$

となる．これを全段にわたって総和した数はルール数 R に等しいから，

$$R = \sum b_m = P \sum D_{2(m)} = PD_2 \tag{2.6}$$

となる．2 入力ノードの総数 D_2 に対して解いて，

$$D_2 = R/P \tag{2.7}$$

を得る．また，

$$D_{2(m)} = D_2\{1 - (B - P)\}(B - P)^{m-1} \tag{2.8}$$

である．

定数テストノードから 2 入力ノードへの接続枝数 S

RETE ネットワークでは 2 入力ノードの右側入力は必ず定数テストノードに接続されている．また 1 段目では左側入力も定数テストノードに接続される．したがって

$$S = D_{2(1)} + \sum D_{2(m)} \tag{2.9}$$

となり，$D_{2(1)}$ を D_2 で表わして代入すれば，

$$S = D_2\{2 - (B - P)\} \tag{2.10}$$

を得る．

活性化される定数テストノード数 O_1

1 個のトークンによって活性化される定数テストノードは，入力されたトークンと等しいクラスをテストするノードの数に等しい．したがって，全定数テストノードをクラス数で除した値，

$$O_1 = D_1/L \tag{2.11}$$

となる．

弁別ネットを通過するトークン数 T_t

活性化した定数テストノードでのテストは α_c の確率で成功するので，テストに成功する定数定数テストノード数は $O_1 \cdot \alpha_c$ となる．定数テストノードは平均 S/D_1 の分岐を持ち，これらすべてにトークンが配布されるから，弁別ネットを通過し結合ネットへと入力されるトークン数は，

$$T_t = O_1 \cdot \alpha_c \frac{S}{D_1} = \frac{S \cdot \alpha_c}{L} \tag{2.12}$$

となる．

活性化される 2 入力ノード数 O_2

弁別ネットからトークンが結合ネットへと伝播することにより T_t 個の 2 入力ノードがまず活性化される．個々の 2 入力ノードには，従属に平均 $E-1$ 段の 2 入力ノードが接続されている．下段のノードは必ずしも活性化されるとは限らないが，ここでは，平均的には全段が活性化されるとして，

$$O_2 = T_t(E-1) \tag{2.13}$$

とする．

平均照合回数 j

2 入力ノードでの照合回数は，そのノードの入力上のメモリノードに記憶されたトークン数に等しい．これは推論を通して変化するが，弁別ネットを通過したトークンは必ずメモリノードに記憶されるため，全推論過程での平均を考えれば，全発生トークンのクラスあたりの数 (クラスあたりの WME 数に等しくなる) と条件内テスト成功率との積，

$$j = \frac{W}{L}\alpha_c \tag{2.14}$$

になる．

認知–実行サイクルあたりに発生するトークン数 M

トークンは生成及び削除操作で 1 個，更新操作で 2 個発生するため，平均実行要素数と 1 実行要素あたりのトークン発生数との積，

$$M = A(1+\beta_m) \tag{2.15}$$

となる．

表 2.3: RETE ネットワークの構造的特性

項目	記号	計算式
定数テストノードの総数	D_1	$R \cdot E(1 - S_c)$
2 入力ノードの総数	D_2	R/P
定数テストノードから 2 入力ノードへの接続枝数	S	$D_2\{2 - (B - P)\}$

表 2.4: RETE ネットワークの処理量特性

項目	記号	計算式
活性化される定数テストノード数	O_1	D_1/L
弁別ネットを通過するトークン数	T_t	$S \cdot \alpha_c/L$
活性化される 2 入力ノード数	O_2	$T_t(E - 1)$
2 入力ノードでの平均照合回数	j	$(W/L)\alpha_c$
1 認知サイクルで発生するトークン数	M	$A(1 + \beta_m)$

以上の結果を表 2.3 及び表 2.4 にまとめる．これらの値を用いることにより，平均活性化ノード数や各ノードでの平均テスト回数，認知サイクルでの総テスト回数などが計算できる．例えば，1 認知-実行サイクルあたりの結合ネットでの総テスト回数は， $O_2 \cdot j \cdot M$ と計算できる．

2.4.4 速度向上度の推定式

ここでは，上記の RETE ネットワークのモデルに基づいて，ES 特性と並列化方式を代入するだけで速度向上度を推定できる一般式を導出する．

認知–実行サイクルを逐次処理した場合の処理時間を T_0 とすると，並列化時の同サイクルの処理時間 T は，

$$T = \frac{QT_0}{N}\gamma + T_C \tag{2.16}$$

で表される．ここで，それぞれの記号は

N: プロセッサ数

Q: 処理の分割による総処理量の増加係数

γ: 負荷ばらつきによる処理時間の増加係数

T_C: 通信オーバヘッドによる処理時間の増加

を表す．ところで，RETE マッチ・アルゴリズムの処理は，ノードでの照合とその結果であるトークンの伝播であるため，トークン伝播に伴うデータ転送量は各プロセッサで処理するトークン数とノード数とに比例すると考えられる．したがって，通信オーバヘッドによる処理時間の増加 T_C は，プロセッサあたりの処理量に比例し，

$$T_C = \frac{QT_0}{N} C_o \tag{2.17}$$

と書ける．C_o は並列化方式に固有の通信オーバヘッド係数である．したがって処理時間は，

$$T = \frac{QT_0}{N}\gamma + \frac{QT_0}{N}C_o = \frac{QT_0(\gamma + C_o)}{N} \tag{2.18}$$

となり，速度向上度 U は，

$$U = \frac{T_0}{T} = T_0 \frac{N}{Q \cdot T_0(\gamma + C_o)} = \frac{N}{Q(\gamma + C_o)} \tag{2.19}$$

となる．処理量増加係数 Q は，RETE ネットワークをどのように分割したかにより容易に決定できる．例えばルール並列化方式の場合，オリジナルの RETE ネットワークでは，定数テストノードと 2 入力ノードとの接続枝は S 本であるが，定数テストノードを共有しないことにより $R \cdot E$ 本に増加する．定数テストノード数も 2 入力ノード数もともにこの枝数に比例して増加し，処理量も同様に増加するから，この場合は，$Q = RE/S$ となる．

また，通信オーバヘッド係数 C_o は，認知–実行サイクルあたりの総テスト回数に対する通信回数の割合を C_R，一対のトークンのテスト時間に対する 1 トークン転送時間の比を C_C とすると，

$$C_o = C_R \cdot C_C \tag{2.20}$$

となる．

つぎに，負荷ばらつき係数 γ について考える．通信オーバヘッドを除いた並列処理時間 T_P は最も処理時間のかかるプロセッサにより支配され，

$$T_P = \max(T_1, T_2, \cdots, T_N) \tag{2.21}$$

となる．ここで，T_i は PE_i の処理時間を表す．負荷ばらつき係数は，負荷均等と想定した処理時間と T_P との比と考えられるから，

$$\gamma = \frac{\max(T_1, T_2, \cdots, T_N)}{\sum T_i / N} \tag{2.22}$$

となる．T_i について詳細化すれば，RETE ネットワークの諸特性を用いて γ を定式化できることがわかる．

いま，分割した個々の処理 g_j の処理時間を t_j とする．これらの中には相互に逐次的に処理されなければならないものもあるから，そのような処理を集めて並列化の処理単位を構成する．それらの処理単位を $G_1, G_2, \ldots, G_m$ と表す．また，G_j の処理時間を T_{Gj} と書く事にする．処理単位をプロセッサに均等に割り付ければ，プロセッサあたりの処理単位の数は m/N 個となるから，あるプロセッサでの処理時間 T_i は m/N 個の T_{Gj} を加算したものとなる．すなわち，

$$T_i = \sum_{j=x}^{j=x+m/N} T_{Gj}, \tag{2.23}$$

ただし，$x = (i-1)m/N + 1$ である．この定式化に基づき，以下の手順で γ が計算できることになる．

1) モデルにより T_{Gj} の分布を計算．例えば，ルール並列化方式の場合，処理時間は処理単位に含まれる 2 入力ノードの数に比例すると考えられるので，2 入力ノード従属段数と相似な分布 (指数分布) となる．

2) 上で求めた分布を持つ母集団から m/N 個取り出した標本値の総和の分布 (すなわち T_i の分布) を求め，その平均値 $E(T_i)$ を計算 [52].

3) T_i なる分布を持つ母集団から N 個の標本を取り出した場合の最大値の分布を求め，その平均値 $E\{\max(T_i)\}$ を計算．

4) $\gamma = E\{\max(T_i)\}/E(T_i)$ を計算．

T_{Gj} の分布を解析的に求めることができれば，以上の手順により γ も解析的に計算できる．また，モデルから直接に分布が求まらなくとも，実際の ES を動作させて処理時間を測定し，既知の分布関数に近似してやることで，以後は解析的に計算することが可能である．

なお，γ は定性的に以下の性質を持つ．

表 2.5: 並列化方式と好適なアーキテクチャ

並列化方式	処理単位	アーキテクチャ
ルール	個々のルール	放送機能を持つ (トークンを全 PE に配布するため) バス結合型
トップノード	トップノードとそれに連なるすべてのノード	放送機能を持つ (トークンを全 PE に配布するため) バス結合型
ノード	個々のノード	ツリー，メッシュなどのネットワーク結合型
ジョインスライス	ノード内での個々のトークン照合	トークン配布，収集機構を持つ階層的ネットワーク結合型

- 処理単位の数が多ければ処理時間が平均化されるため，処理量の分布や逐時性に関わらず γ は小さくなる．逆に，並列性が小さければ，処理量分布，逐次性が γ に与える影響は顕著となる．
- 処理単位間の逐次性が大の場合には，G_j が含む処理の数が多くなり，処理単位内での処理量ばらつきが平均化されるため，γ は逐時性に支配される．
- 逐時性小の場合は処理単位の処理量分布が大きく影響する．

2.4.5 各並列化方式の解評価析と考察

本節では，前述した並列化方式のそれぞれに対し，速度向上推定式の各項を計算し，速度向上度の比較を行う．各方式に対して，それぞれ好適と考えられる並列プロセッサのアーキテクチャを仮定し，そのアーキテクチャ上での速度向上度を評価する．並列化方式とアーキテクチャの対応を表2.5 に示す．並列化方式とアーキテクチャから，速度向上推定式の各項は表 2.6 の様に求める事ができる．

ルール並列化方式とトップノード並列化方式の処理量は処理単位に含まれる 2 入力ノード数に比例とすると考えられるため，処理量の分布は 2 入力ノード数の分布と相似になる．これはまた，ルールあたりの条件要素数の分布と相似でもある．逐次的に処理されるものは処理単位内に閉じているため，処理単位間での逐次性はない．また，通信はトークンが RETE ネットワークに入力された際に，それを全プロセッサに配布するために必要となるだけである．放送機構をもったバ

表 2.6: 速度向上度推定式の項の値

	トップノード	ルール	ノード	ジョインスライス
処理単位の数 n	T_t/L	$Q \cdot T_t/L$	$T_t \cdot M/L$	$T_t \cdot M \cdot j/L$
処理量増加係数 Q	$\simeq 1$	$R \cdot E/S$	1	1
処理量分布	処理単位あたりのノード数の分布に相似	処理単位あたりのノード数の分布に相似	ノード内での比較回数の分布に相似	ほぼ均一
逐次性	なし	なし	ノードの従属数	無視可
通信比率 C_R	$\simeq 0$	$\simeq 0$	$1/j$	2

スによりプロセッサを結合してあれば，これは一度の放送だけで済むため，通信オーバヘッドはほとんど無視できる．

ノード並列化方式では，処理量のばらつきはノード内での比較回数のばらつきとノードの接続関係による逐次性とに起因する．前段の 2 入力ノードの処理が完了しない限り次の段の 2 入力ノードの処理は開始できないため，ノードの逐次性の方がより深刻であり，処理量のばらつきもこれが支配的である．通信は個々のノードの処理ごとに 1 回必要となる．1 ノードあたり j 回のテストが行われるのであるから，総テスト回数に対する通信回数の比率は，$1/j$ となる．

ジョインスライス並列化方式は個々のテストが処理単位であるから処理量ばらつきはほとんどない．一方，テストを行うために他のプロセッサからトークンを受信し，テスト結果を他プロセッサに送信する必要があるため，総テスト回数の 2 倍の回数の通信が必要となる．

以上により速度向上度を推定するのに必要な項がすべて導出できるので，これを用いて各並列化方式を評価する．まず，適当なパラメータを持つ ES を想定し，それに対する条件照合フェーズの速度向上度を計算して，各並列化方式の実用性能とスケーラビリティを概略的に調べる．次に，ES パラメータを変化させ，問題特性に対する速度向上度の感度を評価する．

図 2.7 は，表 2.7 に示すパラメータを持った中規模 ES に対して条件照合フェーズの速度向上度を計算した結果である．ノード並列化方式やジョインスライス並

表 2.7: 評価に用いた ES 特性

R : 300	A : 2.0	α_c : 0.35
E : 3.5	β_m : 1.0	α_j : 0.07
L : 7	B : 1.2	
W : 250	S_C : 0.3	

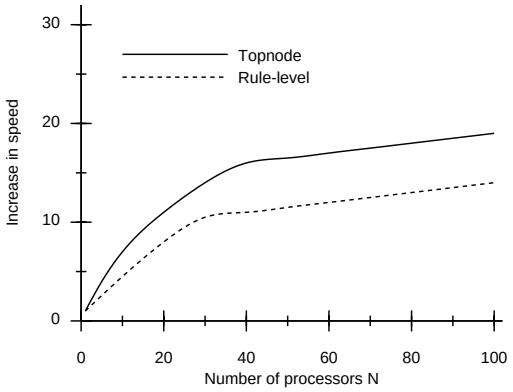
列化方式の細粒度方式に対しては，1 個のトークンの転送時間（1 トークン転送時間の 1 トークン照合時間に対する比 C_c）を変化させている．この値と表 2.6 に示した通信発生の割合 C_R との積が，通信オーバヘッド係数 C_o となる．

条件照合フェーズの消費する時間が推論全体の約 90%程度とすると，条件照合フェーズをいかに高速化しても，推論全体の速度向上度は 10 倍程度が限界となる．このことから，条件照合フェーズとしても，10 倍〜数十倍程度の速度向上が意味のある高速化であるといえる．この観点で図 2.7 をみると，トップノード並列化方式は辛うじて意味のある速度向上が達成できる事がわかる．また，ルール並列化方式は処理量増加が原因となりトップノード方式より常に速度向上が小さい．

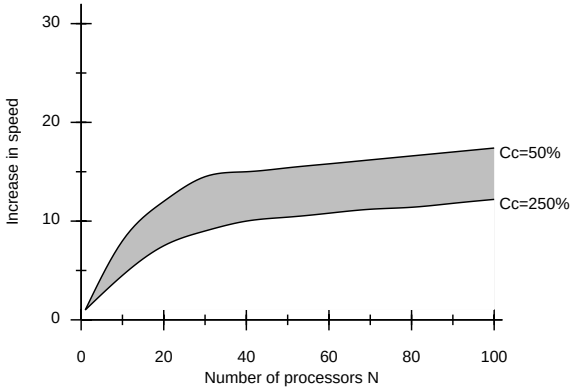
より粒度の細かいノード並列化方式は分割した処理の数が多いため一見より大きな速度向上が得られるように思える．しかし実際には，従属接続されたノードの逐次性により並列処理できる処理単位が限定されてしまうため，グラフからわかるようにトップノード並列化方式などの粗粒度の方式と同程度の性能しか得られない．ただし，認知–実行サイクルごとに多数のトークンが発生する ES では，従属接続ノードをパイプライン処理できるので，より大きな速度向上が期待できる．また，この方式は相互結合網のデータ転送性能が低いと性能が低下する．

ジョインスライス並列化方式は，高速のデータ転送が可能ならば非常に大きな速度向上が期待できる．しかし，データ転送性能が低い場合には，トップノード方式よりも速度向上が小さい．一対のトークン比較は数マシンステップで済むため，この方式の性能を十分に引き出すには，トークンの転送も数マシンステップで行うことが可能な相互結合網が要求される．

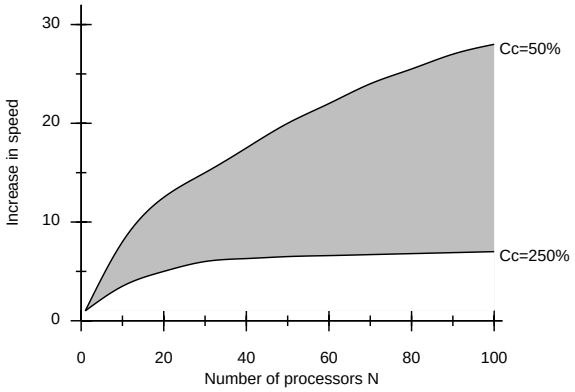
ES のパラメータを変化させた場合の速度向上度の変動範囲を図 2.8 に示す．変化させたパラメータは，RETE ネットワーク構造への影響が大きく，速度向上の規定要因となるものである．他のパラメータは，その変化が推論の絶対時間に影響するが，並列化による速度向上にはほとんど影響しないと考えられる．この図



(a) トップノード方式，ルール方式
(a) Topnode scheme and Rule-level scheme



(b) ノード方式
(b) Node-level scheme



(c) ジョインスライス方式
(c) Join-slice scheme

図 2.7: 解析評価による速度向上特性

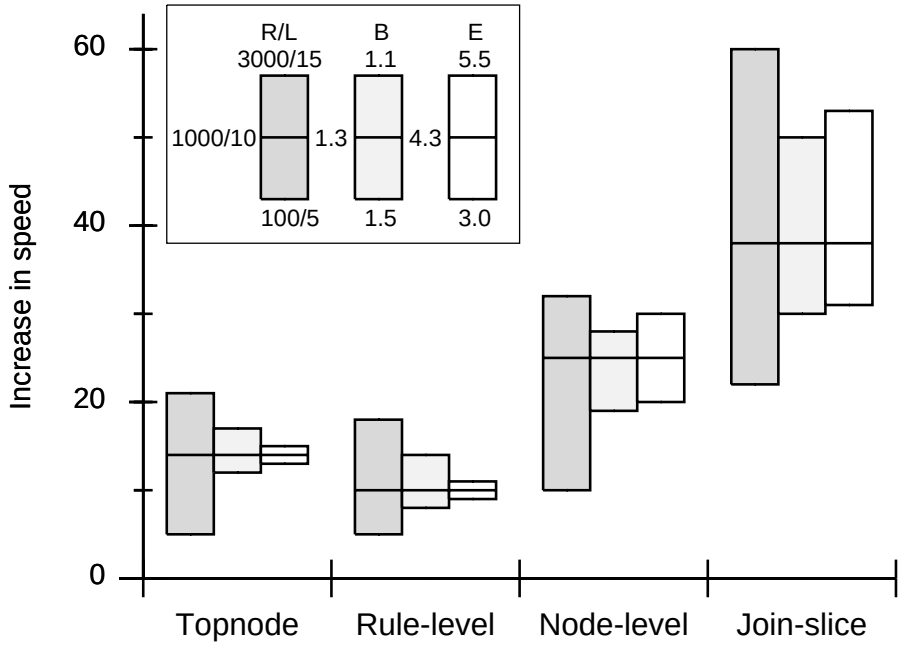


図 2.8: 速度向上の問題依存性

から，並列化方式の速度向上はルール数 (すなわち ES 規模) に最も大きく影響される事がわかる．また，小規模領域ではトップノード方式やルール方式などの粗粒度方式は数倍の速度向上度しか得られないことがわかる．これは処理単位の数が不足してプロセッサの負荷ばらつきが増加するためである．すなわち，ルール数が十分に大きい領域 (おおよそ 1000 ルールを超えるような規模の領域) にしか，これら粗粒度方式は適用できない．ジョインスライス方式は，本質的には，あらゆる ES 特性領域で十分な速度向上が期待できるが，前述のように，プロセッサ間のデータ転送速度が遅い場合は，性能が急激に低下する．

以上から，ルール並列化方式やトップノード並列化方式などの粗粒度方式では，通信の発生頻度が低いため，バス結合型などの低廉にハードウェアを構成可能なアーキテクチャでも並列化方式の持つ本質的な速度向上を実現できることがわかる．しかし，その速度向上度の絶対値として，ルール数が 1000 以上と大きい場合には意味のある値となるが，そうでなければ満足な性能は期待できない．一方，ジョインスライス並列化方式のような細粒度方式では，どのような ES 特性に対しても十分な速度向上が期待できるが，実際にこれを達成するためには，非常に高速な相互結合網が必要となることがわかる．このようなハードウェアは高価となる．

2.5 TWIN 方式の提案

前節の評価結果から，従来方式では，安価に構成できるハードウェアでは限定的な ES 特性の領域でしか十分な速度向上が得られないことがわかった．逆に，広範囲の ES 特性に対して十分な速度向上を得ようとすれば，高価なハードウェア構成となってしまうことがわかった．前述のとおり条件照合フェーズ並列化による速度向上度は，その目標を 10 倍〜数十倍に置いている．この程度の速度向上を得るために，非常に高速な相互結合網を持つ高価なハードウェアを用いるのは得策ではないだろう．比較的安価に構成できるハードウェア構成で，この目標を広範囲の ES 特性に対して達成できる並列化方式が望まれる．本節では，このような並列化方式として TWIN 方式を提案する．

まず，RETE マッチ・アルゴリズムの中核を担っている 2 入力ノードについて，詳細に考察する．この考察に基づき，ジョインスライス並列化方式のように照合ごとに通信を要求することなく，2 入力ノードを多数のプロセッサで処理できる方法を示す．そして，この方法を利用し，通信発生の割合を小さく抑えたまま構造的な並列性と 2 入力ノード処理の並列性の両方をひきだす，TWIN 方式について詳述する．

2.5.1 2 入力ノードに関する考察

RETE ネットワーク及び各ノードでの処理は 2.1.3 でも説明したが，ここでは 2 入力ノードの構造及び処理についてより詳細に述べ考察する．

2 入力ノードは弁別ネットの下位に位置して結合ネットを構成し，ルールごとにいくつかの連鎖があり，最下段は終端ノードに接続されている．2 入力ノードでは，ルールの条件要素にまたがるテスト (条件間テスト) が行われる．より具体的には，ある条件要素の条件内テストを通過した WME と別の条件要素の条件内テストを通過した WME のペアに対して，両 WME で同一のパターン変数が同一の値に束縛されているかをテストする．2 入力ノードでのテストを通過した場合，その WME ペアの情報をすべて含むトークンを生成して下段に位置する 2 入力ノードに伝播させる．したがって，この下段の 2 入力ノードでは 3 つの WME に含まれるパターン変数の一致をテストできることになる．このような連鎖構造により，ルールの条件要素が E 個あった場合に，これから二つ取った組合わせの数は $E(E-1)$ もあるにもかかわらず，$E-1$ 個の 2 入力ノードですべての条件間テストを行うこ

とができる．条件要素の中にはパターン変数を持たないものもあり，この場合は条件間テストは不要である．しかし，上記の連鎖構造を成立させるためには，このような条件要素に対しても 2 入力ノードが必要となる．2 入力ノードは，パターン変数の一致をテストする以前に WME の組を生成する機能も担っているからである．このような条件要素に対しては，あらゆるトークンの組に対してテストが成功する 2 入力ノードが対応する．このため，パターン変数の多寡にかかわらず，2 入力ノードの連鎖は，必ず $E-1$ 個のノードを含む．

このような連鎖構造とするために，2 入力ノードの一方の入力 (例えば図 2.4 では右側の入力) は必ず WME1 個のみの情報が伝播して来る弁別ネットに接続されている．すなわち，最上段に位置する 2 入力ノード (これを特にトップノードと呼ぶ) は両入力とも弁別ネットに接続され，2 段目以降の 2 入力ノードは一方の入力が上段の 2 入力ノード，他方の入力が弁別ネットに接続されるという構造となっている．

次に，図 2.3 を参照して，トークンが到来した場合の 2 入力ノードでの動作を詳述する．ここでは，図にあるように，左側入力からトークンが到来する場合について述べているが，左右を入れ換えれば右側入力から到来した場合でも同じとなる．

1. 左側の入力にトークンが到来する
2. トークンを左入力直上のメモリノードに記憶する
3. 右入力直上のメモリノードから記憶されたトークンを一つ取出す (ただし取出したトークンをメモリノードから取除くわけではない)
4. 到来したトークンと取出したトークンとをペアとして，この 2 入力ノードに割り付けられた条件間テストを行う
5. テストが成功すれば，そのペアの情報を含む新たなトークンを生成し出力に流す
6. 右入力直上のメモリノードから記憶されたすべてのトークンを取出してテストするまで 3〜5 を繰り返す．

このように，2 入力ノードにおいて，到来したトークンとのペアを作ってテスト対象となるのは，メモリノードに記憶されたトークンである．したがって，2 入力ノードでの処理量は，メモリノードに記憶されたトークンの数に支配されることになる．

ところで，メモリノードに記憶されたトークンというのは，最初からそこに記

憶されていたわけではない．上の 2 番目のステップにあるように，トークンが到来した際に記憶されたものである．すなわち，右側入力直上のメモリノードに記憶されているのは，いずれかの時点で右側の入力に到来したトークンなのである．上の動作からわかるように，メモリノードへの記憶と条件間テストとは，必ずしも不可分ではない．つまり，到来したトークンに対し，メモリノードへの記憶を行わずに，条件間テストを行うことも可能である．もちろん，逐次処理でこのようなことをすれば，本来メモリノードに記憶されなければならないトークン，すなわち条件にマッチした WME，が照合の対象から洩れてしまうことになり，正しい結果が得られなくなってしまう．しかし，このことは，2 入力ノードでの処理を並列化しようという場合には重要な示唆を与える．メモリノードに記憶されたトークンを複数のプロセッサに分散させることができれば 2 入力ノードを並列に処理することができる．そしてこのトークン記憶の分散は，メモリノードへの記憶を制御することにより実現できそうに見える．

2.5.2　2 入力ノード処理の並列化

上記考察から，メモリノードへの記憶動作を適切に制御すれば，2 入力ノードでの照合処理を複数のプロセッサに分散させることが可能になると予測できる．ここでは，2 入力ノード処理並列化のための具体的な記憶制御法及び処理手順を提案する．

一つの 2 入力ノードを N 個のプロセッサに分散させることを考える．処理をこれらに分散させ，かつ，重複したテストを行わないためには，どれかの一つのプロセッサのメモリノードだけにトークンを記憶すれば良い．また，プロセッサを通じて処理量ができるだけ均等となるためには，各プロセッサのメモリノードに記憶されたトークンの個数が均一となるようにすれば良い．このことから，次のようにして，2 入力ノードを並列化できる．

まず，2 入力ノードを N 個のプロセッサに割り付け，それぞれのプロセッサに 0〜$N-1$ までの番号をふる．またトークンにも一連番号をふることにする．トークンは既にタイムタグとして一連番号を持つので，これを流用すれば新たな番号を付与するまでもない．2 入力ノードが割り付けられたすべてのプロセッサにトークンが入力されるようにする．各プロセッサは，トークンが到来した際に，この 2 入力ノード及び直上のメモリノードの処理として以下のようにする．

- トークンの番号を N で除した余りがプロセッサの持つ番号と同一ならば，そのプロセッサはメモリノードへの記憶を行う．

- トークンに対するテストはすべてのプロセッサで行う．

本並列化手法による 2 入力ノードを動作を図 2.9 に示す．白丸が到来したトークンである．逐次処理の場合 (図の左端) は，到来したトークンが左側のメモリノードに記憶されるとともに，1 台のプロセッサで右側のメモリノードに記憶された 3 個のトークンのすべてとのテストが行われる．ここでは，メモリノードの一番下にあるトークンがテストに成功し，新たなトークンが生成されて下段に伝播させられる．並列化した場合には，図の右側に示したように，まず到来したトークンはこの例では中央のプロセッサのメモリノードに記憶される．次に各プロセッサは，自己が持つメモリノードの内容を対象としてテストを行う．右側のメモリノードには，過去の処理によって，3 個のトークンはそれぞれ異なったプロセッサに記憶されているため，それぞれ 1 個のトークンに対してテストが行われる，結果として，逐次処理の場合と同じく，3 個のトークンに対するテストすべてが行われることになる．ここでは，右端のプロセッサでテストが成功し，このプロセッサにおいて新たなトークンが生成され下段に伝播させられる．

図には示されていないが，更に右側入力よりトークンが到来した場合を考えてみよう．上と同様に，到来したトークンはまずどれか 1 個のプロセッサの右側メモリノードに記憶される．続いて，各プロセッサでそれぞれが持つ左側メモリノードの内容に基づいたテストが行われる．右端及び左端のプロセッサでは過去のいつかの時点で記憶されたトークンを対象とし，中央のプロセッサでは先ほど到来して記憶されたトークンを対象としてテストを行う．この場合も結果として逐次処理と同じ 3 個のトークンに対するテストが行われることになる．

このように，複数のプロセッサに 2 入力ノードを割り付け，メモリノードへの記憶をどれか 1 個のプロセッサが行うことにより，テストを繰り返す過程で処理対象となるトークンの記憶が分散され，条件照合処理を並列化することができる．また，このようにしても，逐次処理と全く同じくすべてのトークンに対してテストが行われ，洩れが生ずることはない．

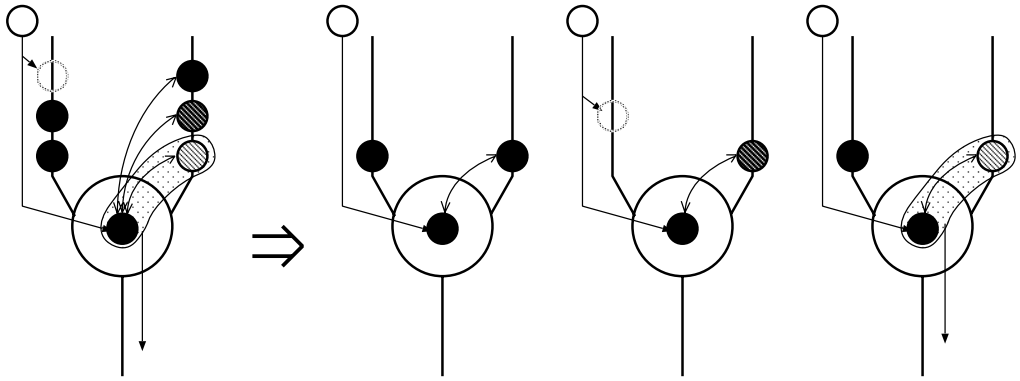


図 2.9: 2 入力ノード処理の並列化

2.5.3　2 入力ノードの連鎖の並列化

前節で述べたのは，単独の 2 入力ノードの並列化の方法であった．実際の RETE ネットワークは 2 入力ノードの連鎖構造を持っているため，更に，下段につながる 2 入力ノードでの処理も考慮する必要がある．特に問題となるのは，テストが成功した場合に，新たに生成したトークンを下段に伝播させることができるのは，そのテストを担当したプロセッサのみであるということである．上述の並列動作では，2 入力ノードが割り付けられたすべてのプロセッサに到来トークンが入力されることが前提となっている．下段の 2 入力ノードでも同様の動作をさせるためには，テストに成功した場合に生成されたトークンは全プロセッサに転送される必要がある．このような転送は，ジョインスライス並列化方式と同様に，通信オーバヘッドを大きくする原因となる．

このオーバヘッドを避けるために，2 入力ノード一つひとつを処理単位とするのではなく，その連鎖を処理単位とし以下のような動作とする．

- 2 入力ノードの連鎖を処理単位として複数のプロセッサに割り付ける
- 最上段の 2 入力ノード (トップノード) に対しては前述の並列処理を行う
- 2 段目以降の 2 入力ノードに対しては，逐次処理と同様にすべてのプロセッサがメモリノードへの記憶もトークンのテストも行う

このような処理方式で，2 段目以降の 2 入力ノードでも処理が並列化され，かつ，テストが洩れなく行われることを示そう．

まず，上段とその下の段の 2 入力ノードとの間にあるメモリノードがどのようにトークンが記憶されているか考える．上段では並列にテストが行われ，テストに成功したプロセッサではトークンが生成されて下段に伝播させられる．そのプロセッサにおいて伝播してきたトークンは，すべてそのプロセッサのメモリノードに記憶される．ところで，上段の 2 入力ノードにおけるテストの成功は必ずしも 1 個とは限らず，全く成功しない場合もあれば多数成功する場合もある．特に，パターン変数を持たない条件要素に対応する 2 入力ノードでは，テストが必ず成功するのでメモリノードに記憶されたトークンすべてに対し成功することになる．多くのトークンを処理する間には，これらのテストの成功はどのプロセッサにも均等に起こると考えられる．したがって，従属接続された 2 入力ノード間のメモリノードには，均等にトークンが記憶されると考えられる．また，言うまでもなく，すべてのプロセッサのメモリノード内容の和集合は，逐次処理でのメモリノードの内容と同一となる．このようにメモリノードに記憶されたトークンがプロセッサ間で分散しているため，2 段目以降の 2 入力ノードに対して弁別ネットからトークンが到来した場合にも，その 2 入力ノードは並列に処理されることになる．

一方，上段でテストに成功して生成されたトークンそのものの処理に関しては，それが生成されたプロセッサでしか処理できない．しかし，上述のように，多数のテストが成功するような場合は，複数のプロセッサがそれぞれに自プロセッサで生成したトークンを処理するため，並列化されることになる．また，対向する入力上のメモリノードには到来したすべてのトークンを記憶しているため，照合対象となるトークンの洩れはない．

弁別ネットからのトークンは，2 入力ノードの連鎖が割り付けられたすべてのプロセッサに転送される必要があるが，これは粗粒度方式と同様である．連鎖の中で発生したトークンは，そのプロセッサ内で処理されるため，通信の必要はない．このように本方式は，粗粒度方式と同程度の通信頻度で，2 入力ノードでの処理の多くを並列化できるものである．

2.5.4 TWIN 方式

上記で，2 入力ノードの連鎖のみについて，少ない通信頻度で並列化する方法を構築した．これに基づいた RETE マッチ・アルゴリズム並列化方式である TWIN 方式を提案する．

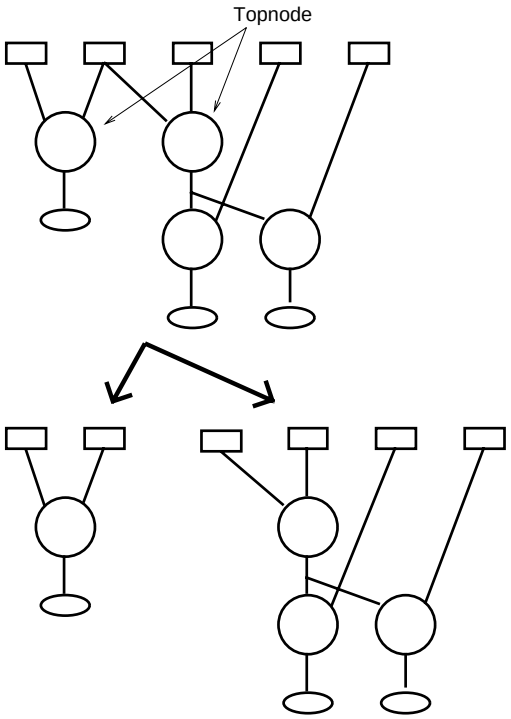


図 2.10: TWIN 方式の構造並列化

TWIN 方式 [100] は，本質的に通信不要な構造的粗粒度並列化をベースとして，上述の通信を増加させないデータ並列性を組み合わせ，良好な負荷バランスを実現する方式である．TWIN 方式の構造並列化は，トップノード並列化方式と同一である．図 2.10 に示すように，第 1 段目の 2 入力ノード (トップノード) とそれに続く 2 入力ノード，及びそれらの 2 入力ノードに接続されたすべての定数テストノードからなる部分的なネットを一つの並列処理の単位とする．この並列処理単位として分割されたネットを RETE サブネットと呼ぶことにする．2 入力ノードはトップノードから従属に接続され，かつ右側の枝は必ず定数テストノードに接続されているため，以上のような分割により切断される枝はルートノードに接続する枝のみとなる．ルートノードからの枝に流れるのは同一のトークンであるため，トークン入力時に一斉にトークンを配布すれば，その後の照合処理では通信は一切不要である．なお，この構造並列化では，定数テストノードが複数の RETE サブネットに複写される場合があり，これにより総処理量が増加する．しかし，定数テストノードでの処理量は 2 入力ノードでのそれに比較して非常に小さいので，大きな問題とはならない．

この RETE サブネットは，2 入力ノードの連鎖を完全に含んでいるので，前節

で提案した並列化を施すことによって，2 入力ノード処理も並列化される．2 入力ノード並列化の並列度は，原理的にはメモリノードに蓄積されたトークン数まで可能であるが，実際には蓄積トークン数に応じて RETE サブネットを動的に複写するのは困難なので推論開始前に固定的に与える．

このように TWIN 方式では，粗粒度の構造並列化に，通信頻度を増加させないように 2 入力ノード処理の並列化を組合わせることにより，粗粒度方式と同一の通信頻度で，より大きな並列度を得ることを可能としている．したがって，バス結合型などの比較的安価に構成可能なアーキテクチャで，広い範囲の ES 特性に対して十分な速度向上が得られることが期待できる．

2.6 TWIN 方式の解析的評価

2.4 節で提案した速度向上度の推定式により TWIN 方式を評価する．ES パラメータは従来方式の評価に用いたと同一のものを用い，プロセッサ数は 16，2 入力ノード処理の並列度は 3 とした．結果を図 2.11 に示す．

結果のグラフからわかるように，2 入力ノード並列化の効果により，常にトップノード並列化方式より大きな速度向上度を得ている．また，トップノード並列化方式の速度低下が顕著なルール数が少ない領域で，特により大きな性能改善が得られている．特に，WME 数が十分にあれば，小ルール数でも満足な速度向上が得られることがわかる．

2.7 実験による評価

TWIN 方式の性能評価をより確かとするため，小規模な実験システムを用いた評価を行い，解析結果との比較を行う．両評価の結果が一致すれば，実験ではカバーできない領域でも解析による結果がほぼ正しいと考えることができる．

実験システムはマイクロプロセッサボードを汎用バスで結合したものである (図 2.12)．バスはトークンを全 PE に配布するための放送機能を持っている．また，プロセッサ数は最大 10 である．

実験用として 2 地点間の最短経路を深さ優先法により探索する小規模 ES を構築して用いた．本 ES はルール数 23，WME 数 1000 で構成される，実験によって得られた条件照合フェーズの速度向上を図 2.13 に示す．実線が実測値，破線が推定

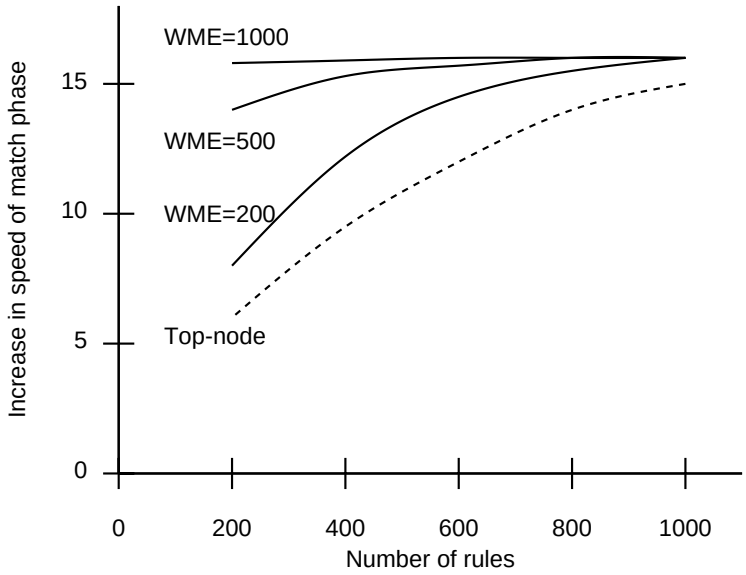


図 2.11: TWIN 方式の解析評価結果

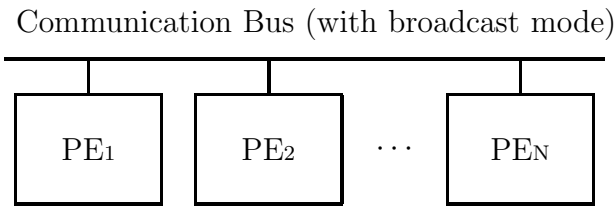


図 2.12: 実験システムの構成

値である．解析評価の場合と同様にデータ並列度は 3 とした．また，トップノード並列化方式の結果もあわせて示している．結果のグラフからわかるように，トップノード並列化方式では 2 倍程度で速度向上が飽和している．これは，ES 規模が 20 ルール程度であり，RETE ネットワークの構造的分割だけでは負荷ばらつきが大きいことを示している．一方，TWIN 方式では，2 入力ノード処理並列化の効果により 10PE でも速度向上が飽和せずに上昇している．しかし，ルール数が極端に小さいため，速度向上比率（速度向上／ PE 数）は低めである．100 ルール規模の ES を用いれば，より高い速度向上比率が得られ，16PE で 10 倍程度の速度向上が得られると予測される．

また，図からわかるように，推定結果は実測とは良く合っており，誤差は 10%

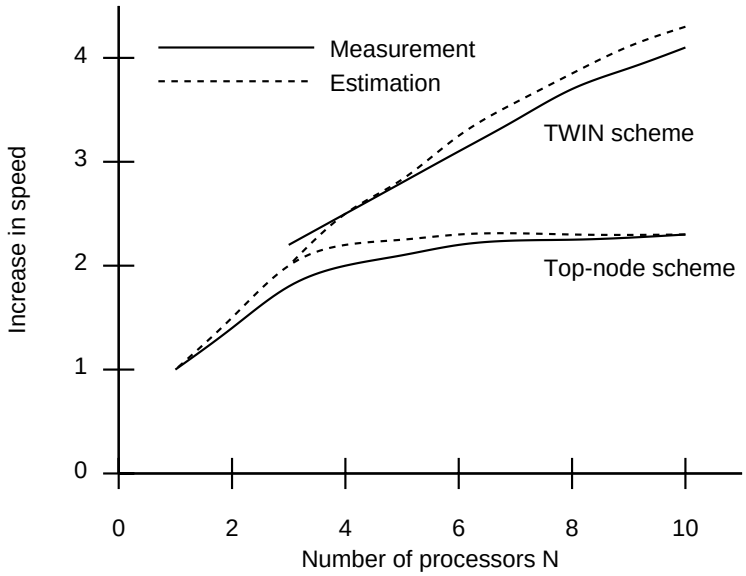


図 2.13: 実測結果

程度である．推定式は，並列化方式の適用領域の比較や，その方式に適したアーキテクチャの探索に利用するには十分な精度を持っていることが示された．

2.8 まとめ

本章では，PS の条件照合フェーズの並列化について論じた．

まず，並列化方式の速度向上を解析的に評価する手法を提案し，これに基づき，従来から提案されている代表的な並列化方式を評価した．解析的評価手法では，2 入力ノードのふるまいに着目して RETE ネットワークをモデル化し，ネットワークの構造や処理量を ES パラメータにより表現する．これに基づき，負荷ばらつき，通信オーバヘッドなどを考慮して，速度向上推定の一般式を導出し，評価用 ES やシミュレータを用いる事なく様々な並列化方式の評価を可能とした．

本解析評価手法を用い，代表的な並列化方式について，ES パラメータを広範囲に変化させて比較評価を行った，その結果，従来の方式では，小ルール数の領域で負荷ばらつきにより十分な速度向上が得られないか，または，非常に高速なトークン転送を行うための特殊なハードウェアが必要とされることを明らかにした．

次に，通信オーバヘッドと負荷ばらつきが共に少ない TWIN 方式を提案した．

本方式は，RETE ネットワークを構造的に粗く分割することによりノード間での通信を不要とし，2 入力ノードでの条件間テスト処理を並列化することにより小ルール時の負荷ばらつきを抑えている．解析手法と実際の ES を用いた実験により評価を行い，小ルール数の領域でも十分な速度向上が得られることを確認した．

第3章　プロダクションシステム向け並列コンピュータPresto

PS条件照合フェーズの並列化方式として2章においてTWIN方式を提案した．本章では，この方式に基づき推論インタプリタを高速実行するべく開発した並列コンピュータ “Presto” について述べる．

PSの推論処理のうち90%以上を条件照合フェーズが占めるため，このフェーズの並列化が最も重要なことに間違いはない．しかし，条件照合フェーズが十分に高速化されると，他のフェーズでの処理が速度を支配する要因となって来る．したがって，推論インタプリタの動作を高速化するためには，条件照合フェーズのみならず，他のフェーズも可能な限り並列処理する必要がある．また，現実の並列コンピュータではプロセッサは有限であり，その個数は実行開始前に固定的に決っている．一般に，並列処理のために分割した処理単位はプロセッサ数よりも多いため，複数の処理単位を一つのプロセッサに割り付けることになる．プロセッサ間の負荷ばらつきはこの割り付けの方法に依存し，これが性能に影響を与える．特に，本論文が対象としている疎結合型アーキテクチャにおいては，実行中に処理単位をプロセッサ間で異動することは，通信オーバヘッドによる性能の著しい低下を引き起こすため，事実上不可能である．したがって，実行以前に負荷ばらつきを推定し，これが小さくなるように処理単位をプロセッサへ割り付ける手法が必要となる．

本章では，まず，条件照合フェーズ以外の並列化及び処理単位の割り付けを含む，認知–実行サイクルの総合的な並列化方式を提案する．つぎに，この並列化方式に好適なアーキテクチャについて論じ，そのアーキテクチャに基づいて実際に開発した並列コンピュータのハードウェアについて説明する．更に，推論インタプリタをこのハードウェアに実装する際の課題を明らかにし，その解決法を提案する．最後に，開発したシステムを実際のESを使って評価する．

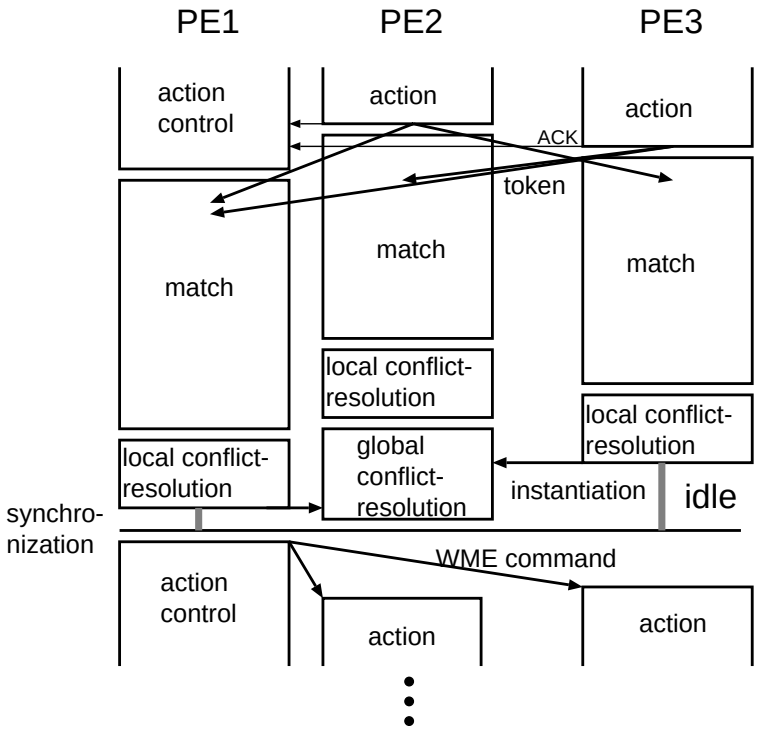


図 3.1: 並列化認知サイクル

3.1 認知–実行サイクルの並列化方式

TWIN 方式の通信不要性を活かし，全体として高速な並列推論インタプリタを実現するためには，競合解消フェーズ・実行フェーズでも通信を抑え，かつ，バランス良く並列化する必要がある．ここでは，条件照合フェーズにおける TWIN 方式と同様に通信頻度が小さいことを特徴とする，認知–実行サイクル全体の並列化方式を提案する．また，本方式は，通品頻度のみならず，認知–実行サイクルのフェーズ移行の際に必要となる同期を少なく抑えることも特徴としている．

本方式における認知–実行サイクル全体の動作を図 3.1 に示す．図に示すように，各プロセッサが非同期に条件照合・競合解消を行い，全体の競合解消 (グローバル競合解消) 終了時に一回だけ同期がとられる．その後のルール実行も (アクション要素は複数あれば) 並列に処理され，実行を終了したプロセッサから順次照合フェーズへ移行する．このように，認知–実行サイクル中に同期は 1 回のみであり，また，プロセッサ間の通信も図に示すように，トークン発生時，グローバル競合解消時，そしてルール実行時に必要となるのみである．

以下では，順を追ってより詳細に本並列化方式の動作を述べる．

実行フェーズで発生したトークンは，放送によってすべてのプロセッサに入力される．条件照合フェーズはTWIN方式により並列に処理される．TWIN方式は，RETEネットワークをRETEサブネットに分割してプロセッサに割り付けているため，各プロセッサの競合集合には，割り付けられたRETEサブネットに対応するインスタンシエーションが蓄積される．

条件照合フェーズが終了すると競合解消フェーズに移行するが，本方式では，そのプロセッサが持つ競合集合内でのローカルな競合解消と，全体の競合を解決するグローバル競合解消とに二段階化している．競合解消処理とは，競合解消戦略に基づきインスタンシエーションの優先度を相互に比較し，最も優先度の高いものを選択する処理である．本方式では，まず各プロセッサがローカル競合解消により自プロセッサ内で最も優先度の高いインスタンシエーションを選択し，これをグローバル競合解消プロセッサに転送する．すべてのプロセッサからインスタンシエーションを受信したグローバル競合解消プロセッサは，全体で最も優先度の高いインスタンシエーションを選択する．プロセッサ内でのローカル競合解消には通信が不要であり，グローバル競合解消でも各プロセッサが一つのインスタンシエーションを転送するだけで良い．このため，各プロセッサが持つすべてのインスタンシエーションを一カ所に転送して競合解消を行う方式に比べ通信オーバヘッドを大幅に削減できる．グローバル競合解消は，最も早くローカル競合解消を終了したプロセッサ (図ではPE2) が行う．グローバル競合解消プロセッサの決定法については後述する．

グローバル競合解消プロセッサでは，インスタンシエーションが他のプロセッサから到着する毎に優先度を比較して，最優先のインスタンシエーションを選択しておく．この動作により，最も遅いプロセッサがインスタンシエーションを転送した後，一回の優先度比較でグローバル競合解消が終了する．したがって，全プロセッサの条件照合の終了を待って集中的に競合解消を行う方式に比べて，プロセッサのアイドル時間も削減される．

グローバル競合解消が終了すると，どのインスタンシエーションが選択されたかを全プロセッサに放送した後，同期を取り，実行フェーズに移行する．実行フェーズでも，アクション部に複数のWME生成／更新／削除操作がある場合には，これを並列に処理する．選択されたインスタンシエーションを持つプロセッサが実行制御プロセッサ (図ではPE1) となり，WME生成／更新／削除命令を他のプロセッサに配布する．命令を受信したプロセッサはWME操作を行い，処理が終了し

た時に終了通知を実行制御プロセッサに返送する．ただし，本方式での実行フェーズ並列化は，1 回の認知–実行サイクルにつき 1 つのルールが実行される従来のプロダクションシステムの推論動作に従ったものであり，1 回の認知–実行サイクルの中で複数のルールを並列実行するもの [40] ではない．

以上のように，本方式では条件照合フェーズのみでなく競合解消及び実行の各フェーズも並列処理し，また，各フェーズともローカリティの高い処理を抽出して並列化している．このため，通信が必要となるのは以下の場合だけである．

- 実行により発生したトークンの転送
- グローバル競合解消のためのインスタンシエーションの転送
- WME 生成／更新／削除命令と終了通知の転送

1 回の認知-実行サイクルでの通信頻度は数回〜10 数回程度である．

3.2 ルール・WME の割り付け法

ここでは，上述の通信が少ないという利点を保ったまま，負荷ばらつきを抑え，各プロセッサが持つローカルメモリが均等に消費されるような，ルール・WME の割り付け法を提案する．

3.2.1 RETE サブネットの割り付け

冒頭でも述べたように，疎結合型並列コンピュータでは処理単位の実行中の異動は困難であるから，本方式ではルール条件部にあたる RETE サブネットを静的に割り付ける．

静的割り付けで良好な負荷バランスを達成するために，実行前に RETE サブネットの処理量を予測して，プロセッサでの処理量が均等となる割り付けを行う．正確な処理量は実際に ES を動作させるまで知ることはできないが，処理量が活性化されるノード数に比例すると考えれば，RETE ネットワークの構造からおおまかに予測することはできる．ルートノードに入力されたトークンは，クラスに応じてそのクラスに関係する定数テストノードへと流れ，定数テストに成功すればその下に接続された 2 入力ノードへ流れる．その 2 入力ノードでの照合が成功すれば更に下段の 2 入力ノードへと流れてゆく．このことから，定数テストノードで

の成功確率や 2 入力ノードでの成功確率を一定と近似すれば，活性化されるノード数は，活性化される定数テストノード数に比例すると考えられる．したがって，各プロセッサで活性化される定数テストノードの数を，どのクラスのトークンに対してもできるだけ均等になるように RETE サブネットを割り付けることで，負荷の均等化を図る．以下に，割り付けアルゴリズムを示す．

割り付けアルゴリズム

クラス $C_1, C_2, \ldots, C_n$ を持つ RETE ネットワークを m 台のプロセッサ PE_1,PE_2,...,PE_m に割り付けることを考える．N_{ij} を，PE_j に割り付けられ，かつ，クラス C_i に関連する定数テストノードの数とすれば，クラス C_i に関連する定数テストノード数の分散 V_i は，

$$V_i = \frac{1}{m}\sum_{j=1}^{m}(N_{ij} - \frac{1}{m}\sum_{k=1}^{m}N_{ik})^2,$$

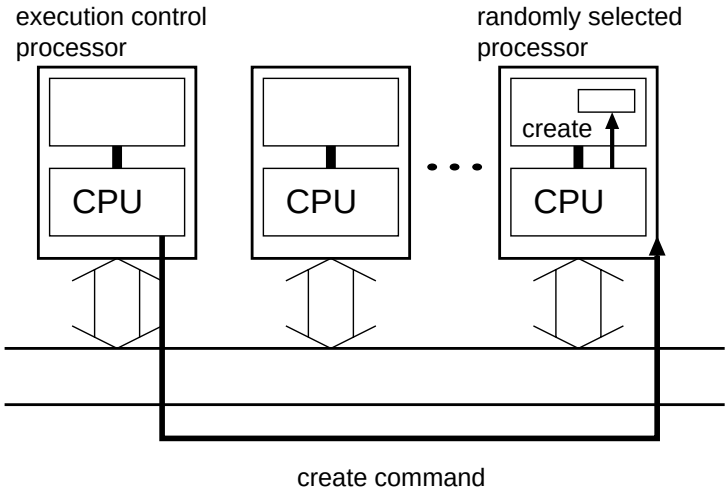
で表される．この分散がどのクラスに対しても小さくなるように，クラス毎の分散の総和，すなわち $V = \sum V_i$ を最小化する．そのために，以下の手順で RETE サブネットをプロセッサに割り付ける．

1. 分割された RETE サブネットを定数テストノード数の大きい順にソートする．
2. ソートされた RETE サブネットを一つ取り出す．
3. x を 1 から m まで変化させて，PE_x にこのネットが割り付けられた場合の分散の和 $V = \sum V_i$ を計算する．
4. 上で計算した V が最も小さいプロセッサにネットを割り付ける．
5. すべての RETE サブネットを割り付けるまで，ステップ 2 からステップ 4 を繰り返す．

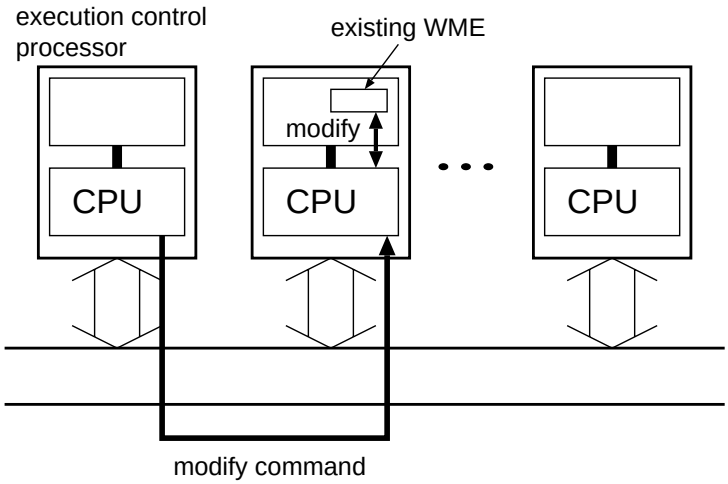
3.2.2 WME の割り付け

WME 操作を並列に行い，かつ，各プロセッサでのメモリ消費を均等化するために WM も各プロセッサのメモリ上に均等に分散して配置する．

WME は WME 生成命令によって生成されるため，この生成命令の宛先プロセッサをランダムにすることで WME の分散を実現する．このように WME を分散した場合には，通信を抑えるために，WME 更新・削除動作は WME を保持するプロセッサが行う必要がある．



(a) WME の生成動作



(b) WME の更新動作

図 3.2: WME 操作

以下に WME 操作の詳細を述べる．図 3.2(a) は WME 生成動作を示している．アクション部に WME 生成記述がある場合には，実行制御プロセッサは他のいずれかのプロセッサに WME を生成命令を発行する．この WME 生成命令の宛先となるプロセッサをランダムに選択することにより WME は均等に分散される．一方，WME 更新／削除処理は，対象となる WME を持つプロセッサ上で行う．このために，インスタンシエーションにその WME を持つプロセッサの番号をタグとして持たせている．図 3.2(b) に示すように，実行制御プロセッサがそのタグを参照して更新／削除命令を発し，WME を保持するプロセッサ上で実際の更新を行う．

3.2.3 アクション部の割り付け

上記の WM 割り付けでは，どのプロセッサでもあらゆるルールに記述された WME 操作を行う可能性があるため，ルールのアクション部はすべてのプロセッサに重複して割り付ける．実際には，アクション部は手続き型プログラムとして表現されているので，このプログラムをすべてのプロセッサに配置する．アクション部のプログラムは RETE ネットワークや WME に比較してメモリ消費が十分小さいため，重複配置してもメモリ消費の総量はそれほど増加しない．2〜3 の ES についての測定では，これによるメモリ消費増加は 10%程度である．

3.3 PS 向け並列コンピュータ Presto

上記方式に基づいて推論インタプリタを並列処理すべく，並列コンピュータ “Presto” を開発した．Presto は，ローカルメモリと CPU とを持ったプロセッシングエレメント (PE) をバスにより結合した疎結合型並列コンピュータである．PE 間の通信や同期はバスを介して行うが，ここに本並列化方式を効率的に処理するための機能を持たせている．また，入出力はバスに接続されたホストプロセッサのみが行う．

ここでは Presto のハードウェア構成と，特徴的なバス機能について詳述する．

3.3.1 全体構成

Presto の構成を図 3.3 に示す．最大 10 台のプロセッサが BCU(Bus Control Unit) を介してバス結合されている．Presto は，NEC の PC9801 パーソナルコンピュータをホストプロセッサとし，そのバックエンドとして動作するよう設計されている．ホストプロセッサは，それが持つ拡張バス (C バス) からインタフェース回路を介して Presto のバスに接続される．このため，若干の回路的な差異があるものの，ホストプロセッサもバスで結合された一つの PE とみなすことができる．図 3.4 に，Presto をホストプロセッサに接続した様子を示す．

Presto そのものは，ディスクドライブ，ディスプレィ，キーボード，プリンタなどの入出力デバイスを持たない．このため，PE からの入出力は，すべてホストプロセッサを介して行うことになる．PE 上には，MS-DOS システムコールをインターセプトして，システムコール要求をホストプロセッサに転送する制御ソフ

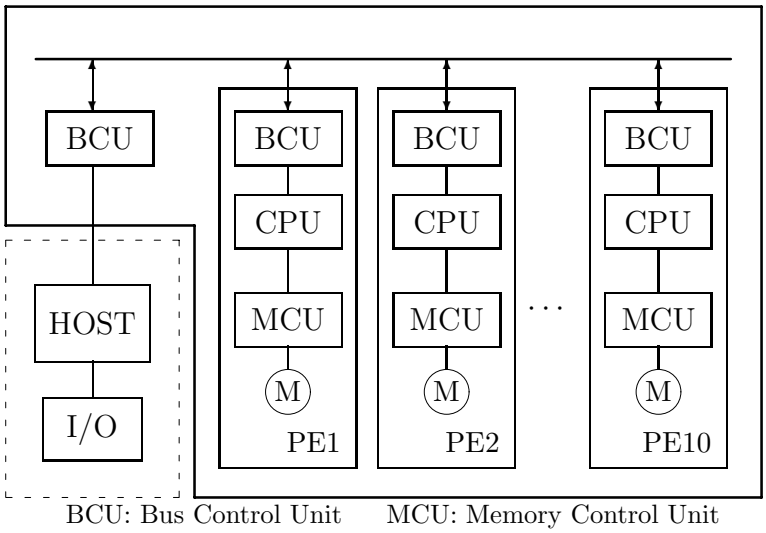


図 3.3: Presto のハードウェア構成

図 3.4: ホストプロセッサに接続された Presto

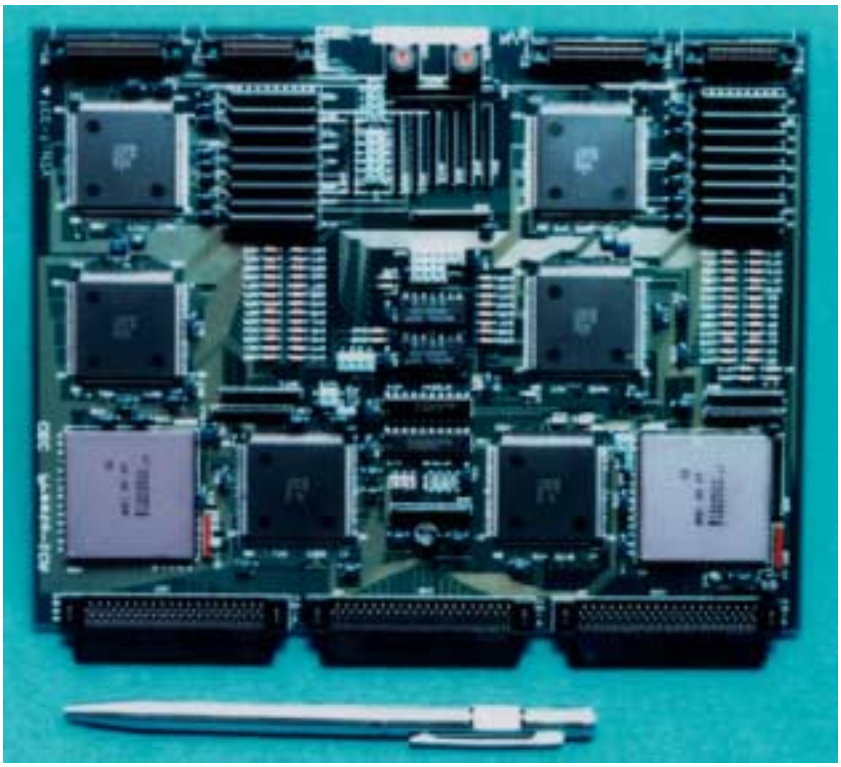


図 3.5: Presto の PE 基板 (1 枚あたり 2PE)

トウェアが搭載される．これにより，PE 上でも，MS-DOS 搭載の通常のパーソナルコンピュータと同じように入出力要求を処理することができる．PE 上のアプリケーションソフトウェアからは，その PE 上で入出力処理がなされているように見えるが，実際には入出力要求はホストプロセッサで処理される．このような機構により，すべての PE は，ホストプロセッサの制御下にあるあらゆる周辺デバイスを利用できることになる．

本論文で提案している PS 並列化方式は，プロセッサ間での処理の独立性が高く，通信オーバヘッドを抑えるよう工夫が加えられているため，このような低廉でコンパクトな並列コンピュータ上で効率的に実行することが可能である．

3.3.2 プロセッシングエレメント (PE)

個々の PE は，CPU とローカルメモリ，メモリ制御ユニット (MCU)，バス制御ユニット (BCU) から構成されている．ローカルメモリはバスを介して他の PE からもアクセスされるため，MCU はローカルメモリに対する CPU からのアクセス

とバスからのアクセスとの調停も行う．

Prestoで用いられているPEの基板を図3.5に示す．1枚のプリント基板上に二つのPEを実装している．

CPU:

CPUはインテル80386を用いた． ホストプロセッサであるPC9801も80x86を搭載しており，これと同一アーキテクチャのCPUを用いることで，開発環境を統一できる上，更に，ホストOSとPE上に搭載するモニタプログラムと間の親和性を高めることが容易にできる．更に，Prestoのアプリケーションとして並列化の対象となるES開発ツール“KBMS”[31]推論インタプリタ(KBMS/C)も，インテルアーキテクチャ上で動作するよう開発されている．このため，アーキテクチャの違いによる移植作業が不要となり，並列化に専念できる．

メモリ:

各PEは最大で4MBのDRAMを搭載する．Prestoは各PEがローカルメモリのみを持ち，共有メモリは持たない．メモリのリフレッシュやタイミング制御は，下に述べるMCUによって行われる．PE間でのデータ転送は，あるPEのメモリにあるブロックデータを，MCUがBCUと連携して他のPEのメモリへ書き込むことにより行われる．すなわち，Prestoでのプロセッサ間データ転送はDMA(Direct Memory Access)によるものであり，CPUの介在を必要としない．このために，メモリは，CPUだけでなくバス側から直接アクセスされる．なお，データ転送の詳細については，バス機能の項で述べる．

BCU:

図3.6にBCUのブロック図を示す．BCUは，PE間バス制御部，CPUバス制御部，DMA制御部，同期及び割込み制御部，バスアービタから構成されている．BCUは10,000ゲートのASIC(Application Specific Integrated Circuit)チップ上で実現されている．

MCU:

メモリの項で述べたように，MCUは，CPU及びバス(BCU)からのメモリアクセスやリフレッシュを制御する．図3.7にブロック図を示す．MCUは，

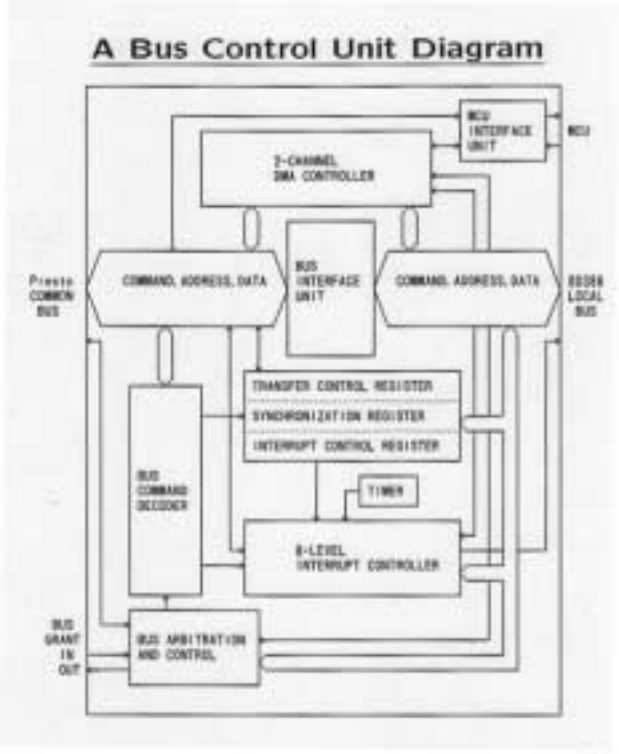


図 3.6: BCU のブロックダイヤグラム

CPU バス制御部，メモリアクセス制御部，メモリリフレッシュ制御部，データパス多重化部から構成されている．MCU は，5,000 ゲートの ASIC チップ上で実現されている．

3.3.3 バスの機能

ここでは，Presto の特徴であるバスを用いたプロセッサ間通信機能を説明する．プロセッサ間通信とは，データ転送だけでなく，同期，割込み制御，バスアービトレーションも含む．

データ転送

前述したように Presto におけるプロセッサ間のデータ転送は，ダイレクトメモリアクセス (DMA) によって行われる．DMA は，メモリと入出力サブシステム間のデータ転送を高速に行うために良く用いられる技術である．TWIN 方式では，プロセッサ間で転送されるデータは，連続したメモリブロックに配置されることが

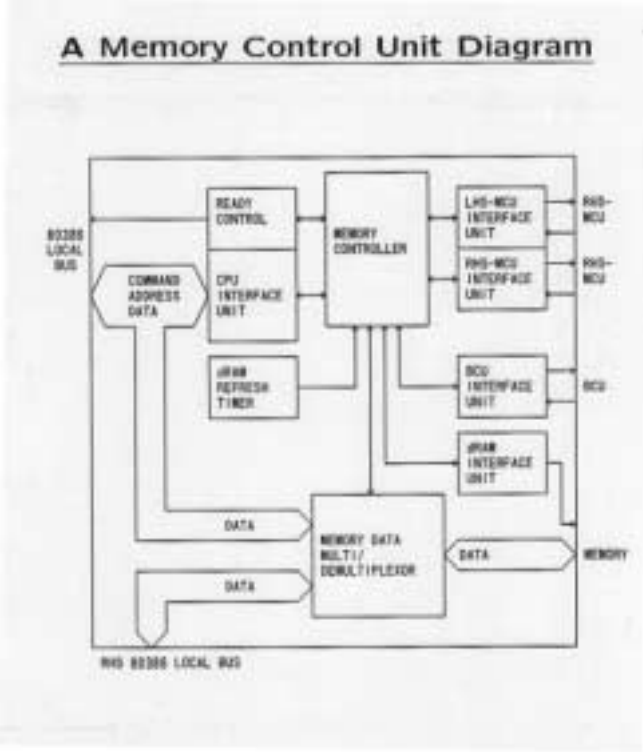


図 3.7: MCU のブロックダイヤグラム

ほとんどである．このため，プロセッサ間のデータ転送においても，メモリブロックを CPU の介在なく転送できる DMA は効率的である．

本並列化方式においては，データ転送の要求が生じて DMA 転送が起動されるのは次のような場合である．

- 最初にローカル競合解消を完了したプロセッサがグローバル競合解消をするために，他のプロセッサからそれぞれのプロセッサでのローカル競合解消結果のインスタンシエーションを受け取る場合
- 実行フェーズにおいて，実行制御プロセッサが WME の生成/変更/削除のコマンドを発行する場合
- WME 操作により生じたトークンを全プロセッサに転送する場合

この DMA 機能は PE と PE の間の 1 対 1 データ転送だけではなく，ある PE からバス上のすべての PE に対してデータを一斉転送する放送モードも持つ．Presto において放送モードでのデータ転送が起るのは，上記の第 3 項，すなわちトークンを全 PE に転送する場合である．放送モードは条件照合のためにトークンを全 PE に繰返し転送するのに非常に効率的な方法である．

DMA では，CPU の介在なしにデータが転送されるため，例えば条件照合フェーズにおいてはプロセッサが一つのトークンの処理を完了する前に別のトークンが送られて来る場合がある．同様に，実行フェーズでも，プロセッサが WME の操作指示を処理し終わる前に次の WME 操作指示が送られる場合もある．このような場合のデータ取りこぼしを防ぐために，二つのリングバッファが個々の PE に備わっており，それぞれ，トークンキューとアクションキューとして利用される．

また，デッドロックを防ぐために，各 PE にはシステム中の PE 数と同じ個数のデータ転送バッファが確保されており，更に PE 個数分のフラグレジスタが備えられている．フラグレジスタは，それぞれ対応する PE からセット/クリアができるようになっており，個々のデータ転送のフロー制御に使うことができる．PE は，自 PE が持つフラグレジスタのうち，送信先 PE に対応するフラグを参照し，これがクリアになっている場合にのみデータの送信を行う．例えば，PE_x が PE_y にデータを転送する場合，PE_x は PE_y の x 番目のバッファにデータを送り込むと同時に，自 PE 内の y 番目のフラグを立てる．データを受け取った PE_y は，データ転送バッファ上のデータをその種別に応じて，トークンキューまたはアクションキーに移す．これをフラッシングと呼び，受信側の PE_y の都合により任意のタイミングで実行される．フラッシングが済むと，PE_y は送信側 PE_x の y 番目のフラグを下す．

このように，データ転送個々のフロー制御をフラグレジスタを用いて行い，更にトークンや WME 操作指示に対してはキューを用意することで，デッドロックとデータの取りこぼしを回避している．

同期

Presto では，PE 間の同期のために 2 種類の機構が備えられている．一つは，専用の同期レジスタを用いた機構であり，もう一つは単純なワイヤード・アンド線による機構である．専用レジスタは柔軟な PE 間同期を提供する．これは，レジスタ上の任意のパターンを同期条件として設定できるためである．一方，ワイヤード・アンド線による同期は高速であるが，単純な条件 (全 PE がこのワイヤード・アンド線をアサートした場合) でしか同期をとることができない．

個々の PE は同期レジスタを持っており，それぞれのレジスタの x 番目のビットが PE_x が同期待ち状態に入っていることを示すようになっている．この同期レジ

スタの x 番目のビットは，PE_x がバスの放送機能を使って全 PE に対し一斉にセットすることができる．同期レジスタが指定したパターンとなった場合に，CPU に割り込みがかかるようになっている．

同期レジスタによる同期動作をより具体的に説明する．PE_x，PE_y，PE_z の間で同期をとる場合を考える．これらすべての PE において，割り込み発生パターンとして x 番目，y 番目，z 番目のビットがセットされた場合を指定しておく．PE_x が自 PE が担当する処理を完了し，他の PE の完了を待つ同期待ちに入るとする．この際に，PE_x は同期レジスタの x 番目のビットをセットするよう放送する．これより，PE_x，PE_y，PE_z のすべての PE でそれぞれが持つ同期レジスタの x 番目のビットがセットされる．同様に，PE_y が担当する処理を完了し，続いて PE_z の処理が完了したとする．PE_z からの放送により，すべての PE で同期レジスタの x 番目，y 番目，z 番目のビットがセットされる．これによって，すべての PE で同時に CPU への割り込みが発生する．この割り込みによって同期待ち状態から次のフェーズへ移行するようにプログラムしておけば，これにより全 PE が同期して次フェーズへ移行できる．

認知–実行サイクルにおいては，ローカル競合解消の後に同期が必要となる．このために本同期機構を利用している．各プロセッサはローカル競合解消を完了すると同期をとるための関数の呼び出して同期待ち動作に入る．すべてのプロセッサがローカル競合解消を完了すると，各 PE で CPU への割込みが発生し，PE は次のフェーズへ移る．

割込み制御

PE は任意の PE に対して割込みをかけることができる．PE をまたがる割込みでは，PE 間でのメッセージ転送によって，ベクタ割込みと同様な動作を可能としている．PE は割込み要求ステータスレジスタを持っている．このレジスタは，割込みをかける先の PE がその割込み要求を受付けたがどうかを示すのに使われる．また，PE は割込み要求フラグを持ち，他の PE が割込み処理中であるかどうかを示すのに使っている．

PE_x が PE_y に対して割込みをかけることを考える．まず，PE_x は割込み要求ステータスレジスタの y 番目のビットを調べる．もしこれがクリアされていれば，この y 番目のビットを立てた後に割込要求を PE_y に対して送る．もしも割込要求ス

テータスレジスタのビットがセットされていれば，PE_y は割込み処理中であることを示しているので，多重割込みによるデッドロックを防ぐために PE_x から PE_y への割込みは禁止される．

PE_y では，PE_x からの割込要求を受付けると，それを示すために割込要求ステータスレジスタの x 番目のビットをセットする．割込処理が完了すると，PE_y は自身の x 番目のフラグと PE_x 上にある割込要求ステータスレジスタの y 番目のビットをクリアする．結局，PE_y はそれぞれの PE から一つの割込要求を受入れることができる．多重割込みによるシステムのデッドロックを防ぐため，個々の PE は，他の PE からの割込要求を一つずつしか受入れないようになっている．そして，PE 上ではこれら割込み要求をキューイングするようになっている．

割込要求は，あらゆるプロセッサ間通信において発生する．この割込要求は次の二つに分類できる．一つは，認知–実行サイクル中のデータ転送により発生する割込要求である．例えば，条件照合フェーズにおけるトークンの転送，ローカル競合解消の結果の転送，実行フェーズでの完了マークの転送などの際に発生する割込要求を含む．もう一方はより緊急性の高い割込要求である．この種の割込は，ローカル競合解消を最初に完了したプロセッサが，自分がグローバル競合解消を制御することを宣言する場合などに発生する．

バスアービトレーション

バスアービトレーション機能は，システム中で最も高位にある (PE 番号の若い)PE が担当する．VME バスにおけるアービトレーション [67] と同様に，バスを要求する PE はディジーチェイン接続されたグラント線によってバス利用の許可を受取る．一般的にはディジーチェイン接続のグラント線を用いた方式はレスポンスタイムを増加させる．しかしながら，低廉で効率的であり，Presto のようなバス接続のマルチプロセッサシステムに対しては好適な方式である．アービトレーションサイクルを高速化するために，現在のバスマスタがバスサイクルを終了しなくとも，他のバスマスタ候補がバス要求を出すことが可能となっている．

3.4 推論インタプリタの実装

以下では，推論インタプリタの並列化をこのような並列コンピュータ上に実現する場合の課題とその解決について述べる．

3.4.1 フェーズ移行処理

前述のように，本並列化方式では，条件照合フェーズの処理を非同期に行っている．このような非同期動作では，各プロセッサが，認知–実行サイクルで発生するすべてのトークンを処理したことをいかに判断するかが問題となる．1 個のトークンの処理が完了した後も，いずれかのプロセッサでまだ WME 操作が行われていて，新たなトークンが発生する可能性があるためである．すべてのトークンが発生してしまい，それらに対する条件照合処理が完了したことを確認するまで，条件照合フェーズを終了することができない．

実行フェーズから条件照合フェーズへの移行に同期を挿入して，全プロセッサが実行終了を待てば確認は不要となるが，このような方法では，実行を早く終了したプロセッサのアイドル時間を増大させてしまう．本実現法では，常にどれか一台のプロセッサのみが実行フェーズを制御している点に着目し，実行終了を示す特殊なトークンを放送することにより，同期が不要な実行フェーズ終了確認を実現する．

各プロセッサはトークン用のキューを持ち，自プロセッサでの実行フェーズ終了直後からこのキューのトークンを取り出しながら条件照合処理を行う．キューが空の場合にも，いずれかのプロセッサでまだ WME 操作が行われており処理すべきトークンが到着する可能性があるため，キューの監視を続ける．実行制御プロセッサは，WME 操作を終了したプロセッサが発行する終了通知により実行フェーズ終了を確認すると，全プロセッサのトークンバッファに対して実行終了トークンを転送する．各プロセッサは，バッファから取り出したトークンが実行終了トークンであった場合にのみ，その認知–実行サイクルでの条件照合をすべて終了したと判断し，ローカル競合解消に移行する．

3.4.2 同期処理

競合解消フェーズから実行フェーズへの移行は，すべてのプロセッサが一斉に行う必要があるため，同期をとることが必要である．同期には，共有メモリに対してすべての PE がプロセッサ状態を書き込み，各 PE がこれを監視するポーリング方式が良く使われる．しかし，この方式では，監視のためのメモリアクセスが多数の PE から発生するためにバス競合が発生し，通信オーバヘッドの増大を招く．特に Presto は共有メモリを持たないためこのような同期法は採ることができない．

Prestoでは，前述したようなバスの放送と特別なハードウェア (同期レジスタ) を用いて同期を実現する．同期レジスタは，PE台数分のビット長を持ち，バスからビット毎のセット／リセットが可能である．また，このレジスタが特定のパターンになった時にCPU に割り込みがかかるようにしている．各プロセッサは，競合解消が終了すると，バスの放送モードを利用して，全PE の同期レジスタに対し，自PE番号に対応するビットをセットする．そして，割り込みパターンを全ビットがセットされた状態に設定して，割り込み待ちに入る．全プロセッサで競合解消が終了した瞬間にすべての同期レジスタの内容は割り込み起動パターンと一致し，すべてのPEで一斉に割り込みがかかる．この割り込みによって，全プロセッサが同時に実行フェーズに移行する．このように，バスの放送機能と論理演算可能なレジスタを利用して各PEが独立かつ同時に同期検出を行うことにより，バスのトラフィックが少なくプロセッサにも負荷とならない同期が実現できる．

3.4.3 グローバル競合解消プロセッサの決定法

ローカル競合解消を最も早く完了したプロセッサが，グローバル競合解消プロセッサとなる．そのためには，全プロセッサが，他のプロセッサのローカル競合解消終了を同時に検出し，最も早く終了したプロセッサを識別する必要がある．本方式では，同期で用いたと同じ機能をもったレジスタを利用して，これを実現している．ローカル競合解消を終了したプロセッサは，まずこのレジスタをチェックする．どのビットもセットされていなければ，バスの放送モードを利用して，全PEのレジスタに自PEに対応するビットをセットする．このプロセッサがグローバル競合解消プロセッサとなる．一方，レジスタのどれかのビットがセットされていれば，既にローカル競合解消を終了したプロセッサが存在することになり，そのビットに対応するプロセッサがグローバル競合解消プロセッサである．この方式では，レジスタのチェックとビットのセットの間に，一度バスを解放するので，複数のPEでローカル競合解消が同時に終了した場合には，複数のビットがセットされる可能性がある．この場合にはPE番号の若いものをグローバル競合解消プロセッサとする．

3.4.4 認知–実行サイクルを通した動作

オーバヘッドとなり得る部分を以上のように処理することにより，並列化方式が本来持っている性能を十分に引き出すことが可能な実装となる．ここでは，本実装での，認知–実行サイクルを通した処理を追う．

個々のPEは，トークンキューとアクションキューという，二つの独立したキューを持つ．ルールの発火により生成されたトークンは，放送によりすべてのPEに送られトークンキューにプッシュされる．アクションキューは，WMEの生成，変更，削除などのWME操作要求を格納する．これらのWME操作要求は，ルール実行部の評価を行うPEが制御して送信する．

より詳細には，並列化された認知–実行サイクルは以下のようになる:

1. PEはトークンキューからトークンを1個取出し，TWIN方式に基づいた条件照合処理を行う．
2. もしもPEが直前の実行フェーズが完了したことを示す発火終了マークを見つけると，ローカルの競合集合から最も優先度の高いインスタンシエーションを選びだすローカル競合解消を行う．
3. 最初にローカル競合解消を終えたPE(以下このPEをPE_xと表す)は，他のすべてのPEに対して割込みをかけ，そのPEがグローバル競合解消を行うことを宣言する．他のPEは，各PEごとにそのPE内のローカル競合解消によって選択されたインスタンシエーションをPE_xに送信する．最も遅いPEが照合とローカル競合解消を終えてインスタンシエーションを送信するまで，他のPEは待つ．
4. グローバル競合解消の後，すべてのPEは実行フェーズに移行する．PE_xは，ルールのアクション部を評価し，アクション部にあるWME操作要求を，ランダムに選んだPEのアクションキューに対して送り込む．それぞれのPEは，アクションキューからWME操作要求を取出し，それぞれ処理する．
5. アクション部を評価し終えると，PE_xは発火終了マークを全PEに対して送信する．

3.5 システムソフトウェア

推論インタプリタは，並列コンピュータであるPrestoにとってはアプリケーション・プログラムの一部である．アプリケーションがハードウェア資源を効率的に利用でき，不正な処理によってシステム全体が停止しないために，アプリケーションとハードウェアの中間層にシステムソフトウェアが存在する．ここでは，Prestoのシステムソフトウェアについて述べる．

すべてのPEには同一のシステムソフトウェアがロードされる．システムソフトウェアの機能は，プロセッサ間通信，メモリ管理，ホストプロセッサ・エミュレーションに分類される．

プロセッサ間通信

プロセッサ間通信機能には，データ転送，データ転送バッファのフラッシュ，同期，割込制御などがあり，3.3.3で述べたようなバス制御をシステムコールによって実現する．

メモリ管理

メモリ管理機能としては，メモリ領域確保，メモリ領域開放，キューバッファの生成，キューバッファの削除がある．

ホストプロセッサ・エミュレーション

ホストプロセッサ・エミュレーションは，PE上で発生する入出力及びグラフィック関連のMS-DOSシステムコールをインターセプトし，ホストプロセッサに転送する．このMS-DOSシステムコールのエミュレーションにより，Prestoでは，MS-DOS向けの高水準言語処理システムを用いたプログラム開発が可能となっている．実際，本システムではプロダクションシステム推論インタプリタの開発にMicrosoftのMS-Cを用いている．

3.6 性能評価と考察

TWIN方式は，2章で述べたように，数10倍の速度向上を達成するだけのポテンシャルを持っている．Prestoのハードウェアとシステムソフトウェアが共にTWIN方式に適した環境を提供していること，及び，推論インタプリタが効率的に並列

表 3.1: ベンチマーク ES

	set A			*set B*		
PROGRAM NAME	BM-A1	BM-A2	BM-A3	BM-B1	BM-B2	BM-B3
Rules	4	10	20	1		
WMEs	800	1260	1800	400	1000	2000
Tokens/Node	100	63	45	200	500	1000
Comparisons	40000			40000	250000	1000000

化されていることを確認するために，いくつかのサンプルプログラムを用いて性能を評価した．まず，並列化方式の各部分の効果を顕著とするためのベンチマーク ES を作成し，これを用いて各部が無駄なく実装されていることを確認する．続いて，いくつかの実在の ES を用いて総合性能を評価する．

3.6.1 ベンチマーク ES による実験

まず，本並列化方式とその実装の各部分の効果をより顕著にできるベンチマーク ES を用いて実験を行った．ベンチマーク ES としては，表 3.1 に示すような 2 組のプログラム群を用いた．一方のプログラム群 (セット A) は，TWIN 方式の構造並列化効果を主に測定するためであり，もう一方のプログラム群 (セット B) は 2 入力ノードの並列化効果を主に測定するように作られている．

セット A には 3 本のプログラムがあり，それぞれ同種のルールを含んでいるがルール数が異なっている．ただし，2 入力ノードでのテスト回数はすべてのルールにわたって同一である．これらのプログラムにおいては，すべてのルールは独立に処理可能であり，また，ルールごとの処理時間も同一となる．したがって，本プログラム群を用いると，通信オーバヘッドや推論エンジンのうち並列化できない処理 (ルートノードの処理やクラスのチェック) の実装上のオーバヘッドを評価することができる．

一方，セット B にも 3 本のプログラムがあり，これらはルール数が同一 (1 個) であるが，2 入力ノードでのテスト回数が異なっている．これらプログラムは，どの PE がトークンを蓄積するかを決定するための処理に起因するオーバヘッドを評価するのに適している．

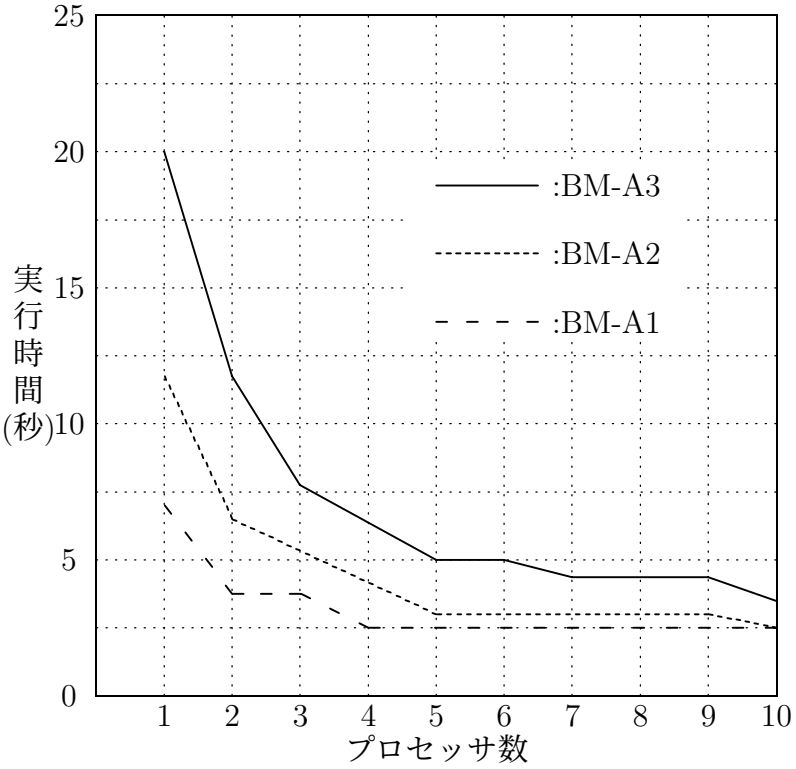


図 3.8: トップノード並列化方式でのベンチマーク結果

TWIN 方式から 2 入力ノード並列化を取除いたトップノード並列化方式に対してセット A を (構造並列化のみを評価するため)，TWIN 方式に対してセット B を，それぞれ用いて，10 プロセッサまでについて速度向上度を測定した．結果を図 3.8 及び図 3.9 に示す.

プログラム BM-A1 では，4 台のプロセッサにより 2.7 倍の速度向上が達成されている．プログラムは 4 つしかルールがないため，それ以上の数のプロセッサを用いてもこれ以上大きな速度向上は望めない．より多くのルールを持つ BM-A2 と BM-A3(それぞれ 10 個と 20 個) では，どのような PE 数においても BM-A1 の速度向上を上回っており，プロセッサが 10 個の時に 5–6 倍の速度向上が得られている．BM-A2 と BM-A3 の結果から，オーバヘッド要因がいくつか考えられる．高水準言語での関数の呼出しメカニズムに起因するオーバヘッドが最も大きい．これは TWIN 方式に限らずどのような方式においても同様であろう．更に，PE 間の通信によるオーバヘッドが考えられる．しかし，図 3.10 からわかるように，これは条件照合フェーズの処理時間の 2%以下であり，全体の処理時間にはほどんど影響しない.

TWIN 並列化方式を用いた場合の結果 (図 3.9) は，トップノード並列化方式の

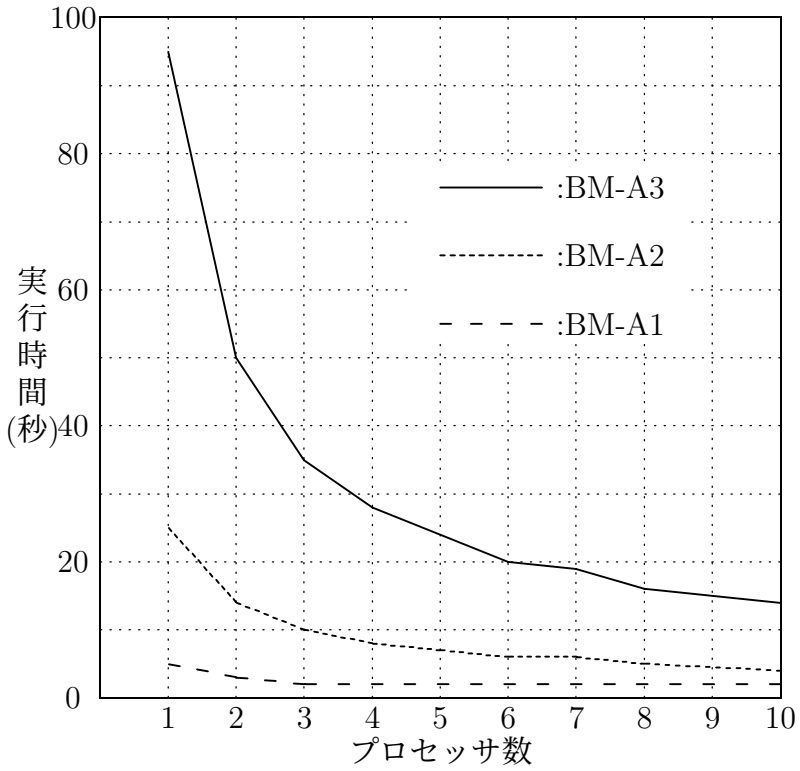


図 3.9: TWIN 方式でのベンチマーク結果

結果 (図 3.8) と同様に良好な速度向上が得られている．BM-B3 ではプロセッサが 10 個の場合に約 7 倍の速度向上度が得られている．BM-B3 に対する速度向上度が BM-B1 や BM-B2 に対するより大きいのは，BM-B3 の 2 入力ノードでのトークン突合回数がそれぞれ 25 倍及び 4 倍大きいからである．これら 3 本のプログラムも，セット A のプログラムと同様のオーバヘッド要因を持つ.

3.6.2 経路探索 ES による総合性能評価

つぎに，2.7 節で用いたと同じ小規模な経路探索 ES プログラムにより実験を行った．総合的な速度向上度は，2.7 節における条件照合フェーズの速度向上度とほぼ同一となった.

ここでは更に，認知–実行サイクル並列化時の各フェーズの処理時間の比を明らかにし，TWIN 方式の特徴である 2 入力ノードの並列処理による効果を確認するために，各フェーズの処理時間比を調べた．結果を図 3.11 に示す．図は推論時間全体を 100%として，各フェーズが推論時間の何%を占めるかを表している．また，M_d は 2 入力ノード処理の並列度である．$M_d = 1$ の場合は 2 入力ノード処理を並

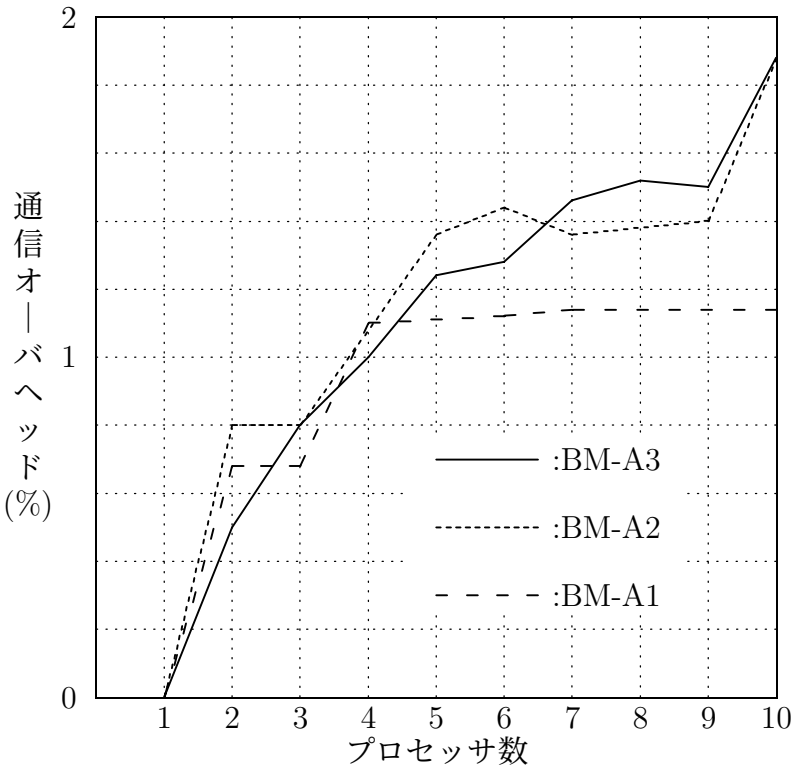


図 3.10: 通信オーバヘッド

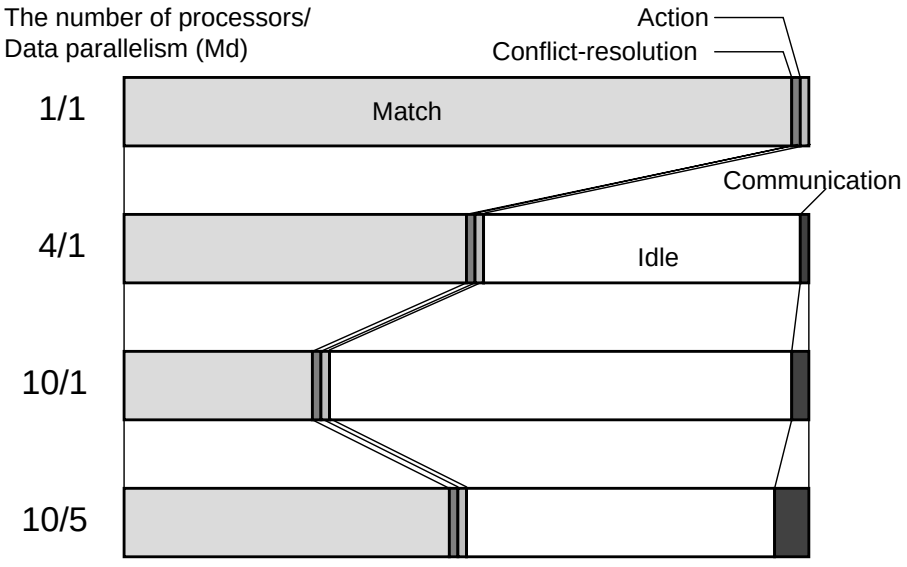


図 3.11: 各フェーズの実行時間比

表 3.2: 負荷ばらつき係数

プロセッサあたりの粒の数	負荷ばらつき係数
1	4.3
2	2.6
3	1.7
4	1.5
6	1.3

列化しない場合，すなわちトップノード並列化方式を示す．また，図の最上段，つまり $M_d = 1$ かつプロセッサ数 1 の場合は，逐次処理にあたる．

トップノード並列化方式では，図からわかるようにプロセッサ数が大きくなると条件照合フェーズの負荷ばらつきによるアイドル時間が非常に大きくなる．一方，TWIN 方式によって 2 入力ノードをそれぞれ 5 個のプロセッサで並列処理した場合 (Md=5) には，負荷ばらつきが減少しアイドル時間が抑えられている．これは次のように説明できる．負荷ばらつきは，並列処理の処理単位の処理量ばらつきと，プロセッサに割り当てられる処理単位の数とに依存する．プロセッサあたりの処理単位の数が少なければ個々の処理単位の処理量ばらつきの影響が顕著となり，逆に処理単位が多ければプロセッサでの処理時間は平均化される．処理単位の最大処理時間と平均処理時間との比を意味する負荷ばらつき係数は，本 ES の場合表 3.2 のようになる．

一つのトークンが RETE ネットワークに入力されたときに，それに対して条件照合を行うルールの数は（ルール数×条件要素数）／クラス数でおおまかに計算でき，本 ES の場合は約 12 である．トップノード並列化方式の場合，本 ES では 5 プロセッサ以上のときには各プロセッサが 2 個しか処理単位を持たないことになり，負荷ばらつきが非常に大きくなる．一方，TWIN 方式ではデータ並列融合の効果によりプロセッサ当たりの処理単位の数はデータ並列数倍に増加するため，負荷ばらつきが抑えられる．

また，同図に示されたように通信オーバヘッドは全体の 1%〜4%である．トークン転送に必要な時間は一定のため，プロセッサ数が増加して全体の処理時間が短くなると比率としては増加するが，速度向上の阻害要因にはなっていない．

更に，本方式で並列化している競合解消フェーズ・実行フェーズの高速化も総合

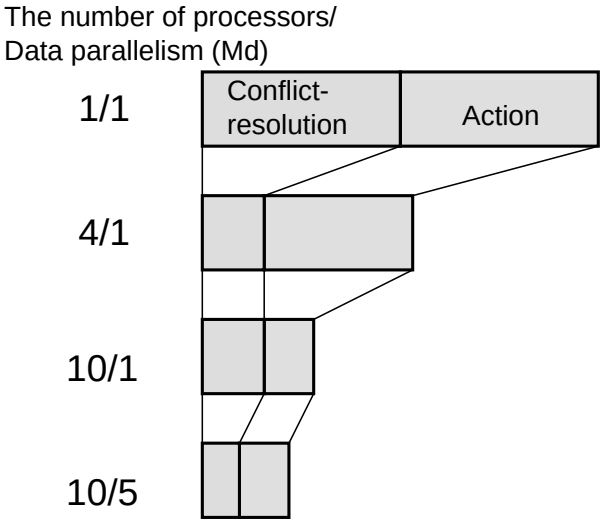


図 3.12: 競合解消／実行フェーズ処理時間の変化

的な速度向上に寄与している．競合解消フェーズ・実行フェーズの処理時間の変化を図 3.12 に示す．両フェーズとも本方式の並列処理効果によりプロセッサ数の増加とともに処理時間が減少していることがわかる．競合解消は並列化不能なグローバル競合解消を含んでいるが，前述のようにローカル競合解消の終了したプロセッサから順次インスタンシエーションを転送しているため，10 プロセッサで 5 倍程度の高速化が実現できている．一方，実行フェーズでは，1 認知サイクルでの WME 更新は 2〜4 程度のため，並列化による速度向上は約 3 倍である．これら競合解消／実行フェーズの並列化を行わない場合には，プロセッサ増加に伴って競合／実行フェーズの処理時間の割合が増加し，10 プロセッサ時には 10%にも達する．すなわち，これらのフェーズの並列化により照合のみの並列化に加えて更に 1 割近い速度向上が得られていることになる．

3.6.3 8 クイーン問題による評価

前述のサンプルプログラムに加え，8 クイーン問題を解くプログラムを実行させてみた．このプログラムは，8 クイーン問題の制約条件をプロダクションルールの条件部に書いたもので，条件照合を通過した個々のインスタンシエーションがそれぞれ問題の解となる．PS における条件照合を制約充足機構として利用することの好例となっている．結果を表 3.3 に示す．4 台の PE を用いて 2.2 倍の速度向上

表 3.3: 8Queen 問題の処理時間 (4PE の場合)

# of PEs	1	4
Time (sec)	54	25

が得られている．

3.7 まとめ

本章では，プロダクションシステムの条件照合フェーズ並列化方式である TWIN 方式に好適な並列コンピュータ Presto を提案した．Presto は，ローカルメモリを持つ PE をバスにより結合した疎結合型並列コンピュータである．最大 10 個の PE を持つことができ，それぞれの PE は汎用のマイクロプロセッサとメインメモリ，専用開発した 2 品種の ASIC から構成されている．部品数が少なく共有メモリやキャッシュメモリも不要のため低廉で小型，高性能なシステムを実現できている．

バスには TWIN 方式及びそれを利用した推論インタプリタ全体の並列化方式が効率良く実装できるように，DMA によるデータ転送，放送，同期，割込み制御などの機能を備えている．TWIN 方式は，通信頻度が小さいため，このような低廉にハードウェアを構成可能なアーキテクチャによって十分な性能を得ることが可能である．

Presto により推論インタプリタを並列処理するために，条件照合フェーズの TWIN 方式による並列化をベースとした推論インタプリタ全体の並列化方式を提案した．本並列化方式では，競合解消フェーズはプロセッサ内のローカル競合解消とプロセッサ間のグローバル競合解消とに分割し，実行フェーズでは複数の WME 操作をプロセッサに分散することで，条件照合以外のフェーズの並列化も図っている．また，RETE サブネットと WM をプロセッサに分散配置し，ルール実行部を重複して配置することで，競合解消フェーズ・実行フェーズでの通信量も削減し，かつメモリ消費を抑えた．更に，RETE サブネットの静的割り付けにおいて，処理量を予測して負荷を均等化する割り付け法を採っている．

この並列化方式を実装する上でオーバヘッド要因となりそうな項目を洗いだし，それらの効率的な実装法も検討した．Presto に持たせたバスの諸機能を最大限に活用し，フェーズ移行，同期，グローバル競合解消プロセッサ決定を効率良く処

理できるような実装を考案した.

総合性能とオーバヘッドを測定するために，いくつかの ES を実際に走行させて性能評価を行った．各フェーズの処理時間の測定により，通信オーバヘッドは最大でも約 4%であり速度向上への影響は無視できることを確認した．また，競合解消フェーズ・実行フェーズの並列化により条件照合並列化による速度向上に加えて更に一割近くの速度向上が得られた.

第4章　ニューラルネットシステムの並列化

ニューラルネット (NN:articifial neural network) システムは，人間の脳の卓越した問題解決能力にヒントを得て，その神経細胞 (neuron) を模擬した素子の結合により情報処理を行うシステムである [6]．実際には，素子の機能をソフトウェアに置き換えて，ディジタルコンピュータにより構成することが多い．一般に NN システムは学習機能を持ち，これによって，これまでの情報処理技術では困難とされていた問題領域のいくつかをうまく扱うことができる．パターン認識や連想記憶が代表的な適用領域であり，様々な分野への応用が期待されている．

しかしながら，NN システムにおける学習は，試行錯誤の反復によって行われるため，そのための計算量が大きなものとなってしまい，NN システムの応用を制限する場合も多い．特に，代表的な教師付き学習アルゴリズムであるバックプロパゲーション (BP:back propagation) は，正答例の提示に基づき内部のニューロンそれぞれの結合係数を微少変化させることにより学習を行う．1 度の例示に対し多数のニューロンの結合係数を計算し更新する必要があることに加え，学習の完了とみなすことができる結合係数の変化の収束に到達するまでには，非常に多数の正答例提示が必要となるため，小規模な問題に対してもその計算量は膨大なものとなってしまう．

そこで本章では，NN の学習を高速に行うことを目的とした，BP の並列化方式について論ずる．まず，NN と BP について説明し，BP が連続行列演算の一形態であることを明らかにする．続いて，この連続行列演算を並列化した場合の通信量と計算量を定式化し，その下限値を導出する．この下限値とは，それより小さな通信量及び計算量で連続行列演算を処理できる並列化方式は存在しないことを意味する．次に，実際に 2 次元アレイ (トーラス) 型並列コンピュータとバス結合型並列コンピュータに対する並列化方式を提案し，2 次元アレイ型に対する方式が前述の下限値を達成していることを示す．下限値を達成していることから，この

方式より良い方式は他に存在しないと言える．すなわち，この並列化方式は最適並列化方式となっている．また，バス結合型では 1 対 1 のデータ転送によってもバスが占有されるという制約のため，前述の下限値では並列化を実現することはできない．提案方式でも通信ステップ数が下限値とはなっていないが，従来の方式がプロセッサ数に比例した通信ステップ数を要したのに対し，本方式ではプロセッサ数の平方根オーダに抑えられていることを示す．

4.1 多層ニューラルネットとバックプロパゲーション

多層 NN は，入力層，出力層及び両者の間に 1 層以上の隠れ層を持ち，隣合う層のニューロンユニット (NU:neuron unit) がある重みを持って全結合されたフィード・フォワード型の NN である．図 4.1 に多層 NN の一例を示す．各層の NU は，前層出力と結合重みとの積和値にシグモイド関数などの非線形演算を施した値を出力とする．入力信号が各層を順に伝播して最終的な出力値が得られる．これを前向き伝播と呼ぶ．この NN の入力を例えば画素パターンに対応させ，出力を図形や文字のコードに対応させれば，パターン認識を行うシステムを構築できる．この場合の認識動作は前向き伝播によって行われる．

多層 NN を前向き伝播によって所望の動作をさせるには，あらかじめ内部の各 NU を結合する重みを適切な値に設定する必要がある．このために，期待される正答の例示を繰り返し，多層 NN に自律的に結合重みを設定させる学習手法が用いられ，そのアルゴリズムの代表的なものが BP である．BP では，前向き伝播により得られた出力と期待される出力信号 (教師信号) との誤差を極小化するように，出力層から入力層に向かって順に結合重みの微修正 (後向き伝播) を行う．この前向き伝播・後向き伝播のサイクルを多数の教師信号に対して繰り返すことにより学習が行われる．

以下では，本章を通して用いる記号を定義した後，前向き伝播と後向き伝播からなる BP の計算法について詳述する．

4.1.1 記号の定義

NU の個数はすべての層で N 個である NN システムを考える．

前向き伝播，後向き伝播の処理詳細は後述するとし，ここでは，記号の定義を

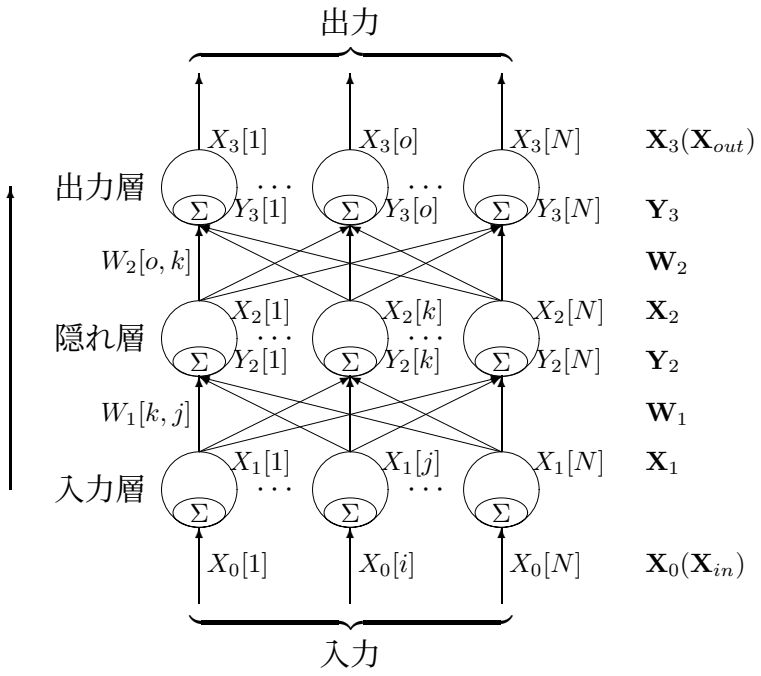


図 4.1: 多層ニューラルネット (前向き伝播)

述べる．以後の計算式で用いられる記号を図 4.1 及び図 4.2 に示す.

第 l 層の i 番目の NU の出力を $X_l[i]$ と書く．$l = 1$ が入力層を表す．更に図の例では，$l = 2$ が隠れ層，$l = 3$ が出力層となる．ただし，特別な場合として $X_0[i]$ はシステムに対する入力とする．また，自明であるが $X_3[i]$ はシステムの出力となる.

第 l 層 i 番目の NU から第 $l + 1$ 層 j 番目の NU への結合重みを $W_l[j, i]$ と表す. 例えば，$W_1[3, 5]$ は第 1 層 5 番目の NU から第 2 層 3 番目の NU への結合重みを表している．$Y_l[j]$ は，第 l 層 j 番目の NU において，第 $l - 1$ 層の各 NU からの出力を結合重み $W_{l-1}[i, j]$ に従って加重し総和した値である.

本論文では，出力，結合重み，加重総和をベクトルや行列として記述する場合もある．すなわち，第 l 層の各 NU の出力をベクトルとしてとらえ,

$$\mathbf{X}_l = \begin{pmatrix} X_l[1] \\ \vdots \\ X_l[i] \\ \vdots \\ X_l[N] \end{pmatrix} \tag{4.1}$$

と表現する．同様に，加重総和は，

$$\mathbf{Y}_l = \begin{pmatrix} Y_l[1] \\ \vdots \\ Y_l[i] \\ \vdots \\ Y_l[N] \end{pmatrix} \tag{4.2}$$

となる．結合重みは $N \times N$ の行列となり，以下のように表現する．

$$\mathbf{W}_l = \begin{pmatrix} W_l[1,1] & \cdots & W_l[1,i] & \cdots & W_l[1,N] \\ & \ddots & & & \\ W_l[j,1] & \cdots & W_l[j,i] & \cdots & W_l[j,N] \\ & & & \ddots & \\ W_l[N,1] & \cdots & W_l[N,i] & \cdots & W_l[N,N] \end{pmatrix} \tag{4.3}$$

図 4.2 の $d[i]$ は，入力に対して期待される出力値，すなわち教師信号であり，学習の際に入力と対になって与えられる．そのベクトル表現は，

$$\mathbf{d} = \begin{pmatrix} d[1] \\ \vdots \\ d[o] \\ \vdots \\ d[N] \end{pmatrix} \tag{4.4}$$

となる．$\delta_l[i]$ 及び $\theta_l[i]$ は，教師信号と前向き伝播によって得られた出力との差に基づいて計算される誤差値である．この誤差値に基づいて結合重みを更新する，$W_l[i,j]$ の新たな重みを $W_l'[i,j]$ と表す．また，これらの記号もこれまで述べたものと同様にベクトル及び行列表現する場合がある．それぞれ，$\boldsymbol{\delta}_l$，$\boldsymbol{\theta}_l$，$\mathbf{W}_l'$ で表す．

4.1.2 前向き伝播

システムとしての本来の動作，すなわち入力が与えられた時にそれに対する出力を得ることは，前向き伝播により行われる．また，BP での学習に際しても，教師データとシステムが計算して得た出力との差分に基づいて結合重みを更新するために，やはり前向き伝播が行れる．

前向き伝播では，第 $l+1$ 層の出力はその下位すなわち第 l 層の出力を用いて次のように計算される．まず，第 $l+1$ 層 j 番目の NU において，第 l 層の NU 出力

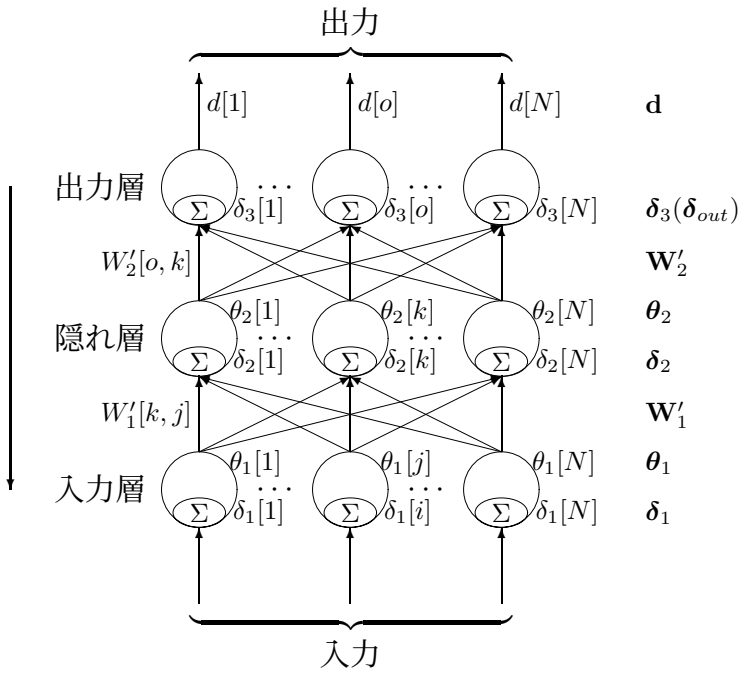


図 4.2: 多層ニューラルネット (後向き伝播)

にそれぞれの結合の重みを加重し総和を計算する．

$$Y_{l+1}[j] = \sum_{i=1}^{N} X_l[i] \cdot W_l[j,i] \tag{4.5}$$

これを行列で表現すれば,

$$\mathbf{Y}_{l+1} = \mathbf{W}_l \cdot \mathbf{X}_l. \tag{4.6}$$

となる．次に，この積算結果に非線形関数 F を作用させることによって 第 $l+1$ 層の NU の出力を得る．すなわち,

$$X_{l+1}[i] = F(Y_{l+1}[i]), \tag{4.7}$$

を計算する．非線型関数 F の引数としてベクトルを許せば，行列を用いて,

$$\mathbf{X}_{l+1} = F(\mathbf{Y}_{l+1}), \tag{4.8}$$

とも書ける．非線形関数 F としてはシグモイド関数

$$F(Y) = \frac{1}{(1 + exp(-Y))} \tag{4.9}$$

などが一般的に用いられる．

上記の (4.6) 式及び (4.8) 式により 1 層分の伝播が行われる．これを入力層から出力層に向かって繰り返すことによって，システムの出力値が得られる．

4.1.3 後向き伝播

後向き伝播では，各層での誤差を計算しそれに基づき重みを更新する．各層での誤差は上位層での誤差から計算される．

出力層での誤差 δ_{out} は以下のように計算する．

$$\delta_{out}[i] = (d[i] - X_{out}[i])F'(Y_{out}[i]), \tag{4.10}$$

ここで $d[i]$ は i 番目の出力 NU に対する教師信号，F' は非線形関数 F を一階微分した関数である．関数 F' が引数としてベクトルを取ることを許せば，以下のようにも書ける．

$$\boldsymbol{\delta}_{out} = M\left((\mathbf{d} - \mathbf{X}_{out}), F'(\mathbf{Y}_{out})\right), \tag{4.11}$$

ただし，関数 $M(\mathbf{a}, \mathbf{b})$ は，ベクトル $\mathbf{a}$ と $\mathbf{b}$ のそれぞれ対応する要素をかけあわせたものを結果のベクトルとして返す関数とする．

また，第 l 層での誤差 δ_l は以下で計算される．

$$\theta_l[j] = \sum_{i=1}^{N}(\delta_{l+1}[i] \cdot W_l[i, j]) \tag{4.12}$$

$$\delta_l[i] = \theta_l[i] \cdot F'(Y_l[i]), \tag{4.13}$$

これをベクトル・行列を用いれば以下のように書ける．

$$\boldsymbol{\theta}_l = \mathbf{W}_l^T \cdot \boldsymbol{\delta}_{l+1}, \tag{4.14}$$

$$\boldsymbol{\delta}_l = M\left(\boldsymbol{\theta}_l, F'(\mathbf{Y}_l)\right), \tag{4.15}$$

ここで， $\mathbf{W}^T$ は重み行列 $\mathbf{W}$ の転置行列である．

以下で計算される $W_l[j, i]'$ を新たな重みとして重み行列を更新する．

$$W_l[j, i]' = W_l[j, i] + \eta \cdot \delta_{l+1}[j] \cdot X_l[i], \tag{4.16}$$

行列表現では，

$$\mathbf{W}_l' = \mathbf{W}_l + \eta \cdot \boldsymbol{\delta}_{l+1} \cdot \mathbf{X}_l^T \tag{4.17}$$

となる．ここで η は学習係数と呼ばれ，重みの更新率を制御する定数である．

以上のように，BP での演算は，行列・ベクトルの積と行列やベクトルの個々の要素に対する演算とから成る．行列・ベクトル積の演算量は多いため，BP の演算ステップ数の多くは (4.6) 式と (4.14) 式とで占められる．

4.2 従来のバックプロパゲーション並列化方式

BP の並列処理のために既に数多くの並列化方式が提案されている．疎結合型プロセッサを対象とした方式 [58, 87, 19, 4, 46, 70] が多いが，そのほとんどが絶対性能を主眼としている．現実の並列コンピュータでは装備できるプロセッサ数には限りがある上，経済的にも限られた数のプロセッサを有効に活用できる必要があるため，並列化方式の性能としては，速度向上度の限界値だけではなく，プロセッサ利用効率 (速度向上度 / プロセッサ数) が重要となる．このような観点からは，これら方式は必ずしも満足できるものではない．

例えば，多層 NN の 1 層を 2 次元アレイプロセッサの 1 行に対応させる Kung らの方式 [58] では同時に計算を行うのは常に 1 行分のプロセッサだけのため，他の行のプロセッサはアイドル状態となる．このため，プロセッサの利用効率は非常に低いものとなってしまう．

Suzuki らの方式 [87] では，まず，入力ベクトルを 2 次元アレイプロセッサの最上行より順次下方に転送して部分積和を計算し，次に部分積和を横方向に転送して最右列のプロセッサで集計し結果を得るため，1 層ごとにすべてのプロセッサが並列動作できる．しかし，最上行と最下行のプロセッサでの入力ベクトルの到着及び最左列と最右列のプロセッサで部分積和の到着にそれぞれ時間差があるためにプロセッサにアイドル時間が生じ，やはりプロセッサ利用効率が低いものとなっている．

これらに対し，久野らが提案する，格子状配置したメモリ・ユニットに対し行方向や列方向に同報書き込みできる特殊なアーキテクチャを用いた方式 [59] はプロセッサ利用効率が高い．しかし，プロセッサ数の 2 乗の数のメモリ・ユニットが必要なため，プロセッサ数が大きい場合に実現が困難となる．

これら NN を直接に意識した方式とは別に，4.1 節で述べたように BP を行列演算の特殊な形態として並列化方式を考えることもできる．BP では前向き伝播においても後向き伝播においても，行列とベクトルの積が主要な処理となる．また，それら演算を，入力層から出力層に向けて，あるいはその逆向きに繰り返すことが必要であり，更に，演算対象として，前の層での演算結果が含まれる．このように前段階の結果を用いて連続的に行われる行列演算を，連続行列演算と呼ぶことにする．

一般的な行列演算を高速に処理するには並列処理の適用が効果的であり，すで

に多くの並列化の研究がなされている [61, 47]．しかし，連続行列演算の場合には，計算結果が次の演算の被演算ベクトルとなるため，ベクトルの割り付けはその直前の演算結果の割り付けによって制限され，一般の行列演算に適した並列化方式が必ずしも最適とは限らない [96].

並列処理での速度向上は，処理の分割に伴う処理量増加，各プロセッサの処理量ばらつき，通信オーバヘッドに依存する．行列演算は本質的に規則的な計算であるため，処理量増加と処理量ばらつきを生じさせないことは容易に実現できる．したがって，行列演算の最適な並列化法とは，これらに加えて通信オーバヘッドが最小となるマッピング及び計算方法のことだといえる．連続行列演算の場合には，計算結果が次の演算の被演算ベクトルとなるため，被演算ベクトルと結果ベクトルの並列プロセッサ上での割り付けを同じにしなければならないという制約が加わっていることになる．

次節以降では，BP を連続行列演算として定式化し，疎結合型並列コンピュータを対象に，その並列化における通信量と計算量について体系的に明らかにし，それらの下限値を導く．これに基づいて，BP の最適並列化方式について論ずる．まず，通信ステップ数を定式化してその最小値を求めることにより，通信量の下限値とそれを達成するための条件を導出する．同時に演算ステップ数を定式化し，前記条件のもとでは計算量も下限値を取り得ることを示す．次に，連続行列演算としてとらえた場合の BP の特性を明らかにし，その特性のもとでの通信ステップ数の下限値と，それを達成するための条件を，上と同様の手順により導く．更にこの議論に基づき，2 次元アレイ (トーラス) 型並列コンピュータとバス結合型並列コンピュータとに対する最適並列化方式について述べるとともに，2 次元アレイに対する方式が通信オーバヘッドの下限値を達成できることを示す．

4.3 連続行列演算の並列化における通信量と計算量の下限値

本節では，連続行列演算を並列化した場合の通信量及び計算量の下限値と，それを達成するための条件を明らかにする．なお，ここでは，通信量を通信のステップ数で，計算量を演算のステップ数で，それぞれ代表させることにする．また，簡単のために正方行列を対象として議論を進める．

4.3.1 連続行列演算の並列処理

連続行列演算は，N 行 $\times$ N 列の行列 $\mathbf{W}$ と入力ベクトル $\mathbf{X}$ (要素数 N) とをかけあわせ結果ベクトル $\mathbf{Y}$ (要素数 N) とする計算

$$\mathbf{Y} = \mathbf{W} \cdot \mathbf{X} \tag{4.18}$$

を行い，その結果を被演算ベクトルとして更にかけ算を繰り返す演算である．この連続行列演算を P 個のプロセッサで並列処理することを考える．ここで，$\mathbf{X}$ の i 番目の要素を $X[i]$，$\mathbf{Y}$ の j 番目の要素を $Y[j]$，$\mathbf{W}$ の j 行 i 列目の要素を $W[j,i]$ とそれぞれ表すと，(4.18) 式は以下のように書ける．

$$Y[j] = \sum_{i=1}^{N} X[i] \cdot W[j,i] \tag{4.19}$$

この演算を並列プロセッサ全体で分散して行うことを考える．ただし，行列の要素は計算開始前にあらかじめ静的にプロセッサに割り付けておくことを前提とする．ここで，以下の記号を定義する．

添字 $*$: 添字として有効な範囲の任意の値．$W[*,i]$ と書いた場合には $W[1,i]$〜$W[N,i]$ のうちどれでも良いことを表す．

$\mathcal{I}_n[j]$: j 行目の行列要素のうちプロセッサ n に割り付けられた列添字の集合．例えば，1 行目の行列要素のうち $W[1,1], W[1,3], W[1,5]$ がプロセッサ 2 に割付けられているとすると，$\mathcal{I}_2[1] = \{1,3,5\}$ となる．

$\mathcal{J}_n[i]$: i 列目の行列要素のうちプロセッサ n に割り付けられた行添字の集合．

この行列演算において，結果のベクトル要素は (4.19) 式に示すように，入力ベクトル要素と行列要素との積の総和により計算される．これを複数のプロセッサで分割して処理するのだから，各プロセッサは自プロセッサが持つ範囲の要素に対する積とその総和 (部分関和) を計算し，その後にプロセッサにまたがった総和を行って最終的な結果を得ることになる．また，入力ベクトルの要素は，このような処理のために，関係するすべてのプロセッサに複写されて配布される必要がある．このような動作は，行列やベクトルの要素をプロセッサにどのように割り付けるかにかかわらず，普遍的に必要となる．したがて，行列・ベクトル積の並列演算には，以下に示す 3 フェーズからなる処理が必要となる．

配布

入力ベクトルの要素 $X[i]$ を，行列要素 $W[*,i]$ が割り付けられたプロセッサすべてに転送する．

部分積和

各プロセッサは自分に割り付けられた範囲の $W[j,i]$ に対し

$$y_n[j] = \sum_{i \in \mathcal{I}_n[j]} (X[i] \cdot W[j,i]) \tag{4.20}$$

を計算する．ただし n はプロセッサの番号を表す．$y_n[j]$ は 結果ベクトル要素 $Y[j]$ の部分積和になっている．

集計

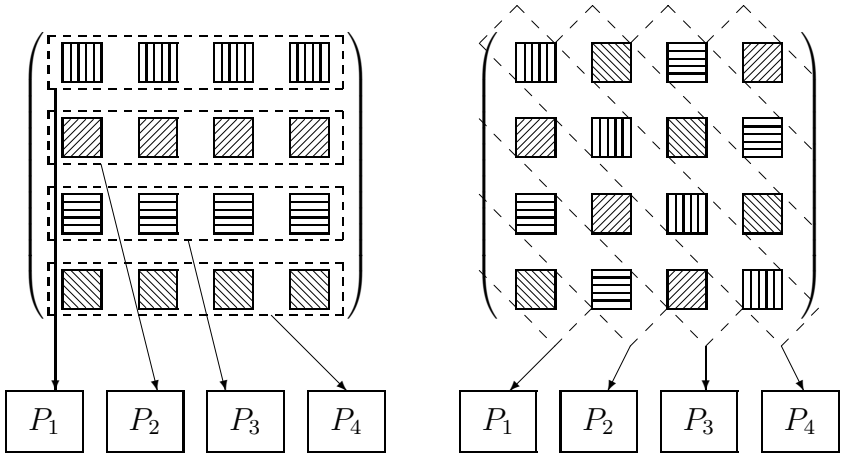
$Y[j]$ の計算を担当するプロセッサは，部分積和 $y_n[j]$ を持つすべてのプロセッサからその部分積和を転送してもらい，総和して $Y[j]$ を計算する．

次の節では，この 3 つのフェーズの処理内容について考察し，それぞれの通信ステップ数と演算ステップ数を定式化する．

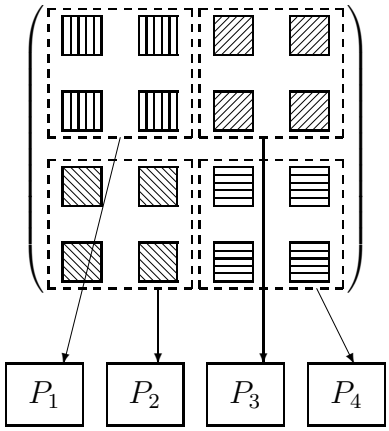
4.3.2 通信ステップ数と演算ステップ数の定式化

上で述べたように，ここでは行列・ベクトル積を並列計算する際の通信ステップ数と演算ステップ数を定式化する．行列の大きさは $N \times N$ とし，プロセッサ数を P と書く．また，各プロセッサには前の計算によって M 個の結果要素が得られており，これをそのまま入力要素として使用するものとする．

定式化のため，上記の 3 つのフェーズを通信ステップと演算ステップの観点から考察する．配布フェーズでは，ベクトル要素がそれを必要とするすべてのプロセッサに配布される必要があり，このためにの通信が発生する．ただし，必要な要素を自プロセッサ内に既に保持している場合に限り通信は必要ない．一つのプロセッサが他のプロセッサ上にあるベクトル要素を多く必要とするほど，通信ステップ数も増大する．どのベクトル要素をどのプロセッサが必要とするかは行列要素の割り付けに依存する．部分積和フェーズでは，自プロセッサが持つ行列要素に対して積和を計算する．このため通信は発生せずに演算のみが行われる．演算量は，プロセッサが持っている行列要素に依存する．これは割り付け法によって決まる．集計フェーズでは，各プロセッサから部分積和を収集し，それを総和



(a) 行分割　　(c) 対角線方向の分割



(b) メッシュ分割

図 4.3: 行列要素割り付けの例

して結果を得るという処理を行うため，通信と演算の両方が発生する．一つの結果を得るために，多くの部分積和を収集する必要があれば通信も多くなるし，また，総和を計算するための演算量も増大する．これらも割り付け法に依存する．

このように，通信ステップ数及び演算ステップ数は，行列及びベクトルの要素をどのように分割してプロセッサ上に割り付けるかに依存していることがわかる．このことから，本定式化では，まず行列要素の割り付けに関するパラメータを定義し，つぎに，それを用いて通信ステップ数と演算ステップ数を導出する．

行列要素の割り付けに関するパラメータ

図 4.3 に具体的な割り付けの例をいくつか示す．なお，この例では $N = 4, P = 4$ である．図に示したように様々な割り付け法が考えられるが，本定式化では，通信量と計算量の下限値を求めることが目的であるため，これらステップ数が全プロセッサにわたり均一となるような割り付けのみを考える．なぜなら，同一の総ステップ数のもとでは，ステップ数が不均一な場合は均一な場合より必ず通信量や計算量が大きくなるからである．更に，割り付けに関するパラメータも様々なものがあるが，演算ステップ及び通信ステップの導出のためには，割り付けられた要素の数に関わるパラメータだけがわかれば良い．

割り付けに関するパラメータを以下のように定義する．

行数 K_r

一つのプロセッサに割り付けられた行の数である．どの列かにかかわらず一つでもその行の要素を持ってれば 1 と数える．例えば，プロセッサに行列要素 $W[1,1]$, $W[1,2]$, $W[2,1]$, $W[2,2]$ が割り付けられているならば，第 1 行と第 2 行の要素を持っているため $K_r = 2$ となる．一方，行列要素 $W[1,1]$, $W[2,2]$, $W[3,3]$, $W[4,4]$ が割り付けられていれば，1 行〜4 行の要素を持っていることになり $K_r = 4$ となる．

列数 K_c

一つのプロセッサに割り付けられた列の数である．上と同様に，どの行かにかかわらず，一つでもその列の要素を持ってれば 1 と数える．

1 行あたりの要素数 K_e

一つのプロセッサに割り付けられ，かつ，同一の行に属する行列要素の個数である．任意の割り付けでは，行によって割り付けられる行列要素数が異なる場合も考えられるが，前述のように均一な割り付けを前提とすれば，プロセッサに割り付けられたどの行も同じ個数だけの行列要素を含むことになる．プロセッサが割り付けられた列の数は上述のとおり K_c であり，1 行にすべての列が含まれているとしてもたかだかこの値となるので，K_e と K_c の間には $K_e \leq K_c$ なる関係が成り立つ．

要素数 K

一つのプロセッサに割り付けられた行列要素の個数である．

本定式化はプロセッサに割り付けられた要素数や行と列の数のみをパラメータとして導出を行うため，具体的にどの要素をどのプロセッサに割り付けるかに依存しない定式化となっている．

図 4.3(a) は行分割の場合で，それぞれの行の行列要素すべてを一つのプロセッサに割り付ける．この場合は，個々のプロセッサは 1 行の中の行列要素しか持たないから $K_r = 1$ である．一方，割当てられた行列要素の列は，どのプロセッサも 1〜4 までを含むため，$K_c = 4$ となる．要素数は明らかに $K_e = 4$ である．図 4.3(b) はメッシュ分割である．この場合，それぞれのプロセッサは，2 行分かつ 2 列分の行列要素を含む．したがって，$K_r = K_c = 2$，また，割り付けられた行に対しては，それぞれ二つの列の行列要素を含むため $K_e = 2$ となる．図 4.3(c) のように対角線方向に分割した場合には，各プロセッサは，4 つの行にわたりそれぞれ異なった列の行列要素を含むことになる．したがって，$K_r = 4$ であり $K_c = 4$ である．各行の要素は一つずつなので $K_e = 1$ となる．当然ながらすべての場合において $K = 4$ である．

通信ステップ数の導出

上記パラメータを用いた通信ステップ数の算出式を導出する．プロセッサ間での 1 個のデータの転送を 1 通信ステップと数える．転送回数は，送信側でも受信側でも計数できるが (当然ながら総数は同一となる)，ここでは受信側でとらえることにする．各フェーズにける個々のプロセッサが受信するデータ数を求め，それを合計することで 1 層あたりの総通信ステップ数を求める．

配布

K_c 列分の行列要素があるから，部分積和計算のためにその列に対応する個数の入力ベクトル要素が必要である．各プロセッサは M 個の要素を最初から保持しており，最善の場合はこれらをすべて利用可能なため，配布での通信ステップ数 C_d は下式となる．

$$C_d = K_c - M \tag{4.21}$$

部分積和

各プロセッサが独立に計算するためデータ転送は不要であり 0.

集計

各行の要素は K_e ずつに分割されているため，一つの結果要素の計算には N/K_e の部分積和が必要となる．自プロセッサ内にも 1 個の部分積和を保持しているため，受信すべき部分積和数は $N/K_e - 1$ 個．各プロセッサは M 個の結果要素を計算するので，集計での通信ステップ数 C_a は下式となる．

$$C_a = M\left(\frac{N}{K_e} - 1\right) \tag{4.22}$$

ただし，ここでは簡単のために N は K_e で割り切れるものとしている (以降も同様)．

以上から，各プロセッサで 1 回の行列・ベクトル積の計算に必要なデータ転送回数 C は下式で表される．

$$C = C_d + C_a = K_c + M\left(\frac{N}{K_e} - 2\right) \tag{4.23}$$

演算ステップ数の導出

演算ステップは，各プロセッサでの加算の回数及び乗算の回数を表す．一般に加算と乗算ではその処理時間が異なるため，それぞれを個別に算出する．各フェーズでの演算ステップ数は以下のようになる．

配布

演算は不要なため 0.

部分積和

各プロセッサは K 個の要素を持つため，乗算は K 回行われる．また，1 行あたり K_e 個の要素を持ちそれが K_r 行あるため，加算は $K_r(K_e - 1)$ 回行われる．

集計

1 個の結果要素を得るためには N/K_e 個の部分積和の総和が必要となる．各プロセッサは M 個の結果要素を計算するので，$M\,(N/K_e - 1)$ 回の加算が必要である．

以上より，1 回の行列・ベクトル積に必要な乗算及び加算のステップ数 A_m，A_s はそれぞれ下式のようになる．

$$A_m = K, \tag{4.24}$$

$$A_s = K_r(K_e - 1) + M\left(\frac{N}{K_e} - 1\right). \tag{4.25}$$

4.3.3 通信ステップ数と演算ステップ数の下限値

上での導出から，通信ステップ数及び演算ステップ数はともに，行列要素の分割とそれらのプロセッサへの割り付けにより変化することがわかった．ここでは，これらのステップ数がどのように変化するかついて考察し，その下限値を求め，更にそれを達成する条件を導く．ただし，いずれのプロセッサも一度に 1 個のデータを一定時間 (1 ステップ) で他のプロセッサに転送できる理想的な場合を考える．現実の並列コンピュータは，プロセッサ間距離やネットワーク輻輳によって転送時間 (ステップ数) が大きくなるので，実際にはここで導出した値に等しいか大きくなる．すなわち，本導出での最小値が疎結合型並列コンピュータにおける通信ステップ数の下限値になる．

通信ステップ数の下限値

通信ステップ数に関しては (4.23) 式からわかるように K_e が大きいほど小さくなる．前述のように $K_e \leq K_c$ であるため $K_c = K_e$ が通信ステップ数が最小となる必要条件である．この条件を (4.23) 式に適用すると下式となる．

$$C = K_c + M\left(\frac{N}{K_c} - 2\right) \tag{4.26}$$

これを K_c について微分し 0 となる点を求めると，

$$\frac{dC}{dK_c} = 1 - M\frac{N}{K_c^2} = 0, \tag{4.27}$$

$$K_c = \sqrt{MN} \tag{4.28}$$

となる．これを満たす時に通信ステップ数は K_c の変化に対して最小値をとる．この通信ステップ数の最小値 C' は，(4.28) 式を (4.26) 式に代入して，

$$C' = 2\sqrt{M}\left(\sqrt{N} - \sqrt{M}\right) \tag{4.29}$$

となる．通信ステップ数の最小値は，行列の行数 N と一つのプロセッサ上に得られる結果要素の数 M とに依存することがわかる．

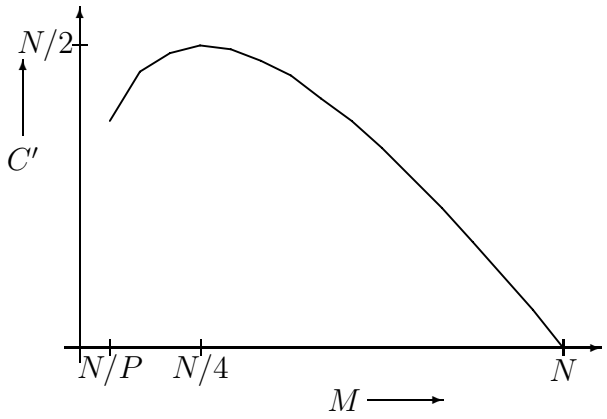


図 4.4: M に対する C' の変化

ここで，M の取り得る範囲について考える．個々の結果要素が少なくともどれかの一つのプロセッサ上で得られている必要がある．結果要素の総数は行列の行数と等しく N であるから，M の最小値は N/P である．(4.28) 式にこれを代入すれば $K_c = N/\sqrt{P}$ となる．また，M の最大値は自明ながら結果要素の総数に等しく N となる．

M を変化させた場合の C' の変化を図 4.4 に示す．図から $M > N/4$ の領域では M の増大とともに通信ステップ数が減少し，ついには 0 なることがわかる．しかしながら，$M > N/4$ という条件は，1 プロセッサに全計算の 1/4 以上が割り付けられることを意味する．このような領域では，通信ステップ数は少ないものの，速度向上度が 4 倍より大きくなることはあり得ない．更に，通信ステップ数を減らすほど演算ステップ数は増加し，通信ステップ数が 0 の場合は演算ステップ数は逐次処理と同一となってしまう．並列処理は高速化を目的としている以上，このような領域には意味がないといえる．以上の議論より，通信ステップ・演算ステップの最小化は $M \leq N/4$ の領域のみが対象となる．

$M \leq N/4$ の領域で C' が最小となるのは，グラフから M が最小値 N/P を取る場合であることがわかる．この時の C' は，

$$C' = \frac{2N}{\sqrt{P}}\left(1 - \frac{1}{\sqrt{P}}\right) \tag{4.30}$$

となる．これが通信ステップ数の下限値である．

演算ステップ数の下限値

行列演算の総計算ステップ数は乗算が N^2，加算が $N(N-1)$ であるから，これらが重複なく，かつ，均等にプロセッサに分散される場合に，プロセッサあたりの演算ステップが下限値を取る．この値は，加算が $N(N-1)/P$，乗算が N^2/P である．

加算及び乗算の演算ステップ数が下限値を取るためには割り付けパラメータがどのような値を取れば良いかを調べてみる．(4.25) 式からわかるように，M が小さいほど加算ステップ数が小さくなるが，前述したとおり M の最小値は N/P である．この値を代入して式を書き直せば，

$$A_s = K_r(K_e - 1) + \frac{N}{P}\left(\frac{N}{K_e} - 1\right) \tag{4.31}$$

となる．上述のように，加算ステップ数が下限値は $N(N-1)/P$ であるから，これを達成するための条件は以下のようになる．

$$K_r(K_e - 1) + \frac{N}{P}\left(\frac{N}{K_e} - 1\right) = \frac{N(N-1)}{P} \tag{4.32}$$

これを整理すると，

$$K_r(K_e - 1) + \frac{N}{P}\left\{\left(\frac{N}{K_e} - 1\right) - (N-1)\right\} = 0 \tag{4.33}$$

$$K_r(K_e - 1) - \frac{N^2}{P}\left(1 - \frac{1}{K_e}\right) = 0 \tag{4.34}$$

$$K_r = \frac{1 - \frac{1}{K_e}}{K_e - 1} \cdot \frac{N^2}{P} = \frac{1}{K_e} \cdot \frac{N^2}{P} \tag{4.35}$$

$$K_e K_r = \frac{N^2}{P} \tag{4.36}$$

となる．この式を満たす K_e と K_r は，計算がもれなく行われるための条件 $K_e K_r P \geq N^2$ も満たしている．また，この場合，$A_m = K = K_e K_r = N^2/P$ なので，乗算ステップ数も最小となる．すなわち，(4.36) 式は計算が完全でかつ演算ステップ数が最小であるための条件式となっている．

通信・演算ステップ数の下限を与える割り付け条件

上記の通信ステップ数と演算ステップ数に関する議論に基づけば，それぞれのステップを両方とも最小とする M, K_c, K_e, K_r の値がそれぞれ次のように求まる．

$$M = \frac{N}{P} \tag{4.37}$$

$$K_c = K_e = K_r = \frac{N}{\sqrt{P}} \tag{4.38}$$

そして，通信ステップ数の下限値 C_{low} は，

$$C_{\mathrm{low}} = 2\left(\frac{N}{\sqrt{P}} - \frac{N}{P}\right) \tag{4.39}$$

となる．どのような並列化方式を用いても，疎結合型並列プロセッサでは (4.39) 式より少ない通信ステップ数で連続行列演算を計算することはできない．また，(4.38) 式が示すように行列を $N/\sqrt{P}$ 行 $\times$ $N/\sqrt{P}$ 列の部分行列に分割することで，通信ステップ数，演算ステップ数ともに下限値をとるような並列化方式を構築できる．

例として $N = 9$ の行列を $P = 9$ 個のプロセッサで並列化する場合に，通信及び演算ステップ数の最小化する割り付け法を図 4.5 に示す．各プロセッサで得る結果要素の数は (4.37) 式より $M = 1$ とすれば良い．また，各プロセッサに割り付ける行列要素に関しては，(4.38) 式より $K_c = K_e = K_r = 3$ となる．このパラメータを持つ最も単純な割り付けが，図に示したような 3×3 の部分行列を一つずつプロセッサに与えた場合である．

4.4 BP 並列化における通信ステップ数の下限値

これまで連続行列演算の並列化について論じてきた．BP の計算は，連続行列演算ととらえることができると述べたが，当然ながらそのアルゴリズムに固有の特性を持っている．特に BP では，前向き伝播を計算した後，その結果と教師信号との差分に基づいて後向き伝播を行うが，この両方向の計算ともに結合重みが使われる．また，結合重みは後向き伝播によって変化し，それが次のサイクルで用いられる．このため，前向き伝播，後向き伝播をそれぞれ独立した連続行列演算としてとらえるのではなく，BP のサイクル全体を通した演算量や通信オーバヘッドを考える必要がある．

ここでは，BP を連続行列演算として見た場合の固有の特性について考察を深め，BP に限定した場合の通信ステップ数の下限値を導出する．

4.4.1 BP における連続行列演算の特性

BP での連続行列演算は次のような特性を持つ．

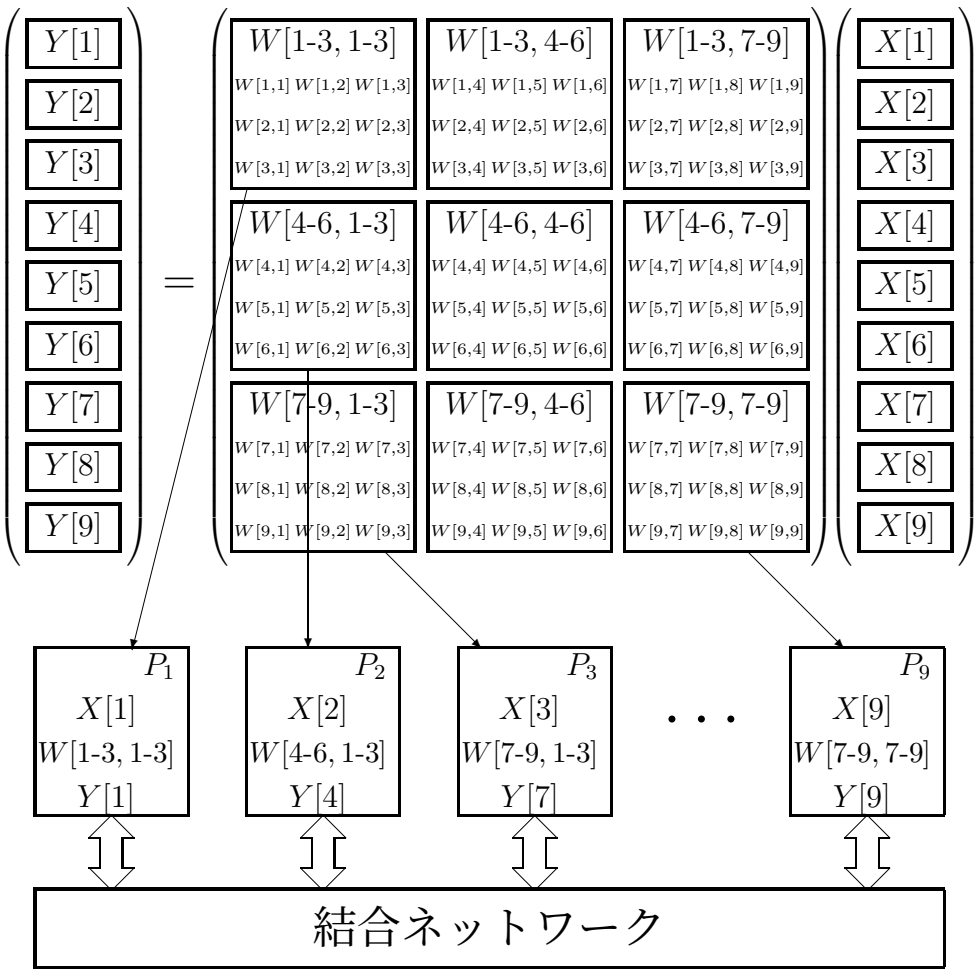


図 4.5: 通信・演算ステップの下限値を与える行列・ベクトルの割り付け法

重み行列要素の双方向伝播での利用

結合重みは，前向き伝播においては出力を得るために使われ，後向き伝播では更新値を得るために使われる．重み行列として見れば，前向き伝播は入力ベクトルとの積がとられ，後向き伝搬では転置されて誤差ベクトルとの積がとられることになる．

重み行列要素の更新

結合重みは，後向き伝播によって，その値が更新される．この更新は前向き伝播・後向き伝播のサイクルごとに繰り返される．したがって，毎回の例示に対する処理で用いられる結合重みは常に異なっていることになる．

結合重みがシステムの動作を通して変化しないのであれば，前向きと後向きの両伝播を独立と考え，実行に先だって，それぞれ都合の良いように重み行列を割り付ければ良い．それによって，一つの重み行列要素が前向き伝播と後向き伝播で異なったプロセッサに割り付けられたとしても，それはメモリを少し余計に消費するだけでしかない．しかし，BP では結合重みは前向き伝播・後向き伝播の毎サイクルごとに更新される．このため，後向き伝播によって更新された結合重みを次のサイクルの前向き伝播で用いなければならない．一つの重み行列要素が前向き伝播と後向き伝播で異なったプロセッサに割り付けられていると，後向き伝播によって更新された要素を他のプロセッサに転送する必要がある．すなわち，後向き伝播から次のサイクルの前向き伝播へと処理の流れが移行する際に，すべての行列要素があるプロセッサから他のプロセッサに転送されることになる．このような転送が必要なため，前向き伝播と後向き伝播を独立して考えた場合に通信ステップを抑えることができる方式でも，BP 全体を通すと通信ステップ数が大きいものとなる場合がある．

以上のことから，BP の並列化では，個々の伝播処理での通信ステップ数・演算ステップ数だけではなく，前向き伝播・後向き伝播の処理の移行に伴う通信ステップ数・演算ステップ数も考慮する必要がある．並列化方式は，これらの総合的な通信ステップ数・演算ステップ数を小さく抑えるように考案されなければならない．一つの行列要素に対し，前向き伝播での利用，後向き伝播での利用，そして値の更新のすべてを同一プロセッサで処理するような並列化方式が，より BP に適した方式となると予測される．

4.4.2 BP における通信ステップ数の下限値

前述の通り，$W_l[j,i]$，$X_l[i]$，$\delta_{l+1}[j]$ の三者を同一のプロセッサに割り付ければ，前向き伝播と後向き伝播の切替え時には結果や重みの転送は不要となる．この場合の BP での通信ステップ数は，前向き伝播，後向き伝播のそれぞれの連続行列演算に必要な通信ステップ数の和で済む．

このような割り付けによる NN の並列計算の様子を図 4.6 に示す．説明を簡単にするために，図では重み行列のサイズが 2×2，プロセッサ数が 4 という非常に小さい規模を例としている．図において，長方形で示したプロセッサの中に，そのプロセッサに割り付けられた行列・ベクトルの要素や，計算で得られる要素を示している．$p = 2(i-1)+j$ であるプロセッサ P_p(ただし $1 \leq i \leq 2, 1 \leq j \leq 2$) に対して，行列・ベクトル要素をそれぞれ表 4.1 に示すように割り付けている．

表 4.1: プロセッサへの要素の割り付け

割り付け先プロセッサ	P_p ただし $p = (i-1) \times 2 + j$
入力層出力	$X_1[i]$
入力層-隠れ層間結合重み	$W_1[j,i]$
隠れ層での積和	$Y_2[j]$
隠れ層出力	$X_2[j]$
隠れ層-出力層間結合重み	$W_2[i,j]$
出力層での積和	$Y_{out}[i]$
出力	$X_{out}[i]$
出力層誤差	$\delta_{out}[i]$
隠れ層誤差	$\delta_2[j]$

図 4.6(a) は前向き伝播において隠れ層の出力を得る処理である．隠れ層での積和の一つの要素 $Y_2[1]$ を例にとると，その部分積和が P_1 と P_3 において計算される．そして，それぞれが相互に転送され集計されて，これら二つのプロセッサ上で要素の値が得られる．積和が求まれば，それに基づいて転送なしに対応する隠れ層出力要素を得ることができる．

同様に図 4.6(b) では出力値を計算している．特徴的なのは，重み行列 $\mathbf{W}_2$ のプロセッサへの割り付けが，直下の層の重み行列 $\mathbf{W}_1$ の割り付けを転置した形となっている点である．このような割り付けにより，(a) で各プロセッサ上に得られた隠れ層出力を，転送することなくそのまま利用して出力層での部分積和を計算する

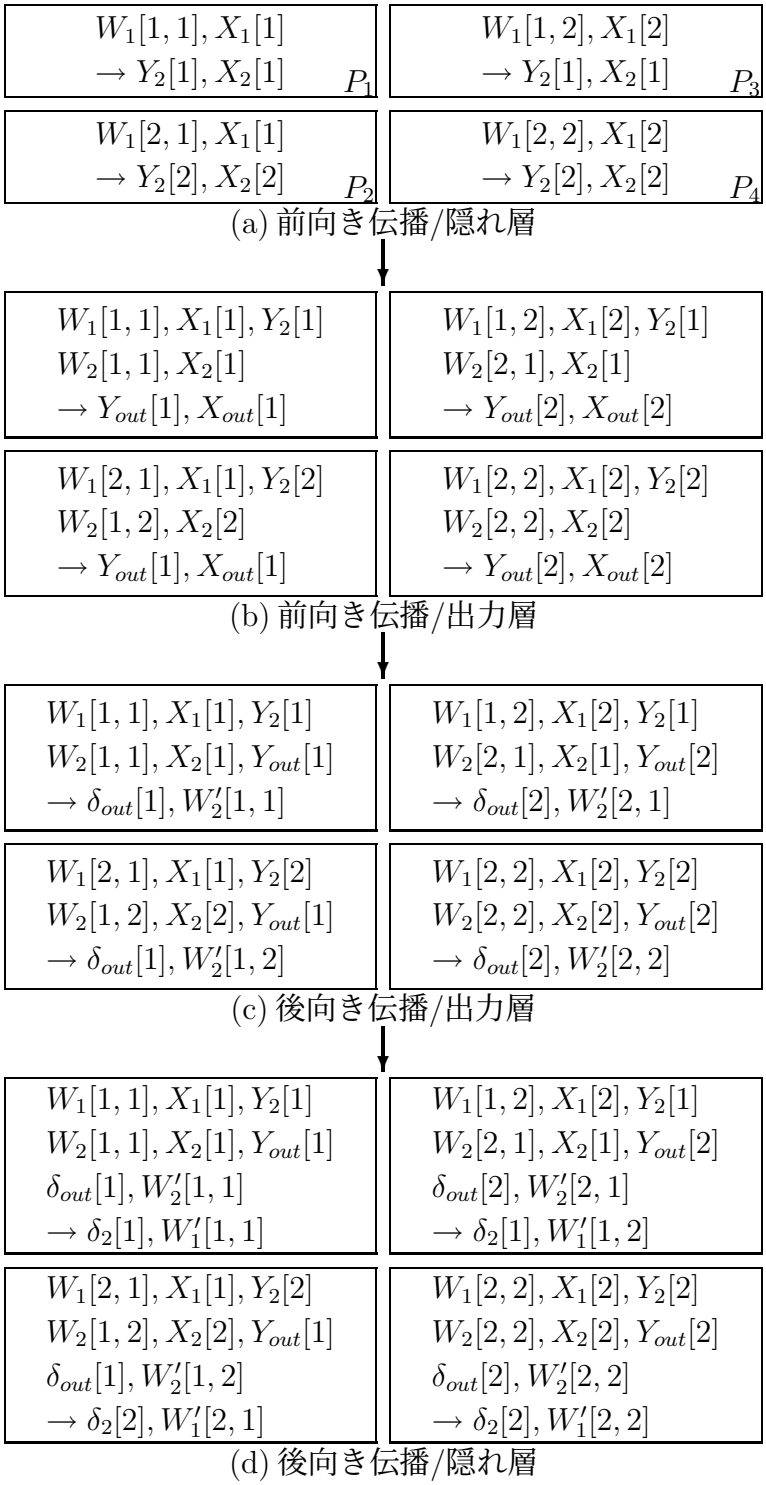


図 4.6: BP 並列演算の例

ことができる．部分積和も (a) を転置した形となり，要素 $Y_{out}[1]$ に対するものがプロセッサ P_1, P_2 上で得られる．これを相互に転送・集計して，両プロセッサ上で出力値 $X_{out}[1]$ が得られることになる．

図 4.6(c)，(d) は後向き伝播の処理であり，前向き伝播と同様に，プロセッサ上で得られた結果を利用して，重み更新の計算ができることを示している．例えば，出力層の重み行列要素 $W_2[1,1]$ を更新するには $\delta_{out}[1]$ と $X_2[1]$ が必要であり，$\delta_{out}[1]$ を得るためには，教師信号 $d[1]$，出力要素 $X_{out}[1]$，積和値 $Y_{out}[1]$ が必要である．(c) からわかるように，これらはすべてプロセッサ P_1 上に揃っている．このためデータ転送の必要なしに重みを更新することができる．同様に，隠れ層の重み行列要素 $W_1[1,1]$ の更新では $\delta_2[1]$ と $X_1[1]$ が必要となり，$\delta_2[1]$ は，$\mathbf{W}_2^T \cdot \boldsymbol{\delta}_{out}$ の 1 行目の値 $\theta_2[1]$ と積和値 $Y_2[1]$ とが必要となる．(d) が示すようにこれらもプロセッサ P_1 上で揃うので，行列演算で必要とされる以外のデータ転送を生じさせずに重みの更新ができる．

以上のように，各層の行列・ベクトル要素の割り付けの工夫により，層の間でのデータ転送や前向き・後向きの移行に伴うデータ転送を伴わずに，BP を並列計算できる方法が存在することがわかった．この場合の通信ステップは，前向き伝播と後向き伝播それぞれの行列演算に必要とされる通信ステップの和になる．後向き伝播では重み行列の転置とベクトルとの積が計算されるので，この時の通信ステップ数は，前向き伝播時の通信ステップ数の式のなかの行と列を入れ換えたものになり，(4.26) 式の場合と同様に計算できる．したがって，前向き伝播・後向き伝播 1 サイクル 1 層分の計算に必要な転送ステップ数 C_{BP} は以下のようになる．

$$C_{BP} = \underbrace{K_c + M\left(\frac{N}{K_c} - 2\right)}_{\text{前向き伝播}} + \underbrace{K_r + M\left(\frac{N}{K_r} - 2\right)}_{\text{後向き伝播}} \tag{4.40}$$

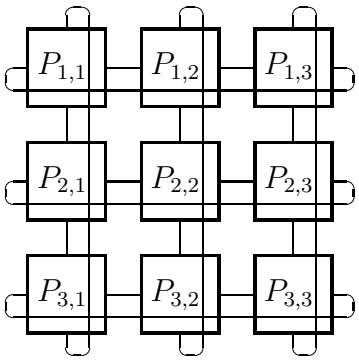
ここで，前章と同じ議論により，それぞれの項を最適化すると，その条件は，

$$K_c = K_r = N/\sqrt{P} \tag{4.41}$$

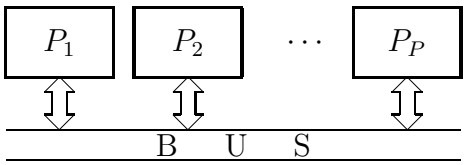
となり，その時の通信ステップ数 $C_{BP\text{low}}$ は，以下のようになる．

$$C_{BP\text{low}} = 4\left(\frac{N}{\sqrt{P}} - \frac{N}{P}\right) \tag{4.42}$$

(4.39) 式は 1 回の行列演算での下限値を表しているのに対し (4.42) 式は前向き伝播と後向き伝播の計 2 回の行列演算における下限値である．$C_{BP\text{low}}$ が C_{low} の 2 倍



(a) 二次元アレイプロセッサ



(b) バス結合型並列コンピュータ

図 4.7: 並列アーキテクチャ

で済んでいるということは，前向きと後向きの両伝播を独立として見ても，それぞれが最適な行列演算並列化になっているような BP 並列化方式の存在を示唆している．

ただし，ここで述べた例は，同一のプロセッサ上に必要な行列・ベクトル要素を配置するような BP 並列化が可能なことを示しているだけで，必ずしも最適な並列化とはいえない．実際，演算ステップ数が 4.3.3 節で導出した下限値を達成していない．これは，非常に規模の小さな問題であるため，同節での図 4.4 で説明した前提が成立しないためである．そこで次節以降では，2 次元アレイ型並列コンピュータとバス結合型並列コンピュータとを対象に，これまでの考察に基づいて行列・ベクトル要素の割り付け及び各プロセッサでの処理を行う，より最適に近い並列化方式を提案する．

4.5 2 次元アレイ型並列コンピュータ向けの並列化方式

本節では，図 4.7(a) に示すような 2 次元アレイ (トーラス) 型並列コンピュータをとりあげ，これに対する並列化方式を提案する．また，本方式において通信ス

テップ数が最小となる条件を導出し，最小通信ステップ数が，4.4.2 節で述べた通信ステップ数の下限値と等しいことを示す．なお，ここで述べる方式は文献 [99] において筆者らが発表したものであるが， Tanno らもほぼ同時期に本質的に同一の方式を提案している [91].

4.5.1 並列化方式

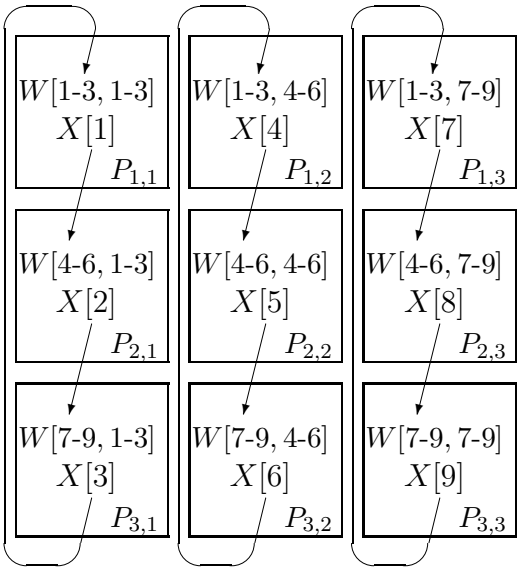
本並列化方式の行列・ベクトル要素の割り付けと各プロセッサの動作を以下に述べる．ここでは，BP の前向き伝播 1 層分についてのみ説明する．4.4.2 節での考察に基づけば，後向き伝播においても，また，すべての層においても，同様の動作を行えば良いこと，そして層間や伝播方向の移行時にデータ転送が不要なことがわかる．

行列・ベクトル要素の割り付け

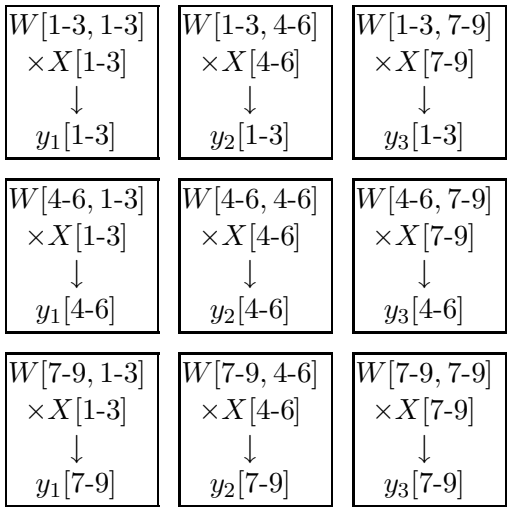
重み行列のサイズが $N \times N$，プロセッサ数が P のとき，この行列を N/q 行 $\times$ N/p 列の部分行列に分割する (ただし $q \cdot p = P$)．左から i 番目，上から j 番目の部分行列を $\mathcal{W}_{j,i}$ と表すことにする．別の表現をすれば，$j_1 = (j-1)N/p + 1$, $j_2 = jN/p$，$i_1 = (i-1)N/q + 1$，$i_2 = iN/q$ として，$\mathcal{W}_{j,i}$ は $W[j_1\text{-}j_2, i_1\text{-}i_2]$($j_1$ 行目から j_2 行目，i_1 列目から i_2 列目の要素を含む部分行列) とも書ける．分割した部分行列を q 行 p 列の 2 次元アレイ上に直接的に割り付ける．すなわち，部分行列 $\mathcal{W}_{j,i}$ を $P_{j,i}$ に割り付ける．

入力ベクトルは N/P 個の要素を持つ部分ベクトルに分割し，プロセッサ $P_{j,i}$ には $\{q(i-1)+j\}$ 番目の部分ベクトルを割り付ける．また，結果ベクトルは N/P 個の要素を持つ部分ベクトルに分割し，プロセッサ $P_{j,i}$ において $\{p(j-1)+i\}$ 番目の部分ベクトルを集計する．

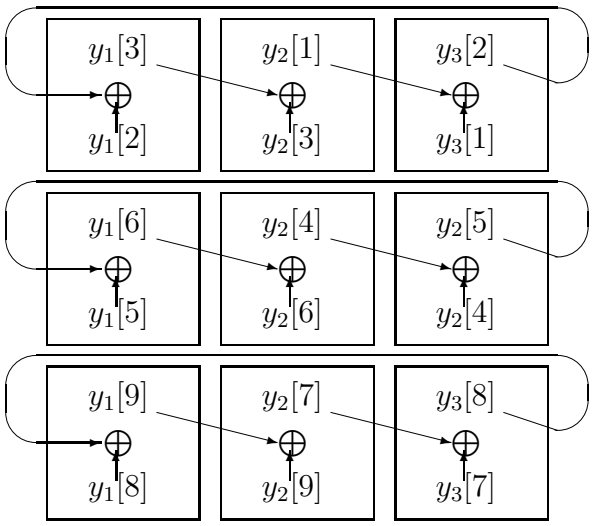
例として $N = P = 9$ の場合の割り付けと動作を図 4.8 に示す．$q = p = 3$ であり，$N/q = 3$，$N/p = 3$ であるから部分行列のサイズは 3×3 となる．$N/P = 1$ となるため，入力の部分ベクトルと結果の部分ベクトルは，それぞれ 1 個の要素を持つ．入力部分ベクトルは左上端のプロセッサより下に向かって順番に割り付けられる．ただし下端の次は右隣の上端になる．結果部分ベクトルは，左上端より右に向かって順番に割り付けられる．ただし右端の次は一段下の左端となる．



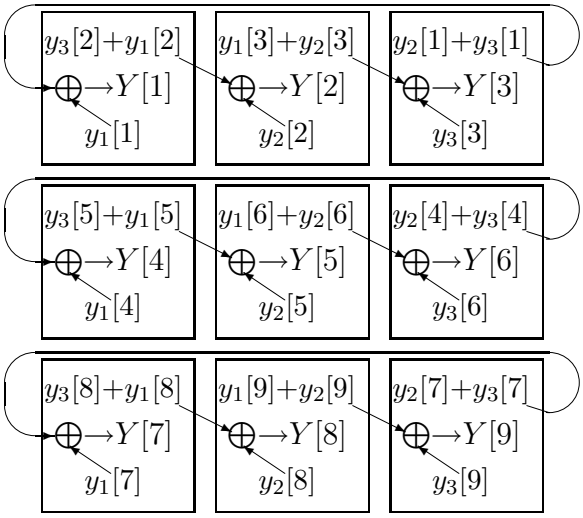
(a) 入力ベクトル要素の配布



(b) 部分積和の計算



(c) 結果ベクトルの集計 (ステップ 1)



(d) 結果ベクトルの集計 (ステップ 2)

図 4.8: 二次元アレイ向けの最適並列化

動作

配布フェーズでは，図 4.8(a) に示すように全プロセッサが下方のプロセッサに自プロセッサの持つ入力ベクトル要素を転送する．一つのプロセッサが発信した要素が同一列の全プロセッサに行き渡るまで転送を繰り返す．この例では，2 回の転送で，1〜3 列目を含む部分行列 W[*,1-3] を保持する全プロセッサ上で $X[1]$〜$X[3]$ が揃うことになる．

配布が終了したら，各プロセッサは割り付けられた部分行列に対する部分積和を計算する．$Y[j]$ に対する部分積和 $y_i[j]$ は各プロセッサ上に図 4.8(b) に示すように得られる．ここで添字 i はプロセッサの列番号である．

集計フェーズでは，図 4.8(c) に示したように，部分積和を右隣のプロセッサに転送し，転送されて来た部分積和を自プロセッサの保持する行の同じ部分積和に加算する．そして図 4.8(d) に示すように，加算結果を右隣プロセッサに転送する．

以上の繰り返しにより，最終的に各プロセッサ上に結果の部分ベクトルを得る．この動作はすべてのプロセッサで並列に行われる．図の例では，例えば $Y[1]$ に関しては，まず，$P_{1,2}$ が自プロセッサ上の部分積和 $y_2[1]$ を右隣のプロセッサ $P_{1,3}$ に転送し，$P_{1,3}$ は自プロセッサ上の部分積和 $y_3[1]$ と加算する．次のステップでは，加算された値を $P_{1,1}$ に転送し，$P_{1,1}$ では自プロセッサ上の部分積和 $y_1[1]$ との加算を行い $Y[1]$ を得る．同様に $Y[2]$ に関しては，$P_{1,3}$ がまず部分積和 $y_3[2]$ を転送し，$P_{1,1}$ 上の部分積和 $y_1[2]$ と加算された後，$P_{1,2}$ で最終的な $Y[2]$ を得る．$Y[3]$ から $Y[9]$ に対しても同様の動作で集計が行われる．このような動作により，例では 2 回の転送ですべての集計が完了する．なお，後向き伝播時には，行と列が入れ替わった動作となる．

4.5.2 通信ステップ数の最小化

4.4.2 節と同様の手順で通信ステップ数を定式化し，最適条件を導出する．

配布フェーズでは，同じ列のプロセッサに $N \cdot q/P$ の要素が転送されなければならないが，プロセッサは N/P の要素を最初から保持しているので，通信ステップ数は $C_d = N \cdot q/P - N/P = (N \cdot q - N)/P$ となる．集計フェーズでは各プロセッサは N/P 個の結果ベクトル要素を計算する．また，結果ベクトルの 1 要素につき p 個の部分積和が必要であるが 1 個は自プロセッサが保持しているので，プロセッサあたり $(p-1)N/P$ の転送が必要となる．これを全プロセッサが並列に

行うので，通信ステップ数は$C_a = (N \cdot p - N)/P$ となる．したがって，前向き伝播１層分の通信ステップ数 C_f は，$P = p \cdot q$ の関係も用いて，以下で表される．

$$C_f = C_d + C_a = N/p + N/q - 2N/P. \tag{4.43}$$

後向き伝播では，上記定式化の行と列が入れ替わるだけであり，式の形としては同一となる．したがって，BP1 サイクル 1 層あたりの通信ステップ数 C_{BP} は，

$$C_{BP} = 2\left(\frac{N}{p} + \frac{N}{q} - 2\frac{N}{P}\right) \tag{4.44}$$

となり，$q = P/p$ の関係を用いれば，

$$C_{BP} = 2\left\{\frac{N}{p} + \frac{N}{P}\left(N\frac{p}{N} - 2\right)\right\} \tag{4.45}$$

となる．N/p を K_c に，N/P を M にそれぞれ対応させれば，(4.26) 式と全く同じ議論が成り立つので，転送ステップを最小にする行列の分割は，

$$\frac{N}{p} = \frac{N}{q} = \frac{N}{\sqrt{P}} \tag{4.46}$$

であり，その際の通信ステップ数は，

$$C_{BPopt} = 4\left(\frac{N}{\sqrt{P}} - \frac{N}{P}\right) \tag{4.47}$$

となる．これは，(4.42) 式が示す通信ステップ数の下限値に等しい．また，この行列分割は (4.36) 式を満たしているので演算ステップ数も下限値に等しくなる．

以上述べたように，本方式で最適な行列の割り付け・動作を行った場合の通信ステップ数，演算ステップ数は，ともに 4.4.2 節で導出した下限値に等しいことがわかった．すなわち，本方式は，BP の最適な並列化方式であると言える．また，このことから，BP を最小の通信量・計算量で並列化するためのアーキテクチャ(トポロジ) は 2 次元アレイで十分であるともいうことができる．

4.5.3 従来方式との比較

従来提案されていた Suzuki らの方式 [87] は，トーラスから両端のプロセッサ間に結合を取り去ったメッシュ結合を対象としている．この方式では，入力ベクトルを 2 次元アレイの最上行より順次下方に転送して部分積和を計算し，次に部分

積和を横方向に転送して最右列のプロセッサで集計し結果を得る (次層では最右列より入力ベクトルを転送し最下行のプロセッサで結果を得る).

プロセッサがパイプラインとして動作するため，本方式と同様に通信と演算は並列に行われる．しかし，入力ベクトルや結果ベクトルの要素が最上行や最右列に偏って配置されることになるため，これに起因して並列に転送できない余分な通信ステップを生ずる．すなわち，最初の入力ベクトル要素の転送を開始して最下行に到達するまでのステップ，及び，最左列のプロセッサが部分積和の転送を開始してそれが最右列に到達するまでのステップが，本方式に比べて余分に必要である．

演算についても同様で，下列のプロセッサは最初の入力要素が到着するまで計算を開始できず演算時間が長くなる．この余分な通信ステップは，行列のサイズがプロセッサ数に比べて大きくなるに従い相対的に小さくなるが，文献で用いられた例 (行列サイズ 512 行 $\times$ 256 列，プロセッサ数 64 $\times$ 32) でも 10% 以上の余分な通信ステップを生ずる．これに対し，本方式の場合は，各ベクトルの配置をプロセッサにわたって均等としているので，通信・計算の負荷が常に均等となり，通信量，演算量ともに下限値を達成できるのである．

4.6 バス結合型並列コンピュータ向けの並列化方式

本節では，バス結合型並列コンピュータを対象とした並列化方式を提案する．また，前節と同様に通信ステップ数が最小となる条件を導出し，従来のバス結合型並列コンピュータ向け並列化方式と比較する．

4.6.1 並列化方式

2 次元アレイに対すると同じく前向き伝播 1 層分の動作を説明するが，前節同様，4.4.2 節の考察に基づいて後向き伝播時や他の層での割り付けや動作も容易に構築できる．また，層間や伝播方向の切替えにデータ転送が不要なのは，これも前節同様である．

割り付け

行列を N/q 行 $\times N/p$ 列の部分行列に分割し，図 4.9 に示したように，部分行列 $\mathcal{W}_{j,i}$ をプロセッサ P_x に割り付ける．ただし，$x = p(j-1)+i$ とする．入力ベクトルは P 分割し，P_y に対して以下の式で与えられる k 番目の部分ベクトルを割り付ける．

$$k = \begin{cases} \{q(y-1) \bmod (P-1)\} + 1 & (y \neq P) \\ P & (y = P) \end{cases} \tag{4.48}$$

ただし，式中の演算子 mod は左項を右項で割った余りを意味する．また，結果ベクトルも P 分割し，プロセッサ P_z 上で z 番目の部分ベクトルを集計する．

動作

配布フェーズでは，図 4.9(a) に示したように，各プロセッサが順番にそのプロセッサの保持する入力部分ベクトル要素を放送機能を利用して転送する．必要な入力部分ベクトルが揃ったところで，各プロセッサは割り付けられた部分行列に対する部分積和を計算する．

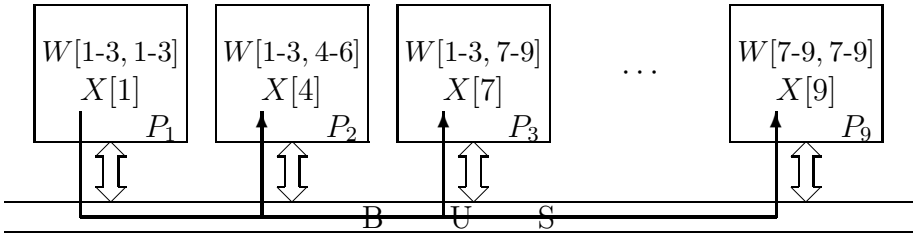
集計フェーズでは必要な部分積和を 1 対 1 転送によって集め総和を計算する．$Y[1]$ を例にとると，まず，プロセッサ P_1 は図 4.9(c) に示すように P_2 から部分積和を転送してもらって加算し，次のステップで P_3 から部分積和を転送してもらい，最終的な結果を得る．バスでは 1 ステップで 1 組の転送しかできないので，集計は逐次的に処理される．

4.6.2 通信ステップ数の最小化

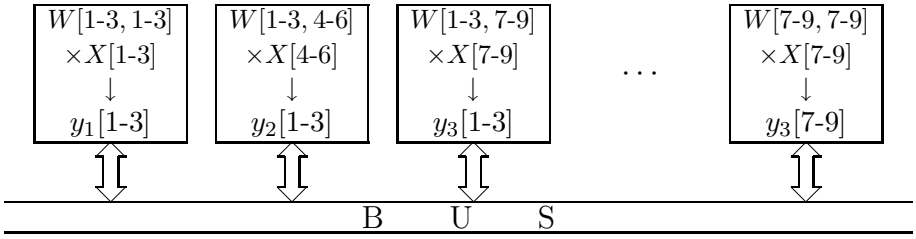
上述の割り付けと動作に基づく場合の通信ステップ数を定式化し，これを最小とする条件を導く．配布フェーズでは，入力ベクトルのすべての要素が 1 回ずつ転送される．したがって，配布の通信ステップ数は $C_d = N$ となる．ただし，同一の列を含む部分行列が一つしかない場合，すなわち列分割 $(p = P, q = 1)$ の場合には，部分積和の計算に必要な入力ベクトル要素は最初からすべて揃っているので転送の必要はない．したがって，配布のための通信ステップ数 C_d は，

$$C_d = \begin{cases} 0 & (q = 1) \\ M & (q \geq 2). \end{cases} \tag{4.49}$$

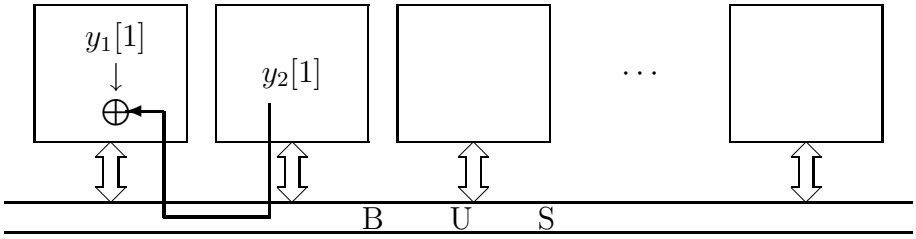
となる．



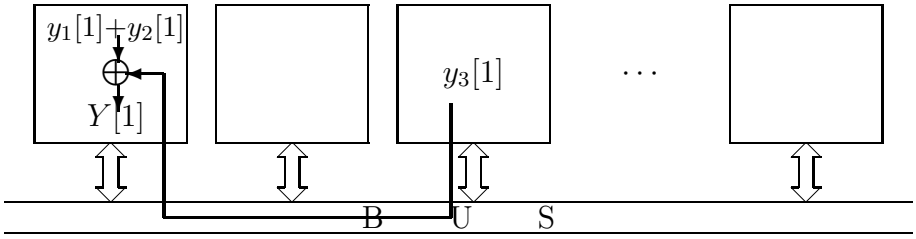
(a) 入力ベクトル要素の配布



(b) 部分積和の計算



(c) 結果ベクトルの集計 (ステップ 1)



(d) 結果ベクトルの集計 (ステップ 2)

図 4.9: バス結合向けの最適並列化

集計フェーズでは，一つの結果ベクトル要素の計算に $p-1$ 個の部分積和を他のプロセッサから転送してもらわねばならない．結果ベクトル要素は N 個あるので，転送すべきデータ数は $N(p-1)$ である．これらの転送はすべて逐次的に行われるので，

$$C_a = N(p-1) \tag{4.50}$$

となる．

以上より，前向き伝播 1 層分の通信ステップ数 C_f は，

$$C_f = C_d + C_a = \begin{cases} N + N(p-1) & (q \geq 2) \\ N(P-1) & (q = 1) \end{cases} \tag{4.51}$$

となる．

後向き伝播時の通信ステップ C_b は，上式の行と列を入れ替えた式となるので，BP 1 サイクル 1 層あたりの通信ステップ数 C_{BP} は以下のようになる．

$$C_{BP} = C_f + C_b = \begin{cases} Np + Nq & (q \geq 2, p \geq 2) \\ NP & (\text{その他}) \end{cases} . \tag{4.52}$$

上式の前者の極小点は，

$$N/q = N/p = N/\sqrt{P} \tag{4.53}$$

の場合で，この際の通信ステップ数 C' は

$$C' = 2N\sqrt{P} \tag{4.54}$$

となる．結局，BP の 1 サイクル 1 層分の通信ステップ最小値 C_{BPopt} は，

$$C_{BPopt} = \begin{cases} 2N\sqrt{P} & (P/4 \geq 1) \\ NP & (P/4 < 1) \end{cases} \tag{4.55}$$

となる．プロセッサ数が 4 よりも大きければ，(4.55) 式前者が通信ステップの最小値となる．

なお，演算ステップ数については，本分割法は (4.36) 式を満たし，4.3.3 節で導出した下限値に等しくなる．

4.6.3 従来方式との比較

従来の BP 並列化の研究では，バス結合型アーキテクチャに対しては重み行列を行分割 ($p = 1$) する並列化方式が提案されてきた [87]．この行分割は単純な連続行

列演算に対しては確かに最適な並列化方式である．単純な連続行列演算の場合の通信ステップ数は (4.51) 式と同一になり，この式は明らかに $p = 1$ で最小値 N をとる．行分割では集計フェーズでの 1 対 1 通信が不要なため，通信ステップ数が最小となるのである．ところが，BP の場合には後向き伝播において転置行列とベクトルとの積の計算がある．行分割は転置行列に対しては列分割と等価なので，後向き伝播時には集計フェーズにおいてすべての部分積和の転送が必要となる．このため，BP の 1 サイクルを通した通信ステップ数はプロセッサ数に比例して増加することになる．

本方式では，通信を前向き伝播と後向き伝播にバランス良く振り分け，BP の 1 サイクルを通しての通信ステップ数を最小化する．そのための条件が (4.53) 式である．これは 2 次元アレイでの最適化条件と同じく行列を正方の部分行列に分割することを示している．(4.55) 式からわかるように本方式では通信ステップ数はプロセッサ数の平方根に比例するので，プロセッサ数が多いほど，本方式は従来方式に比べて有利となる．

4.7 まとめ

NN システムの代表的学習アルゴリズムである BP に対する並列化について論じた．

BP を連続行列演算の特殊な場合としてとらえ，まず，連続行列演算の並列化における通信量と計算量とを定式化して理論的下限値を明らかにした．連続行列演算を並列化すると，その処理は必然的に配布，部分積和，集計の 3 つのフェーズから構成され，配布と集計においてデータ転送が必要となる．また，行列を正方の部分行列に分割した場合に，通信ステップ数の下限値と演算ステップ数の下限値が同時に達成される．

次に，BP 固有の連続行列演算の特徴を明らかにし，これら考察に基づき，BP での通信ステップ数，演算ステップ数の下限値を導いた．BP では，前向き伝播と後向き伝播で重み行列が転置して利用されること，重み行列の値は毎回変化することなどが特徴であり，これによって，単純な連続行列演算に対しより多くの通信が必要な場合がある．しかし，そのような制約にもかかわらず，下限値に関しては連続行列演算と同一であることが示された．また，その下限値を達成するために必要な行列・ベクトル要素のプロセッサ上での配置も明らかとなった．

これらの考察に基づき，2 次元アレイ型並列コンピュータ，バス結合型並列コンピュータでの BP の最適並列化方式を提案した．

2 次元アレイ型並列コンピュータに対する方式は，行列を正方行列に分割してプロセッサに割り付けることで通信ステップ数が最小となる．この通信ステップ数の最小値と演算ステップ数は，上記の下限値を達成しており，本方式が最適な BP 並列化方式であることを証明した．すなわち，BP の並列化においては，2 次元アレイ以上に複雑なアーキテクチャ(トポロジ) は必要ないといえる．

バス結合型並列コンピュータに対しても，単純な連続行列演算の並列化方式を BP に対して適用しても良い性能が得られないことを示し，新たな並列化方式を提案した．従来から提案されていたバス結合型アーキテクチャ向け方式である行分割方式は単純な連続行列演算においては最適であるが，BP では前向き伝播と後向き伝播との移行の際に大きな通信オーバヘッドが生ずる．BP に対しては 2 次元アレイに対すると同様に正方な部分行列へと分割する本方式が優れていることを証明した．通信ステップ数は，従来方式がプロセッサ数に比例したのに対し，提案方式はプロセッサ数の平方根オーダに抑えられることを示した．

第5章　類似概念検索の並列化

人工知能システムは人間の思考を模擬するシステムであるから，人間が話す言語 (自然言語) を理解し，それに基づいて推論を行う機能も重要となる．このための技術は，人間からの自然言語による命令を理解して動作するヒューマノイド・インタフェースに必要なことは当然であるが，それほど統合的なシステムでなくともそれを必要とする分野は多い．

データベース検索やインターネットの WWW 検索では，膨大な数のレコードや WWW ページから，ユーザが必要とするものを的確に見つけ出す必要がある．従来の文字列の一致に基づく検索では，多くの不要なレコードやページがヒットしてしまう．また，ユーザが与える検索条件は，自然言語に基づくものであるが，必ずしも検索に十分な情報を含んでいるとは限らないため，必要な情報がヒットしない場合もある．このため，自然言語で与えられたユーザの指示から，コンピュータがユーザの意図を理解して，検索条件を適切に補完，外挿することが必要となる [68].

推論システムに関しても，従来，知識の表現に使われるシンボルや変数は単なる記号でありその同一性のみしか意味を持たないため，プログラムが非常にわかりにくく柔軟性に欠けるものとなっていた．人間は知識を記述する際にシンボルとして自然言語の単語を利用する．これは，そのシンボルがその背景とする言語知識をも内包し，記述した以外の知識がそれによって外挿されることを暗に期待しているのである．したがって，推論システムの構築をより容易で柔軟とするためには，コンピュータが与えられたシンボルを自然言語として解釈して，類推 [95] や事例に基づいた [28] 外挿や補完を行う発想的推論 (アブダクション) が必要とされる．

このように，自然言語が内包的に持つ知識を用いて補完や外挿を行う機構は，柔軟な人工知能システムの実現には欠かせないものである．このためにまず必要となるのは，自然言語の単語とそれの背景的知識の総体を「概念」としてコンピュータに格納し，概念間の類似性を判別する機能および，類似判別に基づいて類似し

た概念を検出する類似概念検索の機能である．これら機能により，類似した概念により不足知識を補ったり，多くの概念の類似性から知識を外挿することが可能となる．

日常的に用いられる単語を概念としてコンピュータ内に格納する研究は，電子化辞書として多く研究されてきた．その中には，機械翻訳を主目的としており文法知識の重点がおかれている [101] もの，文書理解や推論のために意味知識に重点がおかれたもの [64, 51] がある．このなかで後者は「概念ベース」と呼ばれ，まさに上述したような柔軟でより人間の感覚に合う推論機構への応用に適している [62]．概念ベースは，数万〜数十万語の日常語に対して類似判別を行うことができる．類似性はその語が使われる状況や文脈によって異なるという考えから，これらを観点として表し類似度計算に影響を与えるようになっている点が特徴である．

概念ベースを，推論における補完や外挿に用いる場合，与えられた観点のもとで与えられた概念に最も類似した概念を全概念の中から探し出す処理，すなわち類似概念検索が多くなる．類似概念検索は推論サイクルの中に組込まれるものであるため，これが高速に行われないと推論に著しい速度低下を招くことになる．全概念に対して類似度を計算するという最も単純な方法は，概念数が多いことから全く実用的ではない．また，事前に類似度を計算してテーブルとして準備しておくにも，類似度計算は引数として二つの概念と一つの観点，計 3 つの概念を取るため，テーブルのサイズが現実的な大きさに収まらない．

このようなことから，概念ベースにおける類似概念検索を高速に処理する方法が必要とされている．そこで本章では，これを並列処理により高速化する方式を提案する．まず，類似概念判別の動作を詳細に説明するとともに考察する．つぎに，それに基づき，計算量を削減する手法とバス結合型アーキテクチャ向けの並列化方式を提案する．更に，異なった視点から類似概念判別を定式化し，ネットワーク結合型アーキテクチャとして自然言語処理に適した SNAP(Semantic Network Array Processor)[66] 上での並列化方式を提案する．

5.1 概念の類似判別と検索

上で述べた通り概念の意味的な類似性は，状況や文脈によって変化するため，類似判別システムはこの状況や文脈に応じた判別をできる必要がある．概念ベース・システムは，状況や文脈を「観点」としてとらえ，観点を含む類似判別を行うシ

ステム [51] である．概念ベース・システムでは，さまざまな観点における二つの概念の類似性を判別することができる．大規模な日常語の意味定義データベース (概念ベース) を持ち，概念ベース中の任意の二つの概念の組と観点とに対して類似度を計算する．

類似概念検索は，一つの概念と観点が与えられた際に，与えられた観点のもとで与えられた概念と最も類似する概念を見つけ出すことを指す．発想的推論では，適切なルールが見つからない場合に，類似したルールを代替して利用する．データベースの検索インタフェースでは検索語に完全一致するデータがない場合に，類似する検索語を代替して再度検索を行う．このように，柔軟な人工知能システムを目指すとき，概念ベースが持つ観点に基づく類似判別機能と類似概念検索機能は非常に重要な位置を占める．以下では，観点に基づく類似判別のアルゴリズムと類似概念検索の基本方式について述べる．

5.1.1 観点に基づく類似判別

概念「馬」と類似する概念を考える.「動物」に関する文脈で語られている時，すなわち「動物」を観点とした時は,「自動車」よりも「豚」の方が「馬」により類似ていると思える．一方，観点として「乗物」を考えると「自動車」の方がより類似しているといえる．概念ベース・システムでは，このような観点により変化する類似性を計算により判別する．

観点に基づく類似判別はより形式的には以下のように定義される．

観点に基づく類似判別

> キーとなる概念 W_k，観点となる概念 W_v，そして二つの候補概念 W_a, W_b が与えられた場合に，観点 W_v のもとで，どちらの候補概念がキー概念 W_k により類似しているかを判別する．

この判別ができれば，多数の候補概念がある場合にこれらに対する類似性の半順序関係を決定できる．したがって，最も類似した概念 (最類似概念と呼ぶ) を決定すること，すなわち類似概念検索も可能となる．

与えられた観点のもとでの概念間の類似度を定義できれば，候補概念それぞれに対しこの定義に基づいた類似度を計算し，それらを比較することで類似判別を行うことができる．続く節では，概念を表現して格納する概念ベースの構成と，それに対する類似度計算について述べる．

5.1.2 概念ベース

概念ベース・システムは，日常語に対する概念を知識として格納し概念ベースを持っている．日常語を対象としているため，少なくとも格納される単語数は数万〜数十万語となる．このように大規模となるため，このシステムでは概念ベースを自動構築する．これほど大規模になると，時間と稼働をかけて手作業で構築しても完全なものにするのは困難である．いずれにしろ誤りや欠落を含むのであれば，格納された情報の不正確さを許容する類似判別アルゴリズムを考案することで，概念ベースの構築を容易化しようというアプローチである．

概念ベースに格納される情報は，機械可読な国語辞典の語義文やコーパスより機械的に抽出され，概念は属性名–重みペアの集合によって表現される [50, 90]．国語辞典からの構築では，まず対象とする概念の語義文を形態素解析する．次に，個々の形態素に対し，その語が所属するカテゴリを検索し，そのカテゴリのカウントを 1 増加させる．カテゴリは機械可読な語彙体系辞書 [37] から抽出し，形態素のカテゴリへの所属もそれを利用して判別する．

カテゴリが全部で N 種類あるとし，個々のカテゴリにユニークな番号 $1 \ldots M$ をふる．また，概念 W_i でのカテゴリ m に対するカウント値を $c_i[m]$ とする．概念 W_i は以下のようなベクトルで表現される．

$$W_i = (q_i[1], \ldots, q_i[m], \ldots, q_i[M]) \tag{5.1}$$

ただし，ここで，

$$q_i[m] = \frac{c_i[m]}{\sqrt{\sum_{m=1}^{M} c_i[m]^2}} \tag{5.2}$$

である．これはすなわち，カテゴリが張る直交空間において，各軸の成分がカウント値に比例し長さが 1 であるようなベクトルとして概念を表していることになる．

実際のシステムではカテゴリの種類は約 3000 である．これに対し，各概念においてカウント値が 0 でないカテゴリは数十〜100 程度であるため，(5.1) 式はほとんどの要素が 0 の非常にスパースなベクトルとなる．そこで，重みが 0 でないカテゴリ名 (これをこの概念に対する属性名と呼ぶ) のみを用い，概念 W_i を以下のような属性名–重みペアの集合で表現すれば，よりコンパクトに表すことができる．

$$W_i = \{(p_i[1], q_i[p_i[1]]), \ldots, (p_i[n], q_i[p_i[n]]), \ldots, (p_i[N_i], q_i[p_i[N_i]])\} \tag{5.3}$$

ただし，ここで，$p_i[n]$ は重みが 0 でない n 番目のカテゴリ (すなわち n 番目の属性) の番号である．また，N_i はこの概念 W_i のおいて重みが 0 でないカテゴリの数である．$p_i[n]$ のリストを P_i とも表記することにする．すなわち，

$$P_i = (p_i[1], \ldots, p_i[n], \ldots, p_i[N_i]) \tag{5.4}$$

と書く．

5.1.3 類似度計算アルゴリズム

(5.4) 式で表現された概念ベースに対し，観点 W_v と二つの概念 W_i，W_j が与えられた時に，その類似度を以下の 3 ステップのアルゴリズムによって計算する．

観点による変調

観点を類似度に影響させるため，概念が持つそれぞれの属性を観点が持つ対応する属性により強調する．具体的には観点 W_v と概念 W_i との両者が持つ属性に対し，W_i での重みを C 倍する．ここで，$C(>1)$ は変調度と呼ばれ，観点の類似度への影響を制御するパラメータである．概念 W_i を観点 W_v によって変調したものを $W'_{i(v)}$ と表す．

$W'_{i(v)}$ はベクトル表記では，

$$\begin{gathered} W'_{i(v)} = (q'_{i(v)}[1], \ldots, q'_{i(v)}[m], \ldots, q'_{i(v)}[M]), \\ q'_{i(v)}[m] = \begin{cases} C \cdot q_i[m] & (q_v[m] > 0) \\ q_i[m] & (q_v[m] = 0) \end{cases} \end{gathered} \tag{5.5}$$

と書け，属性名–重みペアによる表記では，

$$\begin{gathered} W'_{i(v)} = \{(p_i[1], q'_{i(v)}[p_i[1]]), \ldots, (p_i[n], q'_{i(v)}[p_i[n]]), \ldots, (p_i[N_i], q'_{i(v)}[p_i[N_i]])\}, \\ q'_{i(v)}[k] = \begin{cases} C \cdot q_i[k] & (k, q_v[k]) \in W_v \\ q_i[k] & \text{それ以外} \end{cases} \end{gathered} \tag{5.6}$$

となる．

重みの正規化

上で述べたように，概念はカテゴリが張る空間での長さ 1 のベクトルと考えることができる．観点により変調すると，観点と概念とが共通して持つカテゴリの重みが C 倍されるため，長さが 1 より大きくなってしまう．これを，

再び単位ベクトルとするために，重みの正規化を行う．正規化された後の概念を $W_{i(v)}$ で表すと，ベクトル表記では以下のようになる．

$$W_{i(v)} = (q_{i(v)}[1], \ldots, q_{i(v)}[m], \ldots, q_{i(v)}[M]),$$
$$q_{i(v)}[m] = \frac{q'_{i(v)}[m]}{\sqrt{\sum_{k=1}^{M} q'_{i(v)}[m]^2}} \tag{5.7}$$

また，属性名–重みペアによる表記は以下のようになる．

$$W_{i(v)} = \{(p_i[1], q_{i(v)}[p_i[1]]), \ldots, (p_i[n], q_{i(v)}[p_i[n]]), \ldots, (p_i[N_i], q_{i(v)}[p_i[N_i]])\},$$
$$q_{i(v)}[k] = \frac{q'_{i(v)}[k]}{\sqrt{\sum_{n=1}^{N} q'_{i(v)}[k]^2}} \tag{5.8}$$

類似度の計算

類似度 $S_v(i,j)$ は二つの概念の上述のように観点で変調し正規化したもの $W_{i(v)}$ and $W_{j(v)}$ の内積から計算される．すなわち，ベクトル表記では，

$$S_v(i,j) = \sum_{m=1}^{M} q_{i(v)}[m] \cdot q_{j(v)}[m] \tag{5.9}$$

であり，属性名–重みペア表記では，

$$S_v(i,j) = \sum_{p \in P_i \cap P_j} q_{i(v)}[p] \cdot q_{j(v)}[p] \tag{5.10}$$

となる．

上述の類似度計算の手順から観点による変調のステップを除いて計算した類似度を，二つの概念の不偏類似度と呼び $S_0(i,j)$ で表す．すなわち，

$$S_0 = \sum_{m=1}^{M} q_i[m] \cdot q_j[m] \tag{5.11}$$

であり，また，

$$S_0(i,j) = \sum_{p \in P_i \cap P_j} q_i[p] \cdot q_j[p] \tag{5.12}$$

とも書ける．

5.1.4 類似概念検索の基本的方式

類似概念検索は，与えられた概念に対し与えられた観点の元で最も類似度の大きい概念を探し出すことである．類似概念検索の最も単純で自明な方法は，システム中に含まれているすべての概念に対して類似度を計算して，これらを比較してやることである．

この方法では，検索時間は概念ベースに含まれる概念数に比例する．1 組の概念に対する類似度計算が 1 ミリ秒かかるとし，概念ベースが 4 万語の概念を持っていれば，1 回の類似概念検索に 40 秒かかることになる．当然ながら，あらゆる概念の組とあらゆる観点に対して事前に類似度を計算してテーブルに格納しておけば，単純にテーブル上の類似度から最大値を見つけ出すだけで類似概念検索が実現できる．しかし，このときのテーブルに含まれる要素は，$40,000^3$，すなわち 64×10^{12} となりサイズ的にも現実的ではないし，テーブルを生成するための計算時間も非現実的である．

5.2 バス結合型並列コンピュータによる類似概念検索の高速化

本節では，類似概念検索の計算量を大幅に削減する手法と，それのバス結合型並列コンピュータによる並列化方式を提案する．

計算量削減手法は，任意の概念の組に対し観点が仮にどのようなものであっても類似度が越えることのない値，すなわち上界，があらかじめ計算できることに着目し，それを利用して探索空間を絞り込むもので，UB-FSR(upper-bound-based fast similitude retrieval) 方式と名付けた．概念ベースには非常に多数の概念が含まれているが，検索語として与えられた概念に対して，ほとんどの概念は非常に小さな類似度しか持たない．これら小さな類似度しか持たない概念をあらかじめ選別できれば，これらに対する類似度計算を省略することができ，それにより探索空間を大幅に縮小できる．

UB-FSR によって大幅に探索空間が削減され，基本方式に比べて数千倍もの速度向上が得られるが，さらにこれを高速化するためにバス結合型並列コンピュータによる並列化方式を提案する．

5.2.1 類似度の上界

類似度は，二つ概念 W_i，W_j を固定しても，観点 W_v を変えることによって変化する．しかしながら，その変化の範囲は無制限ではなく，W_i と W_j とにより何らかの制限を受けるはずである．

そこで，類似度の計算について考察を深め，二つの概念に対しどのような観点が与えられても越えることがない類似度，すなわち類似度の上界を導く．

まず，α を以下のように定義する．

$$\alpha = C^2 - 1, \tag{5.13}$$

また，$R(i,j)$ を以下のように定義する．

$$R_v(i,j) = \frac{\sum\limits_{r \in P_i \cap P_j \cap P_v} q_i[r] \cdot q_j[r]}{\sum\limits_{s \in P_i \cap P_j} q_i[s] \cdot q_j[s]} \tag{5.14}$$

すなわち，$R_v(i,j)$ は，二つの概念 W_i，W_j についての観点 W_v に含まれる属性のみに対する内積を，二つ概念の内積で除した値である．更に，概念 W_i の観点 W_v による変調後のベクトル長を $L_{i(v)}$ と定義する．すなわち，

$$\begin{aligned} L_{i(v)}^2 &= \sum_{t \in P_i} q_i[t]^2 + \sum_{u \in P_i \cap P_v} \left(C^2 - 1\right) q_i[u]^2 \\ &= \sum_{t \in P_i} q_i[t]^2 + \alpha \sum_{u \in P_i \cap P_v} q_i[u]^2 \\ &= \left(1 + \alpha \frac{\sum\limits_{u \in P_i \cap P_v} q_i[u]^2}{\sum\limits_{t \in P_i} q_i[t]^2} \right) \sum_{t \in P_i} q_i[t]^2 \\ &= 1 + \alpha R_v(i,i) \end{aligned} \tag{5.15}$$

である．

これらの定義を用いて，観点 W_v の元での概念 W_i と W_j との類似度の計算式を

書き直すと，

$$\begin{aligned} S_v(i,j) &= \sum_{p\in P_i\cap P_j} q_{i(v)}[p]\cdot q_{j(v)}[p] \\ &= \sum_{p\in P_i\cap P_j} \frac{q_i[p]}{L_{i(v)}}\cdot\frac{q_j[p]}{L_{j(v)}} + \sum_{p\in P_i\cap P_j\cap P_v} \left(C^2-1\right)\frac{q_i[p]}{L_{i(v)}}\cdot\frac{q_j[p]}{L_{j(v)}} \\ &= \frac{1}{L_{i(v)}L_{j(v)}}\cdot\left(\sum_{r\in P_i\cap P_j} q_i[r]\cdot q_j[r] + \alpha\sum_{s\in P_i\cap P_j\cap P_v} q_i[s]\cdot q_j[s]\right) \\ &= \frac{\sum_{r\in P_i\cap P_j} q_i[r]\cdot q_j[r]}{L_{i(v)}L_{j(v)}}\cdot\left(1+\alpha\frac{\sum_{s\in P_i\cap P_j\cap P_v} q_i[s]\cdot q_j[s]}{\sum_{r\in P_i\cap P_j} q_i[r]\cdot q_j[r]}\right) \\ &= S_0(i,j)\frac{1+\alpha R_v(i,j)}{\sqrt{1+\alpha R_v(i,i)}\sqrt{1+\alpha R_v(j,j)}} \end{aligned} \tag{5.16}$$

となる．$S_0(i,j)$ は概念 W_i と W_j と不偏類似度である．

内積の特性から，以下のことがいえる．

$$\sum_{r\in P_i} q_i[r]^2 \sum_{s\in P_j} q_j[s]^2 \geq \left(\sum_{t\in P_i\cap P_j} q_i[t]\cdot q_j[t]\right)^2, \tag{5.17}$$

そして，これより以下が導かれる．

$$Rv(j,j) \geq \frac{R_v(i,j)^2 S_0(i,j)^2}{R_v(i,i)^2}. \tag{5.18}$$

したがって，類似度について以下のように書くことができる．

$$S_v(i,j)^2 \leq \frac{(1+\alpha R_v(i,j))^2 S_0(i,j)^2}{(1+\alpha R_v(i,i))\left(1+\alpha\dfrac{R_v(i,j)^2 S_0(i,j)^2}{R_v(i,i)^2}\right)} \tag{5.19}$$

上式の右辺を $R_v(i,j)$ 及び $R_v(i,i)$ で微分して，極大値をみつければ，それが概念 W_i と W_j の類似度の上界となる．すなわち，概念 W_i と W_j に対して，どのような観点を持って来ても，類似度が越えることのない値である．これは，次式のようになる．

$$\begin{aligned} U(i,j,C) &= S_0(i,j)\frac{1+\alpha}{1+\alpha S_0(i,j)} \\ &= S_0(i,j)\frac{1+(C^2-1)}{1+(C^2-1)S_0(i,j)} \end{aligned} \tag{5.20}$$

また，この式から，

$$S_0(i,j) \leq S_0(i,k) \rightarrow U(i,j,C) \leq U(i,k,C) \tag{5.21}$$

ということが言える．すなわち，上界は不偏類似度に対して単調である．

5.2.2 探索空間削減する類似概念検索アルゴリズム

上記の考察から，観点が異なることによる類似度の変化範囲は，不偏類似度から計算できることがわかる．もちろん，類似度そのものは依然として観点が与えられるまでは計算できない．

どのような観点が与えられても類似度がその値を越えることがないという上界値は，(5.20) 式を見ればわかるように不偏類似度と変調度だけから計算される．このため，概念ベースが生成された際にすべての概念の組に対しあらかじめ計算しておくことが可能である．これを利用して，類似概念検索での探索空間を大幅に削減するアルゴリズムが UB-FSR である．

キーとなる概念 W_i と観点 W_v が与えられ，概念ベース中のすべての概念から類似度 $S_v(i, J)$ を最も大きくする概念 W_J を検索することを考える．本アルゴリズムでは，概念ベース生成時にあらかじめあらゆる概念 W_j に対する不偏類似度 $S_0(i, j)$ を計算し，不偏類似度の降順にソートして並べたリストを作っておく．このリストを以下のように書くことにする．

$$T_i = \Big((j_i[1], S_0(i, j_i[1])), \ldots, (j_(i)[k], S_0(i, j + i[k])), \ldots, (j_i[K], S_0(i, j_i[K])\Big). \tag{5.22}$$

つまり，k 番目に大きな不偏類似度を持つ概念 $W_{j_i[k]}$ であり，その概念と概念 W_i との不偏類似度が $S_0(i, j + i[k])$ である，また，ここで K は概念ベースに含まれる概念数である．

UB-FSR での検索手順は以下の通りである．

1. 最大類似度を保持するための変数 $S_{\max}$ を 0 に，最大類似度を持つ概念を保持するための変数 $W_{\max}$ の空に初期化し，更にインデックス変数 k に 1 をセットする．
2. 概念 $W_{j_i[k]}$ とその概念に対する不偏類似度 $S_0(i, j_i[k])$ をリストから取出す．
3. $S_0(i, j_i[k])$ と変調度 C とから上界値 $U(i, j_i[k], C)$ を計算する．
4. もし，上界値 $U(i, j_i[k], C)$ がその時点での最大類似度 $S_{\max}$ より小さければ，検索を終了し，$W_{\max}$ を最大の類似度を持つ概念として出力する．そうでなければ次へ進む．
5. 観点 W_v のもとでの概念 W_i と $W_{j_i[k]}$ との類似度 S を計算する．
6. もし，S がその時点での最大類似度 $S_{\max}$ より大きければ，S を $S_{\max}$ に上書きし， $W_{j_i[K]}$ を $W_{\max}$ に格納する．

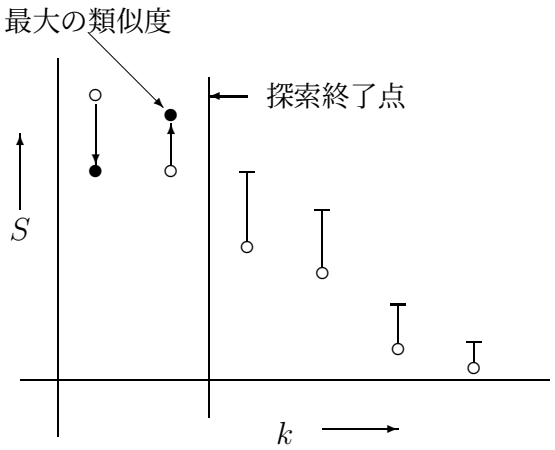
7. k をインクリメントしてステップ 2 へ戻る．

本手順の実行の様子を図 5.1 に示す．

まず，キー概念 W_i に対して不偏類似度が最も大きい概念 $W_{j_i[1]}$ が取出され，観点 W_v のもとで類似度が計算される．図では，観点を与えたことによって類似度が小さくなっている．それでも，まだ候補として 1 個しか概念を取出していないのだから，この時点では，この概念が最大類似度を与えるものとなっている．すなわち，$S_{\max} = S_v(i, j_i[1])$ であり，$W_{\max} = W_{j_i[1]}$ である．

次に，2 番目に不偏類似度が大きい概念 $W_{j_i[2]}$ がとりだされる．この概念に対する上界値 $U(i, j_i[2], C)$ はその時点での $S_{\max}$ より大きいため，観点を与えるとこちらの方が大きな類似度を得られる可能性がある．このため，実際に観点を与えて類似度 $S_v(i, j_i[2])$ を計算する．この計算により $W_{j_i[2]}$ の方がより大きな類似度をもつことがわかった．そこで，$S_{\max} = S_v(i, j_i[2])$，$W_{\max} = W_{j_i[2]}$ とする．

次に 3 番目に不偏類似度が大きい概念 $W_{j_i[3]}$ を取出す．これに対する上界をその時点での最大類似度と比較すると $U(i, j_i[3], C) < S_{\max}$ となっており上界の方が小さい．これはすなわち，概念 W_i と概念 $W_{j_i[3]}$ との類似度はどのような観点が与えられても現在の最大類似度 $S_{\max}$ を越えることがあり得ないことを示している．したがって，これに対する類似度を計算するまでもなく，$W_{j_i[2]}$ の方が大きな類似度を持つという判断を下すことができる．さらに，(5.21) 式が示すように上界は不偏



k : 不偏類似度リストのインデックス
S : 類似度　　● : 観点のもとでの類似度
○ : 不偏類似度　　⊤ : 類似度の上界

図 5.1: 探索空間の絞り込み

類似度に対して単調なため，4 番目以降の概念もその上界が現在の最大類似度を越えることはない．このことから，$W_{\max} = W_{j_i[2]}$ は，厳密に観点 W_v のもとでキー概念 W_i に対し最大類似度を与える概念であると言える．すなわち，探索をここで打ち切っても，厳密な類似概念検索の結果が得られる．

以上のように，基本的なアルゴリズムでは K 個 (現実の概念ベースでは数万〜数十万個) の概念に対する類似度計算が必要なところを，本アルゴリズムを用いた場合，この例では 2 個の概念に対する計算だけで済んでいる．

不偏類似度のリストは，概念ベースに含まれる個々の概念に対して一つずつ存在し，その要素数は K 個である．したがって，K^2 に比例した記憶容量を必要とする．また，このリストを作成するために概念ベース生成時に K^2 回の不偏類似度計算が必要となる．もしも本方式によらず，単純にすべての概念間の類似度をすべての観点に対しあらかじめ計算してリストとして格納しておくとするならば，K^3 回の観点に基づく類似度計算と K^3 に比例する記憶領域が必要となる．概念数 K は非常に大きいため，K^3 に比例する記憶領域の確保と事前計算は非現実的である．

5.2.3 バス結合型並列コンピュータ向け並列化方式

上記のアルゴリズムにより類似概念検索は飛躍的に高速化されるが，並列化することにより更に 1 桁以上の速度向上を目指す．ここでは，3 章で述べたバス結合型並列コンピュータ Presto を用いて，UB-FSR を並列処理する方式を提案する．本方式は，この並列コンピュータのバスが持つ放送機能 (3.3.3 節参照)，同期機能 (3.3.3 節参照) を巧妙に利用することで，通信オーバヘッド，処理量の増加そしてプロセッサ間での処理量ばらつきを小さく抑えている．

概念の割り付け

本方式では，概念をプロセッサに均等に割り付ける．概念ベースに K 個の概念があり，プロセッサ数が P ならば，各プロセッサには K/P 個の概念が割り付けられることになる．また，同時に各プロセッサは，個々の概念についての不偏類似度リストのうち，自プロセッサに割り付けられた概念に対する部分のリストを持つ．すなわち，リストそのものは全プロセッサが全概念分の個数を持つが，それぞれのリストの要素はそのプロセッサに割り付けられた概念のみに限る．したがって

個々のプロセッサが要素数は K/P 個のリストを K 個持つことになる．

図 5.2 に $P = 3$ の場合の割り付け例を示す．図において (a) は概念の割り付け例である．概念 W_i があったとき，$p = ((i-1) \bmod P) + 1$ となるプロセッサ P_p に割り付ける．(b) は不偏類似度リストの割り付けであり，完全な不偏類似度リストのうち自プロセッサが保有する概念に関するもののみを持つ．

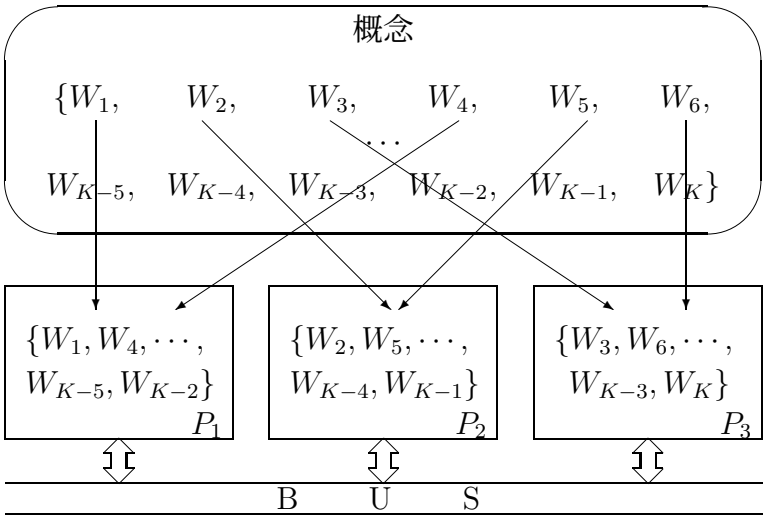
動作

本並列化方式の動作原理は，上記のようしてに割り付けた概念ベースに対し，それぞれのプロセッサが独立に UB-FSR によって最類似概念を検索し，個々のプロセッサでの検索結果を比較して最終的な最類似概念を決定する．これは，3 章で述べた PS 並列化におけるローカル競合解消とグローバル競合解消に類似している．ただし，単純に各プロセッサでのローカルな類似概念検索の完了を待つような動作では，最も検索に時間のかかったプロセッサによって検索時間が支配されてしまう．そのプロセッサは，ある時点ではすでに最類似概念とはなり得ない概念を対象として検索をしているかも知れないにもかかわらずである．

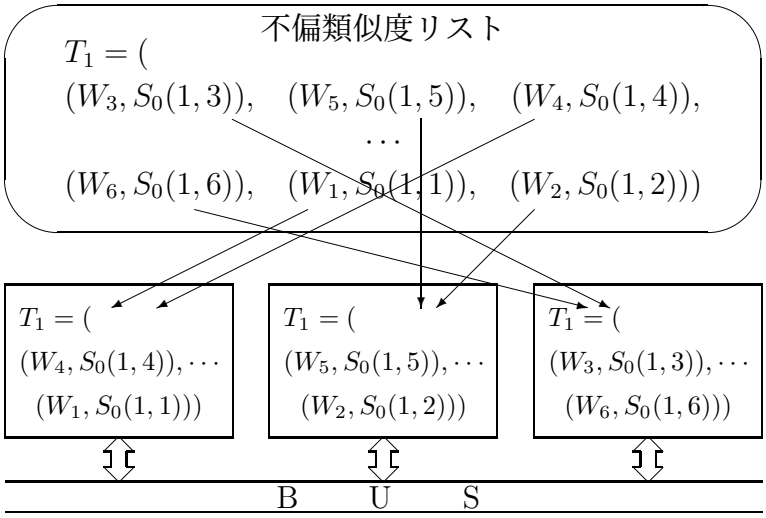
そこで本方式では，ローカルな類似概念検索が完了したプロセッサがその情報を放送することにより，他のプロセッサが検索の打ち切りを判断できるようにしている．また，PS の競合解消フェーズ並列化と同様に，ローカルでの検索処理と，プロセッサ間での結果を比較処理とをオーバラップさせることで，すべてのプロセッサでのローカル検索が完了した瞬間に最終的な結果が得られるようにしている．

本方式での各プロセッサ (ここでは P_p とする) の具体的な動作は以下のようにする．

1. 自プロセッサに割り付けられた概念に対して UB-FSR による類似概念探索を行う．ある時点においてプロセッサ P_p で見つかっている最大類似度を $S_{\max}[p]$，次に不偏類似度リストから取出す概念の上界を $U[p]$ と書くことにする．
2. プロセッサは，自プロセッサ内でのローカルな最類似概念を発見できたら，その概念と類似度をそれをバスの放送機能を利用して全プロセッサに通知し，同期フラグを放送して同期待ちに状態に入る．このプロセッサをここでは P_q とする．
3. P_q の最類似概念を受信した他のプロセッサ P_p は，以下のように動作する．
   (a) 自プロセッサでの現時点での最大類似度が受信された最大類似度より大き



(a) 概念の割り付け



(b) 不偏類似度リストの割付け

図 5.2: バス結合型並列コンピュータ向け類似概念検索方式の概念割り付け

い場合，すなわち $S_{\max}[p] > S_{\max}[q]$ の場合 → そのまま検索を継続.

(b) $S_{\max}[p] \leq S_{\max}[q]$ で，かつ，$U[p] \geq S_{\max}[q]$ の場合 → そのまま検索を継続.

(c) $S_{\max}[p] \leq S_{\max}[q]$ で，かつ，$U[p] < S_{\max}[q]$ の場合 → プロセッサ P_p に割り付けられた概念の中に最類似概念は存在しないことが明らかなので，検索を中止し同期フラグを送信して同期待ちに入る．この際に，最類似概念を保持する変数 $\mathcal{W}_{\max}[p]$ に $W_{\max}[q]$ を，最大類似度を保持する変数 $\mathcal{S}_{\max}[p]$ に $S_{\max}[q]$ をそれぞれ格納する．

4. 同期待ちに入ったプロセッサも，常に他のプロセッサ (ここではプロセッサ P_r とする) からの最類似概念を受信し，自プロセッサが保持する最大類似度 $\mathcal{S}_{\max}[p]$ との比較を行う．もしも，受信した最類似概念の類似度 $S_{\max}[r]$ の方が大きければ，$\mathcal{W}_{\max}[p]$ に $W_{\max}[r]$ を，$\mathcal{S}_{\max}[p]$ に $S_{\max}[r]$ をそれぞれ格納する．
5. すべてのプロセッサが同期待ち状態に入った時点で，プロセッサ上の変数 $\mathcal{W}_{\max}[p]$ および $\mathcal{S}_{\max}[p]$ に最類似概念と最大類似度が得られる．これらは，すべてのプロセッサ上で同じ値が得られる．

ローカルで得られた類似概念検索結果の送信には 3.3.3 節で述べた放送機能を，プロセッサ間の同期には 3.3.3 節で述べた同期機能をそれぞれ利用することにより，ほとんどオーバヘッドなしに実現できる．

5.2.4 解析的性能評価

ここでは，本高速検索方式の性能を評価する．まず，速度向上を推定する計算式を解析的に導出し，これによりさまざまなパラメータに対する性能評価を可能とする．つぎに，本解析評価が実際と合うことを確認するために，小規模な実験を行う．解析評価での速度推定式は，まず逐次処理の場合に対し求め，それに基づき並列化時の推定式を導出する．

本並列化方式は，原理的に通信オーバヘッドや処理量ばらつきがほとんどないため，速度向上に影響する可能性があるのは，概念ベースを各プロセッサに分割したことによる規模の変化である．しかし，逐次処理での導出過程において，速度向上度の規模への依存がないことが明らかにされる．このため，解析評価の妥当性確認のための実験評価は，逐次処理のみを対象とする．

高速検索方式の性能指標である，基本方式に対する高速化の度合い，すなわち速度向上度を定義する．逐次処理での速度向上度 V および並列処理時の速度向上

$V'(P)$ は以下のように書ける．

$$V = \frac{M}{M_X} \tag{5.23}$$

$$V'(P) = \frac{M}{M'_X(P)} \tag{5.24}$$

ここで，M は概念ベースに含まれる概念数，M_X は逐次処理において最大類似度の概念を決定するために類似度を計算する概念数の平均値であり，$M'_X(P)$ は並列処理時のそれを指す．また，P はプロセッサの数である．

(5.23) 式において，M は概念ベースの規模として容易に与えられるため，M_X や $M'_X(P)$ を導出できれば速度向上度が求まる．まず，0 でない重みを持つ属性はランダムに発生すると近似する．この近似により不偏類似度の確率分布を推定することができる．確率分布がわかれば，ある不偏類似度を持つ概念の確率密度も計算することができる．そして，この確率密度から M_X を導き，逐次処理での速度向上度 V を計算する．

不偏類似度の確率分布

二つの概念の間の不偏類似度は，両概念ともに重みが 0 でない属性の数に依存する．言い換えれば，概念において重みが 0 でない属性がランダムに生起するとして，二つの概念で同一の属性が同時に生起する確率に依存するととらえることもできる．このことから，不偏類似度の確率分布は指数分布に従うことになる．不偏類似度が S_a から S_b の間におちる概念の数は，以下で計算される．

$$F_M(S_a, S_b) = M \int_{S_a}^{S_b} \frac{1}{\lambda} e^{-\frac{1}{\lambda}x} dx, \tag{5.25}$$

ここで，λ は不偏類似度の期待値であり，実際の概念ベースからの測定によれば 0.05 程度である．

類似度を計算すべき概念の数 (M_X)

最大類似度を S_x とする．本類似検索方式では，不偏類似度が大きい概念から順に，その上界値及び与えられた観点の元での類似度を順次計算してゆき，上界値が S_x より小さくなった場合に検索を完了する．このことから，検索が完了するまでに計算が行われる概念は，上界値が S_x より大きい概念のみである．(5.20) 式から，ある上界値 U を与える不偏類似度 $S_0(U)$ は以下の式により計算できる．

$$S'_0(U) = \frac{U}{1 + (1 - U)(C^2 - 1)}. \tag{5.26}$$

上述の通り，上界値が S_x より小さくなるまで検索が行われることから，検索対象となる概念が持つ不偏類似度 $S_0(w)$ に対して，$1 \leq S_0(w) \leq S'_0(S_x)$ が成り立つ．この範囲の不偏類似度を持つ概念の個数がそのまま検索対象の概念数 M_x である．これは実は S_x の関数となり，(5.25) 式および (5.26) 式を用いて，

$$\begin{aligned} M_X(S_x) &= F_M(S'_0(S_x), 1) \\ &= M \int_{S'_0(S_x)}^{1} \frac{1}{\lambda} e^{-\frac{1}{\lambda}x} dx \end{aligned} \tag{5.27}$$

となる．

速度向上度 V

上述の議論から，速度向上度 V は以下の式で計算できる．

$$\begin{aligned} V &= \frac{M}{M_X(S_x)} \\ &= \frac{M}{M \int_{S'_0(S_x)}^{1} \frac{1}{\lambda} e^{-\frac{1}{\lambda}x} dx} \\ &= \frac{1}{\int_{S'_0(S_x)}^{1} \frac{1}{\lambda} e^{-\frac{1}{\lambda}x} dx} \end{aligned} \tag{5.28}$$

速度向上度は，最大類似度 S_x と変調度 C との関数になっている．また，概念数 M には依存しないことがわかる．実際には，導出に用いた近似が成り立つ程度には大きな概念数が必要であるが，概念ベースの規模 M には直接は依存しない．

並列処理の場合，プロセッサに割り付けられる概念数は M/P となるが，上述のように速度向上度が概念ベースの規模に依存しないことから，プロセッサ内での速度向上度は (5.28) 式と同一となる．すなわち，

$$\frac{\frac{M}{P}}{M'_x(p)} = \frac{1}{\int_{S'_0(S_x)}^{1} \frac{1}{\lambda} e^{-\frac{1}{\lambda}x} dx} \tag{5.29}$$

であり，これより，

$$\begin{aligned} \frac{M}{M'_x(p)} &= \frac{P}{\int_{S'_0(S_x)}^{1} \frac{1}{\lambda} e^{-\frac{1}{\lambda}x} dx} \\ V'(P) &= V \cdot P \end{aligned} \tag{5.30}$$

となる．これは，本並列化方式では，プロセッサ台数に比例した速度向上度が得られることを示している．実際には，上でも述べたように，プロセッサに割り付けら

れる概念の数が推定式導出のための仮定が成り立つ程度には多くなければならない．また，バスによりデータ転送や同期についてのオーバヘッドを無視できるプロセッサ数にも限度があり数台〜10 数台が実用的な範囲である．以上より，逐次処理に対する高速化に加えて，さらに 10 数倍程度の高速化が本並列化方式によって達成できると言える．

上記の推定式を用いて変調度 C が 1.2 から 5 までの範囲に対する速度向上度を推定した．その結果を図 5.3 に示す．図からわかるように，本方式では $S_x > 0.8$ で $C < 3$ の範囲において 1000 倍以上の速度向上が得られる．最類似概念は，類似概念の中でも最も大きな類似度を持つ概念であるため，一般的には 0.8 以上といったかなり大きな類似度を持つことになる．加えて，実際の概念ベースシステムを用いた実験から，適切な変調度は 1.5 から 2.5 程度であることがわかっている．以上より，本方式は実用的な類似度と変調度の範囲において非常に良好な速度向上を達成できると言える．

5.2.5 実験評価

解析評価の妥当性を検証するために，実際の概念ベースを使って実験を行った．実験に用いた概念ベースは日常的に用いられる約 4 万の概念を持つ．実験システムは，与えられた概念と観点に対し，概念ベースのなかでもっとも大きな類似度を持つ概念とその類似度を UB-FSR 方式により検索して結果を提示するものであ

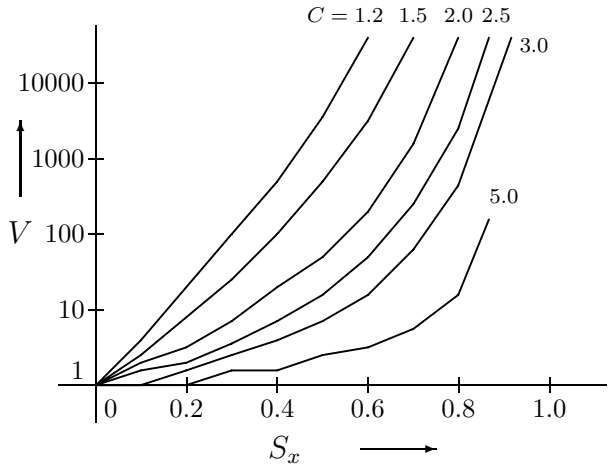


図 5.3: UB-FSR の速度向上度推定値

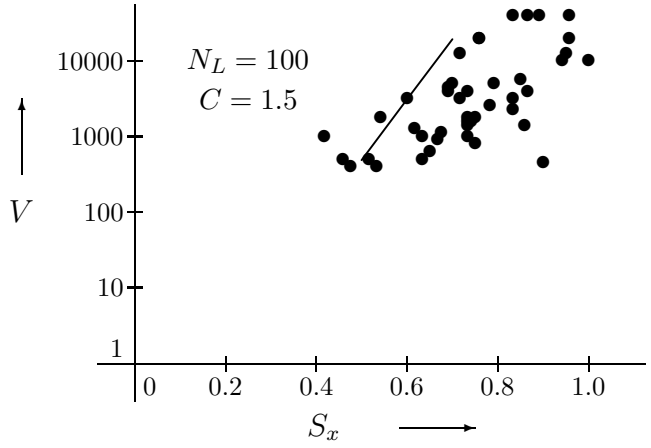
る．速度向上度は，概念ベースに含まれる概念数を検索に際して類似度を実際に計算した概念の数で除したものとして計算した．

実験では，概念ベースから 50 個の概念と観点とをそれぞれ無作為に選び，これらを用いて類似概念検索を試行した．図 5.4 に実験結果を示す．個々の試行における類似概念の類似度と速度向上度とを点として表している．前節で計算した速度向上度の推定値を実線で表している．図からわかるように，ほとんどの試行において 1000 倍以上の速度向上が得られている．並列処理すれば 10000 倍程度の速度向上が期待できる．

5.3 SNAP による類似概念検索の並列化

本節では，より一般的な直接網で結合された並列コンピュータ上で類似概念検索を並列処理する手法について論じる．もちろん 5.2 節のおいてバス結合型並列コンピュータ向けとして提案した方法をこれに適用することも可能であるが，ここでは，より一般的な結合網に適した概念ベース格納法と処理方法を提案する．

概念を表現し処理することを目的として，古くから意味ネット [13, 8, 32] が提案されている．意味ネットでは，概念をノードをリンクによって結んだネットワークとして表現し，リンクにマーカを伝播させることにより処理を行う．また，マーカ伝播を高速処理するために，これに特化した一般直接網結合の高並列アーキテクチャとして Semantic Network Array Processor(SNAP) が提案されている [66]．



V：速度向上度　　—：推定値
S_x：最大類似度

図 5.4: 実験結果

概念ベースを意味ネットとして表現し，類似概念検索を SNAP によって高速処理する方式を提案する．概念ベースでは，5.1.2 節で述べたように，概念をそれに対する定義語及びその語彙体系上での包含によって表す．また，それらの定義語間の関係および包含関係に基づいて類似判別が行われる．このことから，概念，定義語，語彙体系上のカテゴリをノードに対応させ，それらの関係をリンクに対応させることにより，概念ベースを意味ネットとして表現することができる．また，この意味ネット上でマーカを伝播させることによって，概念や定義語の間の関係，それぞれの包含関係に関する情報を収集することができ，この情報に基づいて類似判別を行うことができる．

以降では，まずマーカ伝播モデルと SNAP について概説し，次にこれを用いた類似概念検索方式を提案する．また，実際の概念を用いてシミュレータにより類似計算を行い動作を確認する．

5.3.1 マーカ伝播モデルと SNAP アーキテクチャ

SNAP は米国南メソジスト大学の Moldovan が提唱する意味ネット向け高並列コンピュータである [66]．スケーラブルなアレイ・プロセッサにより，意味ネットで表された知識ベースとマーカ伝播とにより検索，推論などを高速処理することを目指している．知識や概念は意味ネットの形式で表現され，SNAP のアレイ上に分散して格納される．

SNAP での推論は，この分散格納された意味ネット上にマーカを伝播させることにより行われる．マーカは並列に独立して伝播できるため，問題を適切な意味ネット上のマーカ伝播として定式化してやれば，問題が本質的に持つ並列性を最大限に引き出すことができる．SIMD 型の SNAP-1，MIMD 型の SNAP-2 が開発されている．アプリケーションとして，自然言語処理システム，認識処理システムなどが研究されている．

5.3.2 概念ベースの意味ネット表現

5.1.2 節で述べたように，概念ベースにおける概念は属性名 (定義語の属するカテゴリ)–重みペアの集合として表現されている．概念と属性名とをノードとし，この間を “Defining_Feature” リンクで結んで重みを付与すれば，概念ベースを意味ネットとして表現できる．意味ネットでは，概念間に 2 段以上のリンクをはさん

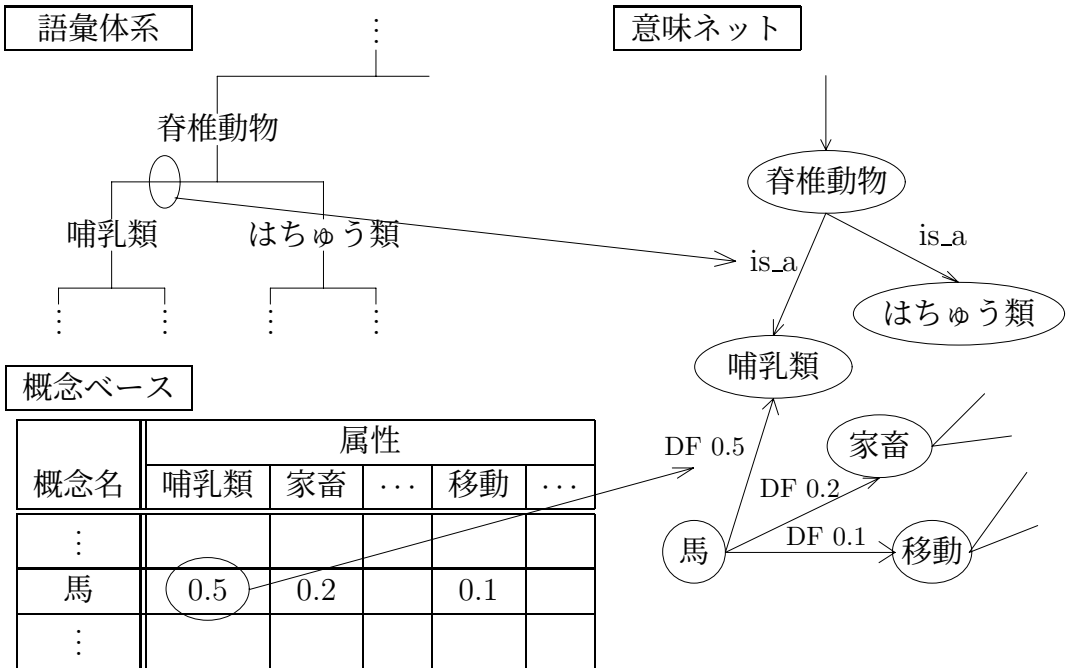


図 5.5: 概念ベースの意味ネットによる表現

だ間接的な関連があるかどうかが容易に判別でき，直接の定義関係のみが明確であった従来の概念ベースに比較して見通しの良いものとなっている．また，類似判別には直接関連しないが，語彙体系上の関係等も，同時にこの意味ネット上に射影すれば，推論や意味理解にもひとつの意味ネットを活用することが可能となる．図 5.5 に本表現手法の例を示す．なお，図では Defining_Feature リンクを “DF” と表記している．

5.3.3 類似判別方式

類似度計算のアルゴリズムである (5.5)〜(5.10) 式について再び考える．観点による変調では，観点が持つものと同一の属性に対して，その重みを定数倍する．続いて，正規化では，単位ベクトルとなるように，変調後の重みの長さ (自乗和の平方根) によって，それぞれの重みを除している．そして，類似度計算の対象となる二つのベクトルの内積を計算する．観点に含まれる (重みが 0 でない) 属性のインデックス集合を $\mathbf{V}$ と表せば，これら一連の演算は，まとめて以下のようにも書

ける．

$$\begin{aligned} S_v(i,j) &= \sum_{m\notin\mathbf{V}} \frac{q_i[m]}{L_{i(v)}} \cdot \frac{q_j[m]}{L_{j(v)}} + \sum_{m\in\mathbf{V}} \frac{Cq_i[m]}{L_{i(v)}} \cdot \frac{Cq_j[m]}{L_{j(v)}} \\ &= \frac{1}{L_{i(v)} \cdot L_{j(v)}} \left(\sum_{m\notin\mathbf{V}} q_i[m] \cdot q_j[m] + C^2 \sum_{m\in\mathbf{V}} q_i[m] \cdot q_j[m] \right) \end{aligned} \tag{5.31}$$

ここで，$L_{i(v)}$，$L_{j(v)}$ は変調後のベクトル長であり，(5.15) 式を参照し，また，概念表現の定義より $\sqrt{\sum_m q_i[m]^2} = 1$ であることに着目すれば，

$$\begin{aligned} L_{i(v)} &= \sqrt{\sum_{m\notin\mathbf{V}} q_i[m]^2 + \sum_{m\in\mathbf{V}} (Cq_i[m])^2} \\ &= \sqrt{\sum_m q_i[m]^2 + (C^2-1) \sum_{m\in\mathbf{V}} q_i[m]^2} \\ &= \sqrt{1 + (C^2-1) \sum_{m\in\mathbf{V}} q_i[m]^2} \end{aligned} \tag{5.32}$$

と書ける．

(5.31) 式のかっこ内第 1 項は，概念 W_i と W_j の属性のうち観点 W_v が持つ属性に含まれないものに対する重みの積和である，また第 2 項は同様の属性のうち観点が持つ属性のみに関しての重みの総和である．ただし，概念 W_i か概念 W_j の一方にしか含まれない属性は，他方の重みが 0 のため，これら積和には寄与しない．すなわち，概念 W_i と W_j の属性の共通の属性に対し，それが観点の持つ属性に含まれるかどうかがわかれば，かっこ内を計算できることになる．また，(5.32) は，ある概念 W_i の属性のうち観点 W_v が持つ属性に含まれるものだけから，ベクトルの長さが計算できることを表している．

以上より，概念ベースにおいて以下の属性を選別することによって類似度が計算できることになる．

- 概念 W_i と観点 W_v との両方に含まれる属性
- 概念 W_j と観点 W_v との両方に含まれる属性
- 概念 W_i と観点 W_j との両方に含まれ，かつ，観点 W_v に含まれない属性
- 概念 W_i と観点 W_j との両方に含まれ，かつ，観点 W_v にも含まれる属性

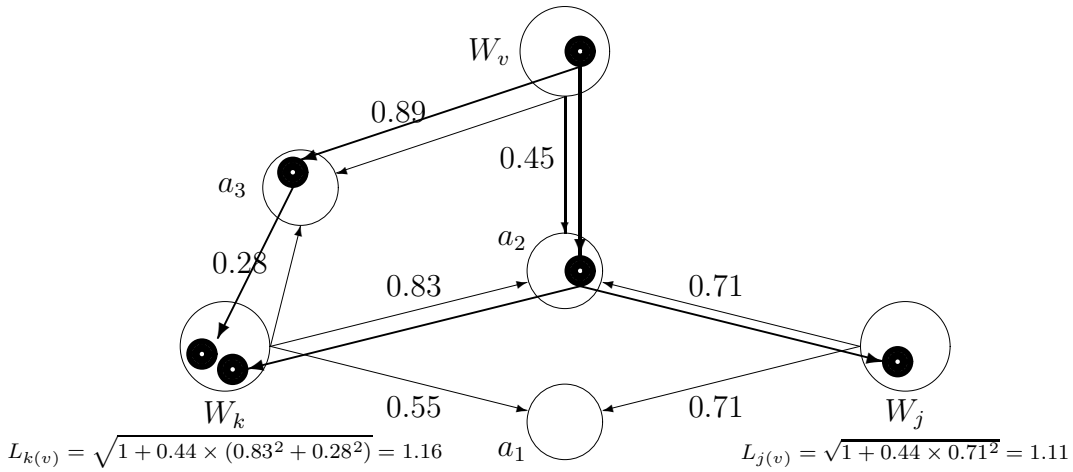
SNAP では，マーカ伝播により上記の属性を選別させ，さらに，リンク上の重みを伝播するマーカに運搬させることにより，上式の計算も同時に行わせること

が可能である．基本的な原理は，マーカを，ある概念から Defining_Feature リンクの正方向に 1 ステップ，続いて逆方向に 1 ステップ伝播させることにより，同一の属性を持つすべての概念にマーカを到達させることである．この原理を用いた本方式の類似度計算手順を以下に示す．

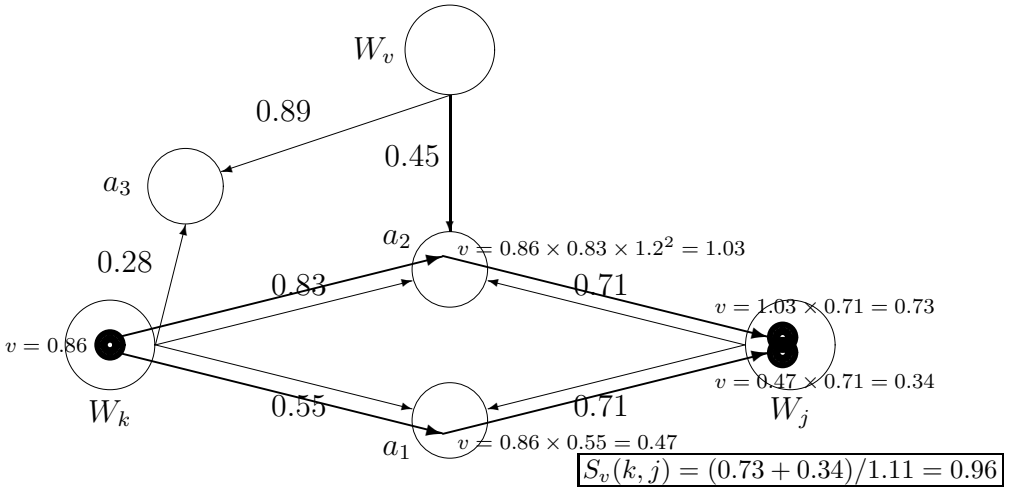
1. 観点 W_v より上述の原理に基づきマーカを伝播させる．Defining_Feature リンクを逆方向に伝播する際に，そのリンク上の重みを運搬する．また，観点からマーカが到達した属性には，そのマーカのコピーを留置する．
2. マーカが一つでも到達した概念ノードでは，到達したすべてのマーカを集めて (5.32) 式を計算する．すなわち，到達したマーカの持つ重みの自乗和を計算し，これに (C^2-1) をかけて 1 を足し，平方根を求める．これによって個々の概念において変調後のベクトル長さ $L_{i(v)}$ が計算される．マーカがまったく到達しない概念ノードは，観点と共通の属性を持たないため，ベクトル長さは変化せず 1 のままである．
3. 類似概念検索においてキーとなる概念 W_k から，原理に基づきマーカを伝播させる．マーカは，概念 W_k のノードからの発出時には，step.2 で計算した W_k のベクトル長 $L_{k(v)}$ の逆数，すなわち $1/L_{k(v)}$ を運搬する．
4. Defining_Feature を正方向に伝播する際に，運搬している値に，そのリンクの重みを乗じて新たな値とする．
5. 属性ノードを経由して Defining_Feature リンクを逆方向に伝播する際，さらにそのリンク上の重みを乗じて新たな値とする．ただし，属性ノードに観点からのマーカが存在している場合には，それを C^2 倍する．
6. マーカが到達した概念ノードにおいて，すべてのマーカが運搬した値を総和を計算した後，自ノードでのベクトル長さによって除して，その値を類似度とする．

以上の手順から，ステップ 2 において (5.32) 式がすべての概念ノードにおいて同時に計算され，ステップ 3〜6 によってすべての検索対象概念での類似度，すなわち (5.31) 式が計算される．

図 5.6 に，実際の概念を用いた類似度計算の動作を示す．まず，マーカは観点となる概念 W_v にセットされ，そこから出るすべての Defining_Feature リンク上を伝播する．この伝播によってマーカが属性 a_2 および a_3 に到達するので，ここにマーカを留置するとともに，さらにそれらのノードに入っている Defining_Feature リ



(a) 観点変調とベクトル長の計算 ($C = 1.2$)



(b) 類似度の算出

図 5.6: SNAP による類似度計算

ンクを逆方向に伝播させる．この伝播の際に，マーカはそれぞれのリンク上の重みを運搬する (step.1)．

検索対象概念 W_j には属性 a_2 を通過したマーカが到達する．このマーカは重み 0.71 を持っている．ノード W_j において，(5.32) 式によってベクトル長が計算され，$L_{j(v)} = 1.11$ を得る．一方，キー概念 W_k には，属性 a_2 を通過したマーカと属性 a_3 を通過したマーカとが到達し，それぞれが重み 0.83 と 0.28 とを持つ．W_j と同様に (5.32) 式によってベクトル長が計算され，$L_{k(v)} = 1.16$ を得る (step.2)．

続いて，キー概念 W_k 上にマーカをセットし，Defining_Feature リンク上を正方向に伝播させる．この際，マーカは W_k のベクトル長 $L_{k(v)} = 1.16$ の逆数 0.86 を値として持つ (SETP.3)．マーカは，W_k から出ている 2 本の Defining_Feature リンクをそれぞれ伝播する．この際，リンク上の重み 0.55 および 0.83 がそれぞれのマーカの値に乗じられる．属性ノード a_1 および a_2 に到達すると，a_2 では step.1 において観点からのマーカが留置されているため，さらに $C^2 = 1.44$ が乗じられる．この結果として，ノード a_1 に到達したマーカは 0.47，a_2 に到達したマーカは 1.03 をその値として持つことになる．これらマーカをさらに Defining_Feature リンクを逆方向に伝播させながら，マーカが持つ値をリンク上の重みを乗ずる．マーカはどちらも W_j に到達し，それぞれ値として 0.34 および 0.73 が得られる (step.4〜5)．

W_j では，到着した二つのマーカの持つ値を総和した後，その値を自ノードでのベクトル長 $L_{j(v)} = 1.11$ で除すことで，その値を類似度として得る．この例では，類似度 0.96 が得られている (step.6)．

このような手順に基づいてすべての概念に対して類似度を計算し，それらを収集して最も大きな類似度を見いだせば，それが最類似概念となる．すべてのマーカ伝播は並列的に行われるので，プロセッサ数が十分に大きければ同時にすべての概念に対する類似度を計算でき，意味ネットの規模にかかわらず一定時間で類似概念検索が可能である．

5.4 まとめ

この章では，概念ベースでの類似概念検索を並列処理により高速化する手法を提案した．

まず，概念の類似度は文脈や状況などにより変化することを指摘し，これを「観点」と呼んで観点に基づく類似度計算法を示した．観点に基づく類似度をすべて

の概念に対して計算すれば，与えられた観点のもとで与えられた概念に最も類似した概念を見いだす類似概念検索を実現できる．しかし，これには膨大な量の計算が必要となり類似概念検索の応用領域を大きく制限されるため，何らかの高速化手法が必要となる．そこで，バス結合型アーキテクチャ向け，及び，より一般的な直接網アーキテクチャ向けの並列化方式をそれぞれ提案した．

バス結合型アーキテクチャ向けの並列化方式では，処理を並列化する前段階として，類似度計算の特性を詳細に考察し検索対象となる概念を大幅に絞り込む手法を提案した．類似度は観点によって変化するが，その変化範囲の上限は不偏類似度から計算できる点をに着目し，上界がその時点で見つかっている類似度を越えないことを検出することで，見込みのない概念に対する類似度計算を不要とした．これにより探索空間を 1/1000 程度に削減し，さらにこれを並列化する方式を提案した．並列化では，各プロセッサで自プロセッサの持つ概念に対し検索を行い，その結果を最後に統合することを原理としている．ただし，プロセッサは完全に独立に検索するのではなく，他のプロセッサでの検索結果を監視して，見込みがない場合には検索を中断する．これにより，分割による処理量増加 (分割損) を小さく抑えている．また，3 章で提案したと同一構成の並列コンピュータを用いることで，結果統合の際の通信や同期の効率的に行っている．

解析的な評価を行い，本方式によれば 10000 倍以上の速度向上度が期待できることが示された．また，実際に 4 万概念を持つ概念ベースを利用して実験も行い，実験結果は解析評価のグラフ上にのっており，解析が支持されることを確認した．

より一般的な直接網による結合アーキテクチャに対しては，概念ベースを意味ネットとして表しマーカ伝播により観点に基づく類似度を計算する並列化方式を提案した．観点と概念の間，概念と概念の間のリンク連接をマーカ伝播させることにより類似度を計算する．マーカ伝播はスケーラブルなため高並列向けであり，プロセッサ数が十分に大きければ原理的に意味ネットの規模にかかわらず一定時間で処理できる．このため，非常に高速に類似概念検索行うことができる．

第6章　結論

本論文では，人工知能システムを疎結合型並列コンピュータを用いて高速化するための並列化方式について論じた．代表的な人工知能システムあるプロダクション・システム (PS)，ニューラルネット (NN) でのバックプロパゲーション (BP) 学習，概念ベース・システムの類似概念検索に対し，これらの並列化方式を提案し，その性能を評価した．

並列化研究では，並列コンピュータのアーキテクチャと並列化方式があった場合に，その速度向上度を推定できることが重要である．実際に並列化方式を並列コンピュータ上に実装して測定するのでは，実装に多くの時間を要するし，測定に用いるアプリケーションや問題も限定されてしまうため，多数の並列化方式を広範な問題領域にわたって同一条件下で比較検討することが困難である．このため本論文では，まず，対象とした人工知能システムそれぞれに対して，並列化時の速度向上度やオーバヘッドを解析的に計算する手法を提案した．そして，これら性能推定手法に基づき，良好な性能が得られる並列化方式をそれぞれ提案し，実験やシミュレーションにより性能を検証した．

PS の並列化では，推論中で最も多くの処理時間を消費する条件照合について，それに用いられている RETE ネットワークをモデル化し，問題特性から RETE ネットワークの構造や処理の特性を推定できるようにした．これに基づき，いくつかの並列化方式に対して性能の評価を行い，PS 条件照合の並列化では 10 倍〜数十倍程度の速度向上が限界であることを示した．これは，従来の研究を裏付けている [60]．従来方式が対象としたアーキテクチャよりも低廉に構成可能なバス結合型アーキテクチャを用いても限界に近い性能を得られる並列化方式として “TWIN 方式” を提案した．PS 並列化では，一般に粒度を細かくすれば通信が増加し，粒度を粗くすれば負荷ばらつきが増加する．バス結合型アーキテクチャでは，通信頻度が多い場合にバス上でデータ転送の衝突が起こるため通信オーバヘッドが急激に増大する．TWIN 方式では，RETE ネットワークの構造的な並列性と，個々のノードでの処理の並列性とを巧妙に組み合わせることにより，通信を増加させ

ずに粒度を細かくした方式である．このため，通信オーバヘッドと負荷ばらつきの両者を小さく抑えることに成功している．解析的な評価により，PS 並列化の限界である 10 倍〜数十倍の速度向上が期待できることがわかった．

更に，バスに TWIN 方式を実装するに適した機能を付加した並列コンピュータ “Presto” を提案した．Presto では，通常のデータ転送機能に加えて，放送，同期，割り込み等の機能をハードウェアによって実現している．TWIN 方式を条件照合部の並列化方式として利用して，PS の推論処理全体を並列化する方式を提案し，これを Presto 上に実装した．実験の結果，ほぼ推定通りの性能が得られた．

BP では，その処理を連続行列演算ととらえて，並列化手法を論じた．連続行列演算では，通信量と計算量とが性能の支配要因となることから，これらの推定式を構築した．推定式に対する考察より，どのような並列化方式をもってしてもそれが下回ることがない通信量，すなわち通信量の下限値が存在することを明らかにした．2 次元アレイ (トーラス) 型並列コンピュータに対する並列化方式を提案し，これが通信量の下限値を達成していることを示した．これは，提案した方式よりも通信量が小さい並列化方式は他に存在しないことを意味している．すなわち，この方式が，BP に対する最適な並列化方式である．また，BP 並列化のための並列コンピュータにおける相互結合網は，2 次元アレイ (トーラス) で十分であり，それ以上に複雑なトポロジは必要がないと言える．さらにバス結合型アーキテクチャに対する並列化方式も提案した．この方式は通信ステップ数の下限値は達成していないが，従来の並列化方式では通信量がプロセッサ数に比例したのに対し，この方式ではプロセッサ数の平方根オーダに抑えることができた．

類似概念検索では，まず，並列化の前段階として，検索対象となる概念の空間を限定する手法を提案し，次に，これをバス結合型アーキテクチャ上で並列処理する方式を提案した．概念の検索空間の限定のために，類似度計算式について考察し，類似度が超えることのない値，すなわち上界値を普遍類似度から計算する手法を導出した．上界値を計算できることにより，最類似概念とはなり得ない概念に対する類似度の計算を省くことができ，検索の空間を大幅に削減することが可能となった．これの並列化では，検索空間を部分に分割して各プロセッサに割当て，それぞれが独立に検索を行うが，分割したことにより検索空間削減に関する情報がプロセッサ間で異なって処理量が増加したり，検索結果を統合する際に通信および同期オーバヘッドが大きくなることが問題である．そこで，Presto のバスが持つ放送，同期，割り込み機能を活用してこれら問題を解決した並列化方

式を提案した．解析的評価により提案方式によって逐次処理に対し 10 倍〜数十倍の高速化が可能であることを示した．更に，概念を意味ネットとして表現し，それを処理するに適した高並列アーキテクチャSNAP 上で類似検索を行う並列化方式も提案した．

対象とした人工知能システムのどれに対しても，オーバヘッドの発生要因を的確に捕捉し，それを抑えるような並列化方式を考案することによって，疎結合型並列コンピュータで数十倍以上の速度向上が得られることがわかった．疎結合型並列コンピュータの中でも特にバス結合型並列コンピュータは，低廉に構築できるだけでなく，汎用プロセッサを用い相互結合網のために汎用部品を流用できるため，汎用コンピュータの技術進歩の成果をほぼそのまま適用できるという利点がある．その反面，通信オーバヘッドの増大を招きやすいという欠点があるが，本論文では，このようなバス結合型アーキテクチャを用いても，提案したどの並列化方式もが 10 倍以上の速度向上を得られることを示した．汎用のコンピュータの性能は 1.5 年で 2 倍向上すると言われているため，10 倍の性能向上の達成には 5 年を必要とする．バス結合型並列コンピュータでは上述のように個々のプロセッサとして常に最新のものを利用できるため，並列化による性能向上がそのままシステムの総合的な性能向上となる．このことから，本論文で提案した並列化によって，常に 5 年以上の性能的なアドバンテージが得られることになる．また，プロセッサ技術の進展への追随が比較的容易な直接網による結合に向けて本論文で提案した並列化方式によれば，更なる高速化を達成することも可能である．以上のことから，本研究により，現実的な並列コンピュータを用いて，人工知能システムの実行性能を常に 5 年分以上先行させることが可能になったと結論づけられる．

謝辞

本論文をまとめるにあたり，懇切なるご指導を賜った，京都大学大学院情報学研究科石田亨教授，富田眞治教授ならびに湯淺太一教授に厚く感謝いたします．

本研究は，NTT 情報通信処理研究所および NTT コミュニケーション科学研究所において，多くの方々のご指導を得て行われました．研究の機会を与えて頂くとともに，論文の執筆を勧めて頂いた同志社大学河岡司教授 (元コミュニケーション科学研究所長)，NTT アドバンステクノロジ八巻俊文部長 (元コミュニケーション科学研究所知識処理研究部長)，拓殖大学石川勉教授 (元コミュニケーション科学研究所グループリーダ) に深く感謝いたします．また，Southern Methodist 大学でご指導頂いた Dan I. Moldovan 教授，Sanda M. Harabagiu 教授に感謝いたします．

本研究の遂行には多くの方々の援助と励ましがありました．プロダクションシステム向け並列プロセッサ Presto の設計・試作・性能評価は，NTT アドバンステクノロジ菊地英夫部長，NTT サービスインテグレーション基盤研究所松澤和光主幹研究員，NTT エレクトロニクス水書章雄課長から多大な協力を受け行われました．概念の類似検索システムでは，NTT コミュニケーション科学基礎研究所笠原要研究主任から概念ベースの提供を受けました．Dan I. Moldovan 教授からは SNAP シミュレータを提供して頂きました．これらの方々に深く感謝いたします．また，NTT コミュニケーション科学基礎研究所佐々木裕主任研究員，阿部明典主任研究員，藤本和則研究主任，NTT 情報流通仲林清部長，NTT アドバンステクノロジ桃井茂晴部長には日頃から討論頂きました．NTT ドコモ石垣昭一郎部長，NTT 西日本森原一郎部長，NTT 串間和彦部長には様々な観点からコメントを頂きました．あわせてお礼を申し上げます．

参考文献

[1] 阿部明典. 欠如節を生成する推論法. 電子情報通信学会論文誌, Vol. J81-D-II, No. 6, pp. 1285–1292, 1998.

[2] D. H. Ackely, Geoffrey E. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, Vol. 9, , 1985.

[3] S. M. Aiken, M. W. Koch, and M. W. Roberts. A parallel neural network simulator. In *Proc. IEEE International Joint Conference on Neural Networks*, pp. 611–616, 1990.

[4] S. W. Aiken, M. W. Koch, and M. W. Roberts. A parallel neural network simulator. In *Proc. IEEE International Joint Conference on Neural Networks*, Vol. 2, pp. 611–616, 1990.

[5] 荒屋真二, 百原武敏, 田町常夫. プロダクションシステムのための高速パターン照合アルゴリズム. 情報処理学会論文誌, Vol. 28, No. 7, pp. 768–775, 1987.

[6] 麻生英樹. ニューラルネットワーク情報処理. 産業図書, 1988.

[7] N. Babaguchi, Y. Sawada, and T. Ohkawa. Association mechanism based on viewpoint. 情報処理学会論文誌, Vol. 35, No. 5, pp. 714–724, 1994.

[8] R. J. Brachmann. On the epistemological status of semantic networks. In N. V. Findler, editor, *Associative Networks*. Academic Press, New York, 1979.

[9] L. Brownston, R. Farrell, E. Kant, and N. Martin. *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming*. Addison-Wesley, Reading, Mass, 1985.

[10] B. G. Buchanan and E. A. Feigenbaum. DENDRAL and Meta-DENDRAL: Their applications dimension. *Articicial Intelligence*, Vol. 11, , 1978.

[11] P. L. Butler, J. D. Allen, and D. W. Bouldin. Parallel architecture for OPS5. In *Proc. 15th International Symposium on Computer Architecture*, pp. 452–457, 1988.

[12] B. D. Clayton. ART programming tutorial. Technical report, Inference Corp., 1987.

[13] A. M. Collins and M. R. Quillian. How to make a language user. In E. Tulving and W. Donaldson, editors, *Organization and Memory*. Academic Press, New York, 1972.

[14] P. T. Cox and T. Pietrzykowski. Causes for events: Their computation and applications. In *Proc. 8th International Conference on Automated Deduction*, pp. 608–621, 1986.

[15] R. Domolombe and L. Farinas de Cerro. An inference rule for hypothesis generation. In *Proc. International Joint Conference on Artificial Intelligence 91*, pp. 152–157, 1991.

[16] E. A. Feigenbaum, B. G. Buchanan, and J. Lederberg. On generality and problem solving: A case study using DENDRAL program. *Machine Intelligence*, Vol. 6, pp. 165–190, 1971.

[17] C. L. Forgy. Rete: a fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, Vol. 19, No. 1, pp. 17–37, 1982.

[18] B. M. Forrest, D. Rowth, N. Stroud, D. J. Wallece, and G.V.Wilson. Implementing neural network models on parallel computers. *The Computer Journal*, Vol. 30, No. 5, pp. 413–419, 1987.

[19] Y. Fujimoto. An enhanced parallel planar lattice architecture for large scale neural network simulations. In *Proc. IEEE International Joint Conference on Neural Networks*, Vol. 2, pp. 581–586, 1990.

[20] Y. Fujimoto. An enhanced parallel planar lattice architecture for large scale neural network simulations. In *Proc. IEEE International Joint Conference on Neural Networks*, pp. 581–586, 1990.

[21] A. Gupta. Implementing OPS5 production systems on DADO. In *Proc. 13th Int. Conference on Parallel Processing*, pp. 83–91, 1984.

[22] A. Gupta. *Parallilism on Production Systems*. Morgan Kaufmann, 1987.

[23] A. Gupta and C. L. Forgy. Measurements on production systems. Technical report, Carnegie Mellon University, 1984.

[24] A. Gupta, C. L. Forgy, D. Kalp, A. Newell, and M. S. Tambe. Results of parallel implementation of OPS5 on the Encore multiprocessor. Technical Report CS-87-146, Carnegie Mellon University, 1987.

[25] A. Gupta, C. L. Forgy, D. Kalp, A. Newell, and M. S. Tambe. Parallel OPS5 on the Encore Multimax. In *Proc. 17th Int. Conference on Parallel Processing*, pp. 271–280, 1988.

[26] A. Gupta, C. L. Forgy, A. Newell, and R. G. Wedig. Parallel algorithm and architectures for rule-based systems. In *Proc. 13th International Symposium on Computer Architecture*, pp. 28–37, 1986.

[27] N. V. Ha, T. Ishikawa, and A. Abe. A mechanism for inferring approxiamate solutions under incomplete knowledge based on rule similarity. In *Proc. AISTA 2000*, 2000.

[28] K. J. Hammond. CHEF: a model of case-based planning. In *Proc. AAAI-86, 3rd National Conference on Artificial Intelligence*, pp. 267–271, 1986.

[29] S. M. Harabagiu and D. I. Moldovan. A marker-propagation algorithm for text coherence. In *Parallel Processing in AI Workshop, IJCAI*, 1995.

[30] 長谷川, 高木編. 高性能ワークステーション. 情報処理学会誌, Vol. 25, No. 2, 1984.

[31] 服部文男, 清水信昭, 土屋秀幸, 桑原和宏, 和佐野哲男. 知識ベース管理システム (KBMS). 情報処理学会 知識工学と人工知能研究会, Vol. 41, , 1985.

[32] G. G. Hendrix. The representation of semantic knowledge. In D. E. Walker, editor, *Understanding Spoken Language*. Elsevier North-Holland, New York, 1978.

[33] W. D. Hillis. *The Connection Machine*. MIT Press, 1985.

[34] J. J. Hopfield. Neural networks and physical systems with emergent collective computational properties. *Proc. National Academy of Sciences*, pp. 2554 – 2588, 1982.

[35] J. J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. National Academy of Sciences*, Vol. 81, pp. 3088 – 3092, 1984.

[36] 井田哲雄, 山本昌弘, 竹内郁男編. 記号処理と計算機アーキテクチャ. 情報処理学会誌, Vol. 23, No. 8, 1982.

[37] 池原悟, 宮崎正弘, 白井論, 横尾昭男, 中岩浩巳, 小倉健太郎, 大山芳史, 林良彦（編）. 日本語語彙大系. 岩波書店, 1997.

[38] S. Ikehara. Multi-level machine translation system. *Future Computer System*, Vol. 1, pp. 261–274, 1989.

[39] 井上克己. アブダクションの原理. 人工知能学会誌, Vol. 7, No. 1, pp. 48–59, 1992.

[40] 石田亨. プロダクションシステムの並列実行可能性の解析. 電子情報通信学会論文誌, Vol. J71-D, No. 3, 1988.

[41] 石田亨. プロダクションシステムの発展, 朝倉 AI らいぶらり, 第 5 巻. 朝倉書店, 1996.

[42] 石田亨, 片桐恭弘, 桑原和宏. 分散人工知能, 並列処理シリーズ, 第 11 巻. コロナ社, 1996.

[43] T. Ishida. Optimizing rules in production systems programs. In *Proc. AAAI-88, 5th National Conference on Artificial Intelligence*, pp. 699–704, 1988.

[44] 石塚満. 知識表現と高速推論. 情報科学コアカリキュラム講座. 丸善, 1996.

[45] S. Ishigaki, I. Morihara, and K. Kushima. KBMS –an expert system building tool–. *Review of the ECL, NTT, Jpn.*, Vol. 37, pp. 3–7, 1989.

[46] D. Jackson and D. Hammerstrom. Distributing back propagation networks over the intel iPSC/860 hypercube. In *Proc. IEEE International Joint Conference on Neural Networks*, Vol. 1, pp. 569–574, 1991.

[47] L. H. Jamieson, D. B. Gannon, and R. J. Douglass. *The Characteristics of Parallel Algorithms*. The MIT Press, 1987.

[48] 情報処理学会（編）. 知識工学. オーム社, 1987.

[49] 金田悠紀夫. Prologマシン. 情報処理学会誌, Vol. 25, No. 12, 1984.

[50] K. Kasahara, K. Matsuzawa, and T. Ishikawa. Refinement method for a large-scale knowledge base of words. In *Proc. Symposium on Logical Formalization of Common Sense Reasoning*, pp. 73–82, 1996.

[51] K. Kasahara, K. Matsuzawa, T. Ishikawa, and T. Kawaoka. Viewpoint-based measurement of semantic similarity between words. In *Proc. International Workshop on Artificial Intelligence and Statistics*, pp. 292–302, 1995.

[52] 河田龍夫. 確率論とその応用. 紀伊国屋書店, 1961.

[53] M. A. Kelly and R. E. Seviora. A multiprocessor architecture for production system matching. In *Proc. 13th International Symposium on Computer Architecture*, pp. 28–37, 1986.

[54] H. Kikuchi, T. Yukawa, K. Matsuzawa, and T. Ishikawa. Presto: a bus-connected multiprocessor for a rete-based production system. In *Proc. Joint Conference of CONPAR90/VAPP IV*, 1990.

[55] J. T. Kim and D. I. Moldovan. Classification and retrieval of knowledge on a parallel marker-passing architecture. *IEEE Trans. Knowledge and Data Engineering*, Vol. 5, No. 5, pp. 735–761, 1993.

[56] 薦田憲久, 大川剛直, 安信千津子. エキスパートシステムの設計と開発, 情報系教科書シリーズ, 第 21 巻. 昭晃堂, 1997.

[57] R. M. Kuczewski, M. H. Myers, and W. J. Crawford. Neurocomputer workstaions and processors: Approaches and application. In *Proc. IEEE International Conference on Neural Networks*, Vol. 3, pp. 487–500, 1987.

[58] S. Y. Kung and J. N. Hwang. Parallel architectures for artificial neural nets. In *Proc. IEEE International Conference on Neural Networks*, Vol. 2, pp. 165–172, 1988.

[59] 久野真司, 岩田彰, 吉田征夫. 同報書き込み行列メモリを用いたニューラルネットワークアクセラレータ. 電子情報通信学会全国大会, 第 6 巻, p. 48, 1993.

[60] S. Kuo and D. Moldovan. The state of the art in parallel production systems. *Journal of Parallel and Distributed Computing*, Vol. 15, pp. 1–26, 1992.

[61] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays · Trees · Hypercubes*. Morgan Kaufmann Publishers, 1992.

[62] K. Matsuzawa, T. Ishikawa, and T. Kawaoka. ABOUT project for a robust problem solving and its method of measuring semantic similarity. 人工知能学会研究会, Vol. SIG-J-9401-14, , 1994.

[63] W. Van Melleand, A. C. Scott, J. S. Bennett, and M. Peairs. The Emycin manual. Technical Report STAN-CS-81-885, Computer Science Dept., Stanford University, October 1981.

[64] G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. Five papers on WordNet. Technical report, Princeton University, 1990.

[65] Daniel P. Miranker. TREAT:a better match algorithm for AI production systems. In *Proc. AAAI-87, 4th National Conference on Artificial Intelligence*, pp. 42–47, 1987.

[66] D. I. Moldovan, W. Lee, and C. Lin. SNAP: a maker-propagation architecture for knowledge processing. *IEEE Trans. Parallel and Distributed Systems*, Vol. 3, , 1992.

[67] Motorola. *VME Bus Specification Manual (Rev.C)*, 1985.

[68] A. Motro. Intentional answers to database queries. *IEEE Trans. Knowledge and Data Engineering*, Vol. 6, No. 3, pp. 444–454, 1994.

[69] S. Muggleton, editor. *Inductive Logic Programming*. Academic Press, 1992.

[70] U. A. Müller, A. Gunzinger, and W. Guggenbühl. Neural net simulation with MUSIC. In *Proc. IEEE International Joint Conference on Neural Networks*, pp. 1737–1741, 1993.

[71] 長野ゆかり, 松澤和光. プロダクションシステムの非同期実行方式. 電子情報通信学会全国大会, 1989.

[72] A. Newell. Production systems, models of control structures. In W. G. Chase, editor, *Visual Information Processing*, pp. 463–526. Academic Press, 1973.

[73] 西田豊明. 人工知能の基礎. 情報科学コアカリキュラム講座. 丸善, 1999.

[74] 奥川峻史. 並列計算機アーキテクチャ, 並列処理シリーズ, 第 2 巻. コロナ社, 1991.

[75] A. O. Oshisanwo and P. P. Dasiewicz. A parallel model and architecture for production systems. In *Proc. 16th Int. Conference on Parallel Processing*, pp. 147–153, 1987.

[76] C. S. Peirce. Elements of logic. In C. Hartshorne and P. Weiss, editors, *Collected Papers of Charles Sanders Peirce*, Vol. 2. Harvard University Press, 1932.

[77] D. Poole, R. Goebel, and R. Aletiunas. Theorist: A logical reasoning system for defaults and diagnosis. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier: Essays in the Representation of Knowledge*, pp. 331–352. Springer-Verlag, 1987.

[78] J. R. Quinlan. Induction of decision trees. *Machine Learning*, Vol. 1, No. 1, pp. 01–106, 1986.

[79] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, Vol. 13, pp. 81–132, 1986.

[80] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing*, Vol. 1. The MIT Press, 1987.

[81] 佐武一郎. 行列と行列式, 数学選書, 第 1 巻. 裳華房, 1958.

[82] M. I. Schor, T. P. Daly, H. S. Lee, and B. R. Tibbitts. Advances in Rete pattern matching. In *Proc. AAAI-86, 3rd National Conference on Artificial Intelligence*, pp. 226–232, 1986.

[83] F. Schreiner and G. Zimmermann. PESA–1 — a parallel architecture for production systems. In *Proc. 16th Int. Conference on Parallel Processing*, pp. 166–169, 1987.

[84] E. Y. Shapiro. Inductive inference of theories from facts. Technical Report 192, Yale University Computer Science Department, 1981.

[85] E. H. Shortliffe. Computer based medical consultations: MYCIN. *American Elsenier*, 1976.

[86] N. Sundararajan and P. Saratchandran. *Parallel Architectures for Artificial Neural Networks*. IEEE Computer Society, 1998.

[87] Y. Suzuki and L. E. Atlas. A study of regular architectures for digital implementation of neural networks. In *Proc. IEEE International Symposium on Circuits and Systems*, pp. 82–85, 1989.

[88] 高木幹雄, 下田陽久監修. 画像解析ハンドブック. 東京大学出版会, 1991.

[89] 高須達監訳. 数値解析概論. 日本評論社, 1970.

[90] Y. Takayama, R. Flournoy, S. Kaufmann, and S. Peters. Information retrieval based on domain-specific word associations. In *Proc. Pacific Association for Computational Linguistics 99*, 1999.

[91] K. Tanno, A. Tanaka, and T. Taketa. Parallel simulation algorithm of neural networks on massively parallel computers. In *Proc. Transputer Applications and Systems*, pp. 651–659, 1993.

[92] 富田眞治. 並列計算機構成論. 昭晃堂, 1986.

[93] 上野晴樹. 知識工学入門 (改訂第 2 版). オーム社, 1989.

[94] S. V. Vrbsky and J. W.-S. Liu. APPROXIMATE – a query processor that produces monotonically improving approximate answers. *IEEE Trans. Knowledge and Data Engineering*, Vol. 5, No. 6, pp. 1056–1068, 1993.

[95] P. H. Winston. Learning and reasoning by analogy. *Communications of the ACM*, Vol. 23, No. 12, pp. 689–703, 1980.

[96] M. Yasrebi and J. C. Browne. Analysis and design of parallel algorithms and implementations of matrix multiplications for image and signal processing. In *Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processsing*, pp. 44–47, 1989.

[97] 湯川高志, 石川勉. 負荷均等化を図った粗粒度 PS 並列化方式. 情報処理学会全国大会, pp. 1421 – 1422, 1989.

[98] 湯川高志. バス結合型プロセッサに適したバックプロパゲーションの並列計算法. 電子情報通信学会論文誌, Vol. J-74-D-II, pp. 1318 – 1320, 1991.

[99] T. Yukawa and T. Ishikawa. Optimal parallel back-propagation schemes for mesh-connected and bus-connected multiprocessors. In *Proc. IEEE International Joint Conference on Neural Networks*, pp. 1748–1753, 1993.

[100] T. Yukawa, T. Ishikawa, H. Kikuchi, and K. Matsuzawa. TWIN: a parallel scheme for a production system featuring both control and data parallelism.

In *Proc. 7th IEEE Conference on Artificial Intelligence Applications*, pp. 64–70, 1991.

[101] 日本電子化辞書研究所. ＥＤＲ電子化辞書 1.5 版 仕様説明書, 1996.

関連発表

学会論文誌

Hideo Kikuchi, Takashi Yukawa, Kazumitsu Matsuzawa, and Tsutomu Ishikawa. Presto: a bus-connected multiprocessor for a RETE-based production system. 電子情報通信学会論文誌, Vol. E75-D, No. 3, pp. 265–273, 1992.

湯川高志, 菊池英夫, 松澤和光, 石川勉. ローカルメモリ型並列プロセッサに適したプロダクションシステム並列化方式. 電子情報通信学会論文誌, Vol. J75-D-1, No. 8, pp. 694–703, 1992.

湯川高志, 石川勉. プロダクションシステム並列化の解析評価手法と TWIN 並列化方式の提案. 情報処理学会論文誌, Vol. 33, No. 12, pp. 1545–1554, 1992.

湯川高志, 石川勉. 連続行列演算の最適並列化とそれに基づく最適 BP 並列化方式. 電子情報通信学会論文誌, Vol. J78-D-I, No. 5, pp. 445–454, 1995.

Takashi Yukawa, Kaname Kasahara, and Kazumitsu Matsuzawa. High-speed similitude retrieval for a viewpoint-based similarity discrimination system. 電子情報通信学会論文誌, Vol. E80-D, No. 12, pp. 1215–1220, 1997.

学会論文誌 (研究速報，レター)

湯川高志. バス結合型プロセッサに適したバックプロパゲーションの並列計算法. 電子情報通信学会論文誌, Vol. J74-D-II, No. 9, pp. 1318–1320, 1991.

藤本和則, 湯川高志, 松澤和光, 石川勉. ルールの論理的構造に着目した結論の ”尤もらしさ ”の定量化. 情報処理学会論文誌, Vol. 36, No. 8, pp. 2071–2074, 1995.

Takashi Yukawa, Sanda M. Harabagiu, and Dan I. Moldovan. Viewpoint-based similarity discernment on SNAP. 電子情報通信学会論文誌, Vol. E82-D, No. 2, pp. 500–502, 1999.

国際会議

Hideo Kikuchi, Takashi Yukawa, Kazumitsu Matsuzawa, and Tsutomu Ishikawa. Presto: a bus-connected multiprocessor for a RETE-based production system. In *Joint Conference of CONPAR90/VAPP IV*, pp. 63–74, 1990.

Takashi Yukawa, Tsutomu Ishikawa, Hideo Kikuchi, and Kazumitsu Matsuzawa. TWIN: a parallel scheme for a production systems featuring both control and data parallelism. In *Proc. 7th IEEE Conference on Artificial Intelligence Applications*, pp. 62–70, 1991.

Takashi Yukawa and Tsutomu Ishikawa. Optimal parallel back-propagation schemes for mesh-connected and bus-connected multiprocessors. In *Proc. IEEE International Joint Conference on Neural Networks*, pp. 1748–1753, 1993.

Takashi Yukawa, Kaname Kasahara, and Kazumitsu Matsuzawa. High-speed similitude retrieval for a viewpoint-based similarity discrimination system. In *Florida AI Research Symposium*, pp. 261–265, 1997.

研究報告

菊地 英夫, 湯川 高志, 松澤 和光, 石川 勉. 並列処理を利用したKBMS高速化装置 (Presto). NTT R&D, Vol. 39, No. 3, pp. 437–445, 1990.

水書 章雄, 湯川 高志, 菊地 英夫. 並列プロセッサ制御プログラムの一検討. 研究開発資料, 日本電信電話株式会社, 1990.

Sanda M. Harabagiu, Takashi Yukawa and Dan I. Moldovan. Marker-Propagation Applications for Text Inference. Technical report, Department of Computer Science and Engineering, Southern Methodist Unicersity, 1995.

大会・研究会等

湯川高志, 松澤和光, 石川勉. Rete ネットモデル化による PS 並列化の評価. 情報処理学会 第 36 回全国大会, 1988.

湯川高志, 石川勉. 負荷均等化を図った粗粒度 PS 並列化方式. 情報処理学会 第 37 回全国大会, 1988.

湯川高志, 松澤和光, 石川勉. PS 並列化の速度向上評価と負荷均等化粗粒度方式の提案. 情報処理学会 知識工学と人工知能研究会, 1989.

湯川高志, 菊地英夫, 松澤和光, 石川勉. 粗粒度並列化方式を用いたプロダクションシステム推論アクセラレータ. 人工知能学会 全国大会, 1990.

湯川高志, 松澤和光, 桃井茂晴. アバウト推論における多階層推論方式. 情報処理学会 第 45 回全国大会, 1992.

湯川高志, 笠原要, 松澤和光, 石川勉. アバウト推論: 多観点概念ベースにおける類似概念検索の高速化. 情報処理学会 第 47 回全国大会, 1993.

湯川高志, 笠原要. 類似概念の高速検索手法. 情報処理学会 第 49 回全国大会, 1994.

湯川高志. SNAP を用いた概念の類似判別. 情報処理学会 第 53 回全国大会, 1996.