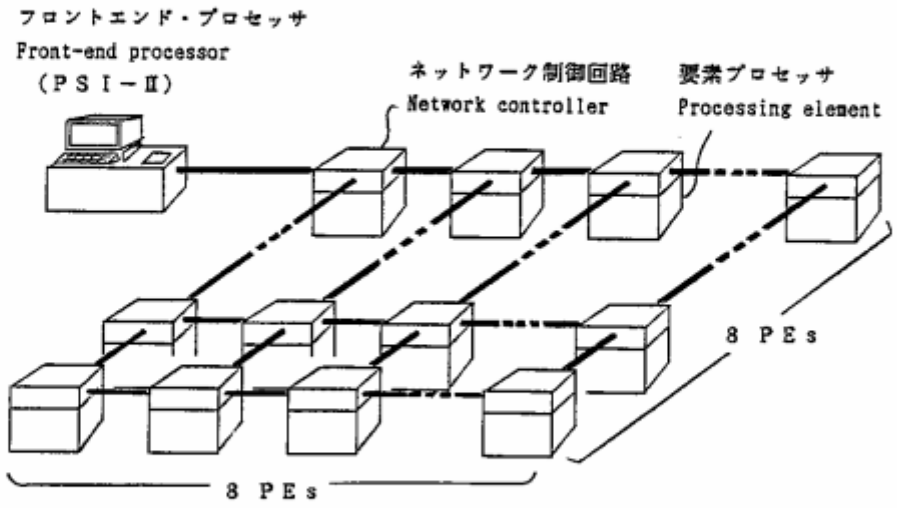
題　名	**並列ソフトウェア開発システム**
目　的	並列マシンの上に論理型言語の処理系を実装し，並列オペレーティングシステム，並列処理のアルゴリズム，および負荷分散方式を研究開発する。
概要及び特徴	• 逐次型推論マシン PSI の CPU を，専用のハードウェアで最大 64 台接続したマルチ PSI を開発 • 並列論理型言語 (KL1) のプログラムを高速に処理する分散処理系を実現 • 並列処理に適したオペレーティングシステム (PIMOS) を開発し，その管理下で並列ソフトウェアを研究開発 • 並列推論マシン (PIM) に向けて，KL1 の処理系や PIMOS を改良拡張するためのツールとして利用
構　成	並列ソフトウェア開発システム 

題　名	並列ソフトウェア開発用ハードウェア ― マルチ PSI
目　的	並列ソフトウェアの研究開発，および，並列分散処理機構の研究開発のための高い処理能力を提供する。並列推論マシンＰＩＭのアーキテクチャ研究へのフィードバックの役割も果たす。
概要及び特徴	ＣＭＯＳゲートアレイＬＳＩなどで小型化した逐次型推論マシンＰＳＩのＣＰＵを要素プロセッサ（ＰＥ）とし，それらをメッセージ交換・自動ルーティング機能を持つ専用のネットワーク制御機構で二次元メッシュ状に接続している。８ＰＥずつ最大６４ＰＥ（８ｘ８）まで接続できる。 入出力機能を提供するフロントエンド・プロセッサとして，小型化版のＰＳＩであるＰＳＩ－Ⅱを最大４台まで接続できる。 タグ・アーキテクチャ：８ ビットタグ ＋ ３２ ビットデータ 要素プロセッサ制御　：水平型マイクロプログラム方式 サイクルタイム　　：２００ ｎｓｅｃ （全系同期クロック） 主記憶　　：各ＰＥ ８０ ＭＢ（１６ ＭＷ） ネットワークチャネル：各チャネル双方向 ５ ＭＢ／ｓ 使用素子　　：８Ｋ，２０Ｋゲート ＣＭＯＳゲートアレイ 等
構　成	

デ　　モ　　内　　容　（1／2）

要素プロセッサ（ＰＥ）

水平型マイクロプログラム方式（５３ビット長のマイクロ命令）のため，ＫＬ１処理系の試作などに柔軟に対処できる。また，論理型言語の高速実行を目的としてタグ・アーキテクチャを採用しているため，十分に高い性能を発揮できる。特に，タグ操作やタグによる多方向分岐機能などをＡＬＵ演算と同時に制御できるため，高レベル抽象機械語命令の直接解釈実行に向いている。

ネットワーク制御回路

隣接した４つのＰＥと双方向１バイト幅のチャネルでメッセージ通信する。受信したメッセージ・データは，自ノードのＰＥのためのバッファ（リードバッファと称する）に取り込むか，もしくは適当なチャネルを選択して転送する。この処理は，メッセージの先頭に指定された行き先ＰＥ番号で転送チャネル選択用のテーブル（パス・テーブルと称する）を引くことによって行われる。

各チャネルの送出側には４８バイトのバッファ（出力バッファと称する）を設けており，送出方向のノードが Busy 状態の場合の待ち合わせのために用いる。
リードバッファにメッセージが取り込まれると，ＮＷＩＮＴ信号でＰＥに知らされ，ＰＥは内部処理の適当な切れ目でメッセージ単位にそれを処理する。
ＰＥからのメッセージ送出は，完全な形のメッセージを送出用バッファ（ライトバッファと称する）に書き込むことにより，自動的に開始される。

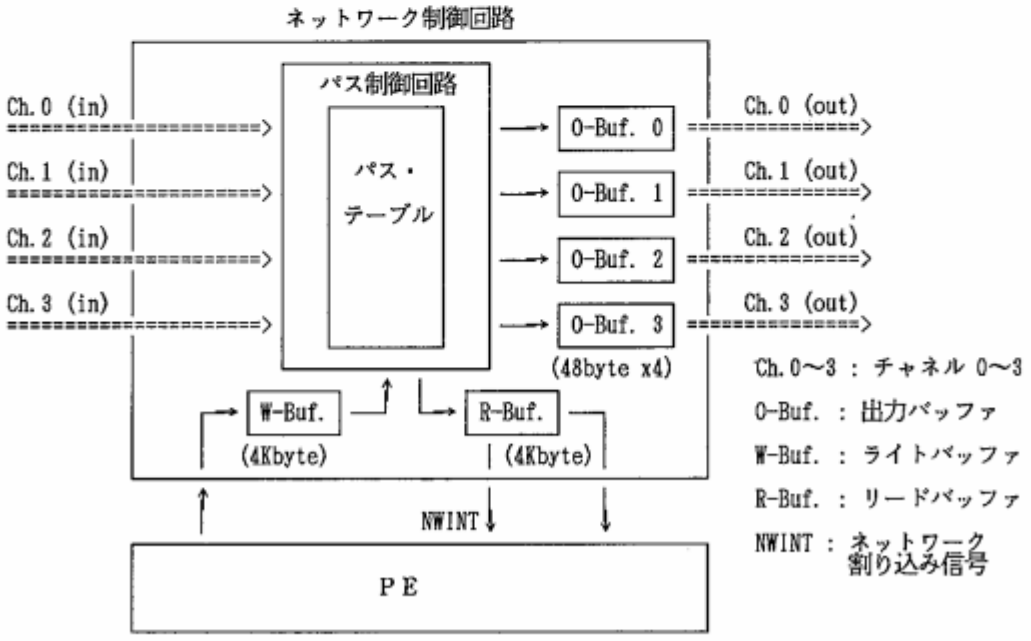


図.　ネットワーク制御回路と要素プロセッサ（ＰＥ）

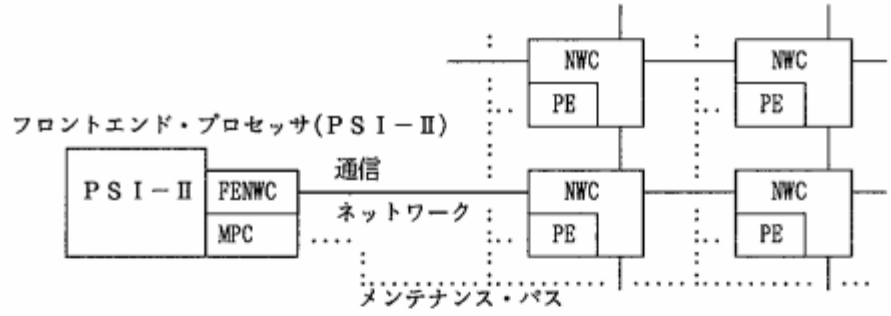
デ　　モ　　内　　容　（2／2）

実装

要素プロセッサは８ＫゲートのＣＭＯＳゲートアレイ９石などを用いプリント基板３枚に，ネットワーク制御回路は２０ＫゲートのＣＭＯＳゲートアレイ２石を中心にプリント基板１枚に実装されている。
主記憶は１Ｍビットのダイナミック ＲＡＭを用い，最大で８０Ｍバイト（２０Ｍバイトプリント基板ｘ４枚）まで実装できる。

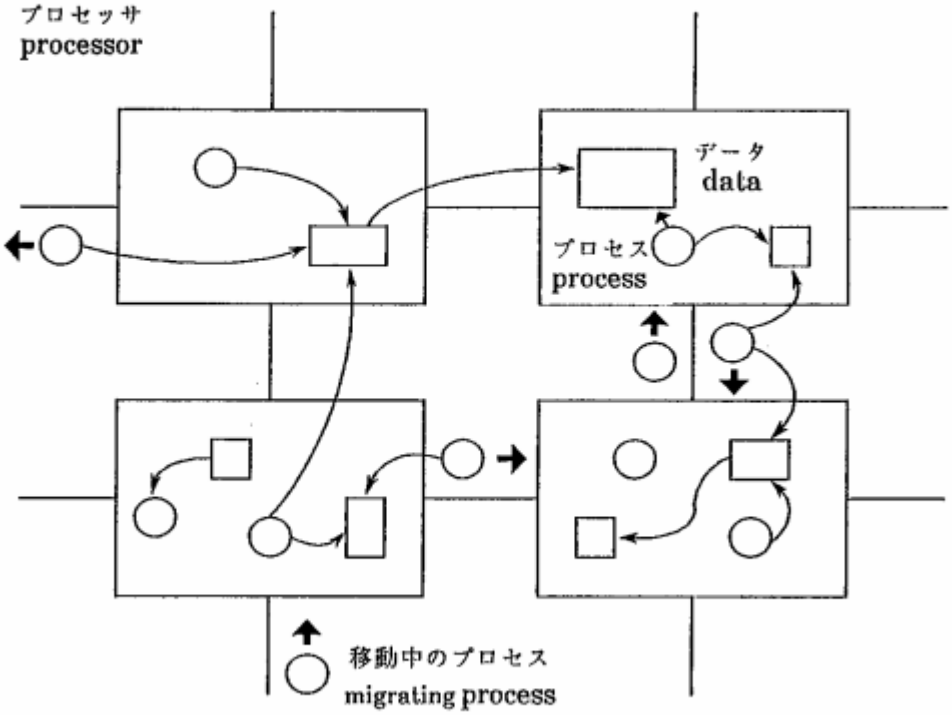
マルチＰＳＩの筐体には８つの要素プロセッサ（プリント基板最大８枚／ＰＥ）が格納されており，８筐体接続で最大の６４ＰＥ構成となる。クロックは一箇所から供給され，システム全体が同相のクロックで動作する。

入出力装置の機能を提供するフロントエンド・プロセッサ（ＦＥＰ）もネットワークに直接接続される。ＦＥＰは４台まで接続できるが，その内の１台はマスタＦＥＰとなり，システムの立ち上げや診断を行うコンソール・プロセッサ（ＣＳＰ）を兼ねる。システムの立ち上げや診断は，通常のネットワークとは別に各ＰＥに接続された，メンテナンス用のパスが用いられる。



PE　　：要素プロセッサ
NWC　：ネットワーク制御回路
FENWC：フロントエンド・ネットワーク制御回路
MPC　：メンテナンス・パス制御回路

図．　マルチＰＳＩのフロントエンド・プロセッサと本体部

題　名	**KL1 分散処理系**
目　的	ネットワーク結合型並列マシン「マルチ PSI」のプロセッサ上に分散した KL1 のプロセスとデータを管理し、プログラムを効率良く実行する。
概要及び特徴	KL1 プログラムは最適化コンパイラにより抽象機械命令に落とされ、マイクロコードで書かれた処理系で実行される (プロセッサ当たり append で 150KLIPS の性能)。処理系は KL1 の持つ単一代入性の利用などにより、プロセッサ間通信量を減らし、分散したプロセスおよびデータを効率よく管理する。新方式の局所 GC およびプロセッサ間 GC が実装されている。
構　成	

デ　　モ　　内　　容　（1／1）

1　KL1

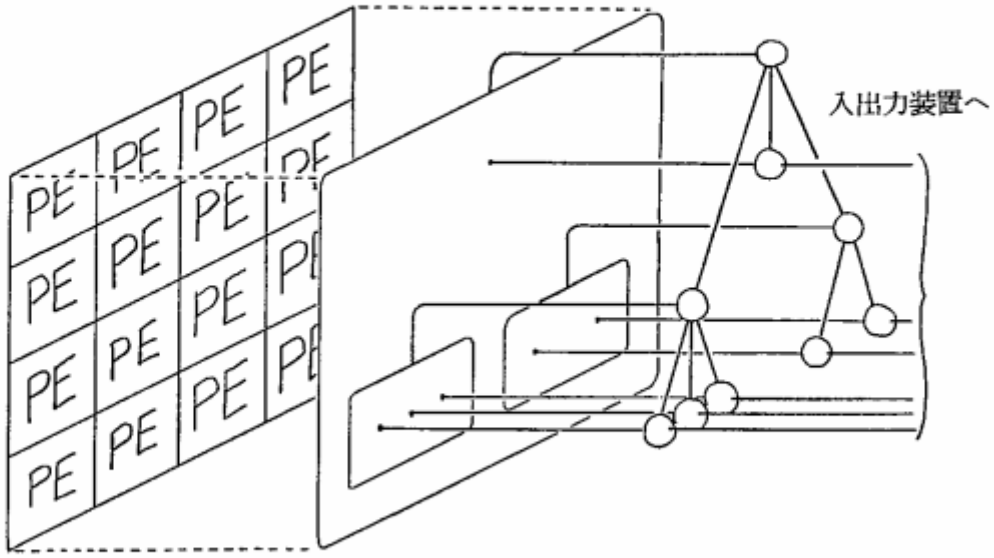
KL1 はストリーム AND 並列論理型言語である。ストリーム AND 並列では、並列に実行されるプロセスがデータを共有し合い、あるプロセスの出力を別のプロセスが使うというデータ因果関係の連鎖としてプロセス構造が構成される。ストリーム型 AND 並列は、OR 並列やデータ共有のない AND 並列と違い、逐次的に書いたプログラムをそのままでは並列に実行できないが、複数のプロセスが協力しあって問題を解く複雑な協調問題解決に向く計算方式である。

2　マルチ PSI のアーキテクチャ

並列計算機には大きく分けて、メモリ共有型とメモリ非共有型がある。前者では共有メモリに情報を書いたりそれを読んだりすることによってプロセッサが情報を伝達し合うが、後者では通信路を介してプロセッサが情報を伝達し合う。メモリ共有型並列計算機は、プロセッサ間の情報伝達オーバヘッドは比較的小さいが、メモリアクセスがネックになるので、プロセッサ台数の上限が小さいという問題がある。マルチ PSI は、将来的に要素プロセッサ 1000 台程度の並列計算機を目指す PIM プロジェクトの実験機でもあるので、メモリ非共有型のアーキテクチャを採用している。ただし、メモリ非共有型マシン上のプログラムは通信オーバヘッドを念頭において設計されなければならない。

3　KL1 分散処理系

マルチ PSI 上で KL1 プログラムを効率良く実行するのが KL1 処理系の役割である。この処理系では、プロセッサ間の通信量を減らすようなプロトコルが工夫されており、また、KL1 はメモリ消費速度が早いので、ガーベジ・コレクション (GC) (不要となったデータの占有するメモリの回収) に新しい方法が採用されている。すなわち、プロセッサ内の 1 ビット・ポインタタグにより回収可能なデータを認識する MRB (Multiple Reference Bit) 技法、および WEC (Weighted Export Counting) というプロセッサ間関時 GC 技法である。また、資源・タスク管理、優先度管理、ユーザ指定による負荷分散などのメタプログラミング機能により OS 基本機能をサポートしているのも KL1 処理系の特徴である。

題　名	並列推論マシン・オペレーティングシステム: PIMOS
目　的	並列推論マシンの処理能力を、応用プログラムから容易にかつ効率良く利用するためのオペレーティングシステム機能を提供する。
概要及び特徴	**KL1 で記述:** PIMOS はすべて並列論理型言語 KL1 で記述している。KL1 の提供するメタ・プログラミング機能を活用して、要素プロセッサ数などのハードウェアに依存しないように設計した。 **複数プロセッサ上の単一 OS:** PIMOS はプロセッサごとに独立した OS 群の集合体ではなく、単一の OS である。しかし、応用プログラムのみならず OS 自身の実行も複数のプロセッサ上で並列に行う。管理に必要な情報を持った管理プロセスを管理対象の応用プログラムの近くに分散させることにより、集中管理した場合に生じる処理や通信のボトルネックを解消している。 **基本機能の提供:** 実行管理・資源管理・入出力管理などの OS としての基本機能を提供している。種々の付加的なサービスは今後充実していく予定である。 マルチ PSI 上で実行するデモンストレーションは、すべて PIMOS の管理下でのものである。
構　成	並列推論マシン　応用プログラムタスク　PIMOS の管理プロセス 

デ　　モ　　内　　容　（1／2）

デモンストレーションでは、管理機能を実際に利用する応用プログラムの実行制御を通じて、PIMOS の機能を紹介する。

ストリーム通信とフィルター

PIMOS と応用プログラムとの間の通信はすべて AND 並列論理型言語の特長であるストリーム通信を用いている。PIMOS はこの通信ストリームの間に適当にフィルターを挿入することによって、応用プログラムの異常終了時の後始末処理や、オペレーティングシステムの保護などを行う。

資源木による分散処理

PIMOS の管理下で実行する応用プログラムのタスク群と、それらのタスクで用いている入出力装置などの資源には、それぞれ対応する PIMOS の管理プロセスが付随している。『プロセス』といっても、KL1 のプロセスは KL1 言語処理系のマイクロコードで実現するごく軽いもので、オブジェクト指向言語でいう『オブジェクト』に対応するようなものである。

PIMOS ではタスクの中にタスクを入れ子状に作ることができるので、応用プログラムのタスク群は全体として木構造になる。これに対応して PIMOS の管理プロセス群もひとつの木構造を形作る。PIMOS ではこれを『資源木』と呼んでいる。PIMOS の持つ管理情報はすべて、この資源木の各ノードである管理プロセスに分散して置かれている。

管理プロセスは、管理対象の応用プログラムタスクと同じプロセッサ (応用プログラムを複数のプロセッサ上で実行する場合は、タスク生成や入出力装置の利用の要求を出したプロセッサ) に割り付ける。このような割り付けには、管理負荷を多数のプロセッサに分散する効果がある。また、管理情報を対象とする応用プログラムのそばに置くことによって、ひとつのテーブルなどに情報を集中した時のような通信の偏りを防ぎ、プロセッサ間の通信量を減らしている。

デ　　モ　　内　　容　（2／2）

シェル

PIMOS の管理下で応用プログラムを実行する時には PIMOS で用意するコマンドインタプリタ (シェル) から起動する。シェルは PIMOS の提供する基本機能を利用して、以下のような機能をユーザに提供している。

- ジョブの起動 / 中断 / 再起動 / 実行放棄
- フォアグラウンド・ジョブ / バックグラウンド・ジョブの制御
- 標準入出力の設定
- パイプによるタスク間通信の制御
- ジョブへの計算資源割当て量の制御
- 例外事象の処理

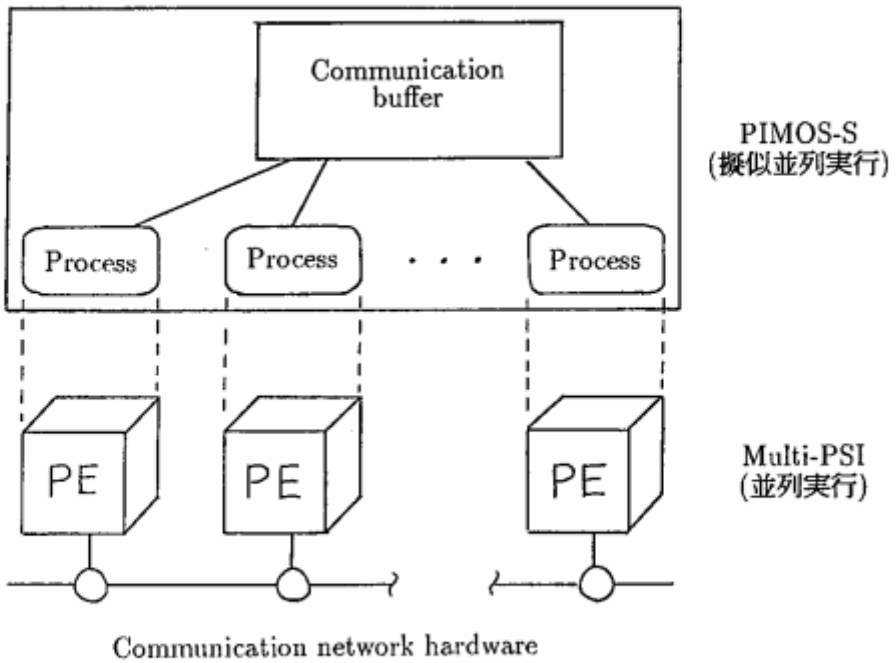
また、ユーティリティプログラムにより、タスクの実行状態や入出力装置などの資源の利用状況の問合せができる。

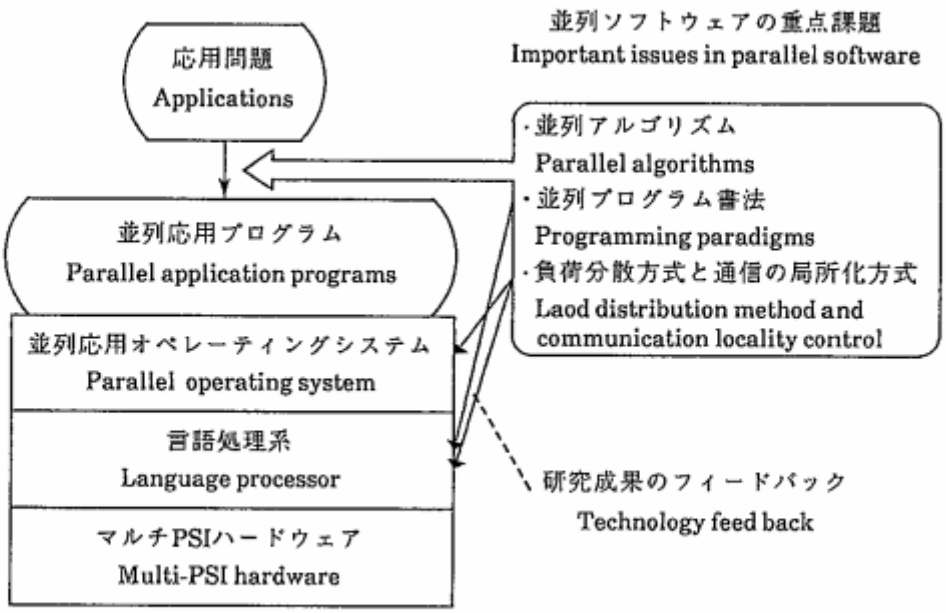
入出力装置

現在の PIMOS は、フロント・エンド・プロセッサ (FEP) の PSI-II 上のオペレーティングシステム SIMPOS の提供するファイル、ウインドウなどの高機能入出力に KL1 プログラムの中からアクセスする機能を提供している。種々のデモンストレーション・プログラムで用いている表示 / 入力用のウインドウも、FEP 上に用意したものを PIMOS の提供する統一的なインタフェースで利用しているものである。

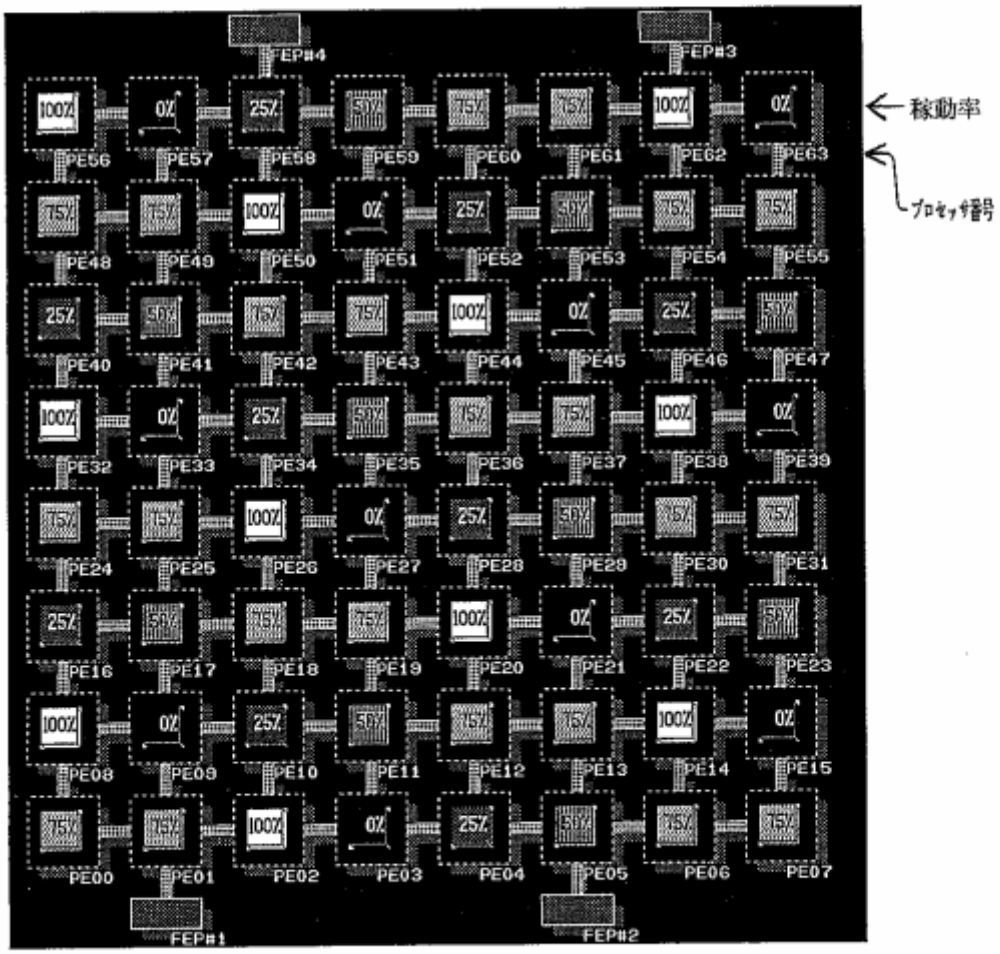
なお、PIMOS の設計の詳細については、11 月 30 日 (水) 午後 3:30 からの ICOT-SS1 セッションで発表する以下の論文にまとめられている。

"Overview of the Parallel Inference Machine Operating System"

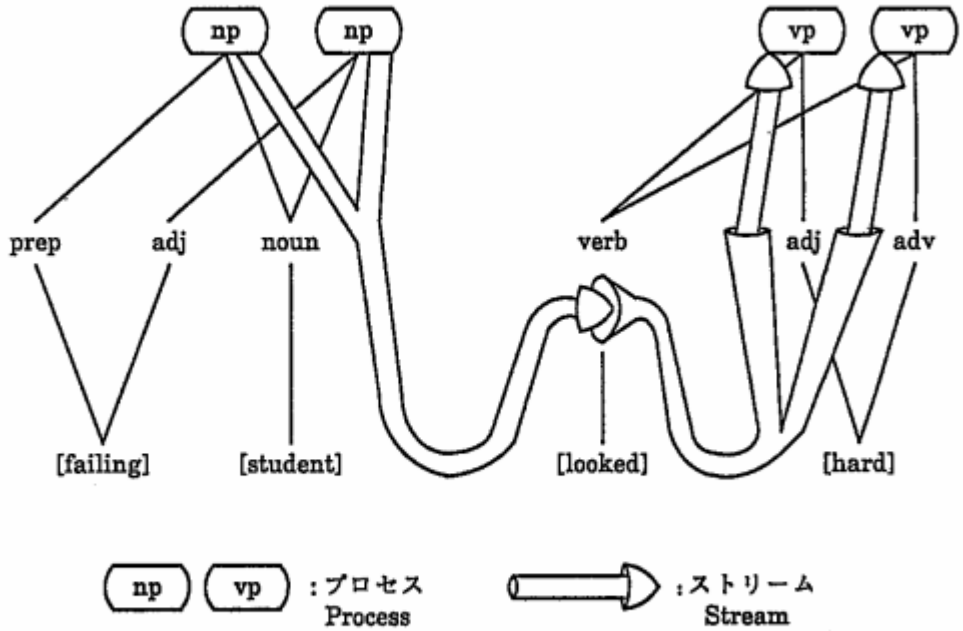
題　名	**PSI-II 上の並列ソフトウェア開発環境: PIMOS-S**
目　的	マルチ PSI システムと質的に同等な並列ソフトウェア開発環境を、広範に利用可能なワークステーション PSI-II の上に提供する。
概要及び特徴	**互換性:** マルチ PSI システムと完全互換の言語・OS 機能を提供する。PSI-II ワークステーション上で PIMOS-S を用いて開発した並列ソフトウェアは、そのままマルチ PSI 上で実行できる。 **擬似並列:** 並列動作するマルチ PSI のプロセッサをソフトウェア管理のプロセスで模擬して、擬似並列実行する。 **高い処理能力:** マルチ PSI と同一のマイクロコードを利用し、マルチ PSI の要素プロセッサ 1 台分と同等の処理能力を持つ。 **良好なデバッグ環境:** マルチ PSI システムと質的に同等のデバッグ環境を提供する。擬似乱数スケジューリングにより実行過程の再現性を保証できることなど、マルチ PSI より有利な点もある。
構　成	

題　名	並列ソフトウェア研究
目　的	ネットワーク結合型の大規模並列マシンを幅広い応用に対して効率良く動かすために不可欠となる、新しいソフトウェア技術体系を構築する。
概要及び特徴	大規模並列マシンを効率良く並列動作させることはたやすくない。このためには応用問題のプログラム化と実行の過程で使われる技法のほとんどを並列向きに改める必要がある。なかでも重要な項目として次のものが上げられる。 ● 並列度が高く計算量の少ないアルゴリズムの開発 ● タイプの異なる各種の問題、大規模な問題を並列プログラム化できるためのプログラミング・スタイルの蓄積 ● 各プロセッサに対する均等な負荷分配と、部分問題間の通信の削減または通信の局所化に関する方式の確立 われわれは各種応用問題の並列プログラム化（例：デモ用プログラム）を題材としてこれらの研究を開始している。また研究成果によりＯＳや言語処理系の機能を改良してゆく。
構　成	

題　名	マルチＰＳＩ用稼動率測定表示プログラム ―パフォーマンス・メータ―
目　的	マルチＰＳＩの各要素プロセッサの稼働率を一定時間（標準２秒）毎に実時間表示する。 ユーザプログラムの負荷分散の良否判定に使用する。
概要 及び 特徴	測定にはＫＬ１で書いたプログラムを使用している。 測定オーバヘッドは小さい。
構　成	測定プログラムは各々のプロセッサに分散された計測プロセスと、計測結果を集める集計プロセス等から構成される。結果はＰＩＭＯＳ経由でＦＥＰ上の表示デバイスに送られ表示される。



パフォーマンスメータの画面表示例

題　名	並列応用プログラム (1)　自然言語構文解析
目　的	ＤＵＡＬＳで使用されている自然言語の構文解析をレイヤード・ストリーム手法を用いて並列化し、その手法に適した負荷分散方式を考案し、実現／評価することにより、構文解析の高速化を図る。
概要及び特徴	【処理概要】複数の文章の構文を解析し、解析木を生成する。構文解析は、解析木の各ノードに対応するプロセス間で通信しながら、ボトム・アップに行われる。 【解析プログラム】拡張文脈自由文法で定義された文法から、トランスレータを用いてＫＬ１の解析プログラムを生成し、負荷分散指定を自動的に付加する。 【問題の性質】処理の過程で複数の解釈が可能な場合、総ての解釈について解析する（全解探索）。文法定義中に、ＫＬ１プログラムの呼出し（補強項）が指定されており、解釈の枝刈りが行われる。 【プログラム手法】全解探索を行う１つのパラダイムである、レイヤード・ストリーム手法を用いて、解析木の各ノードに対応するプロセス間で通信しながら、解釈を行っていく。 【負荷分散方式】レイヤード・ストリーム法は、一般的に通信量が多いため、通信量を極力押さえるような負荷分散方式を採用している。
構　成	 解析木と階層化されたストリーム通信 Parse tree and layered stream communication

デ　　モ　　内　　容　(1/3)

処理概要

構成

PAX は、自然言語の文を文法に従って解析し (構文解析)、解析結果を木状にしてウインドウに表示するシステムであり、文の入力部, 構文解析部, 結果の出力部から構成される。

このうち構文解析部が PAX の中心となる部分であり、マルチ PSI 本体上で並列に実行される。この構文解析部のプログラムは、拡張文脈自由文法により記述された文法からトランスレータを用いて生成される。

また、文の入力部及び結果の出力部は、 ESP で記述されており、フロントエンドの PSI-II 上で実行される。

問題の性質

構文解析には、左隅構文解析法といわれる上昇型幅優先のアルゴリズムを用いている。一般に構文解析処理では、全ての可能性を試みる「全解探索」 が行なわれ、特に自然言語の文法は曖昧さを含むため、解析結果が一意であるとは限らない。

バックトラック機能を持たない KL1 のような言語で全解探索問題を記述するには、はじめに適用可能なすべての候補を集め、順次ふるっていくという方法を用いるのが一般的である。 PAX では**レイヤードストリーム法**と呼ばれる汎用的な全解探索のためのプログラミング手法を用いている。

処理方式

PAX は、読み込んだ文を単語に分け、それを基に解析木の各ノードに相当するプロセスをボトムアップに生成しながら、構文解析を行っていく。解析処理は、兄弟ノードのプロセス間で、兄から弟へ (左のノードから右のノードへ) 解析の途中結果 (部分解析木) をストリームを介して送ることにより進められる。

具体的に構成図に示されている例に基づいて説明する。この例は、

"failing student looked hard"

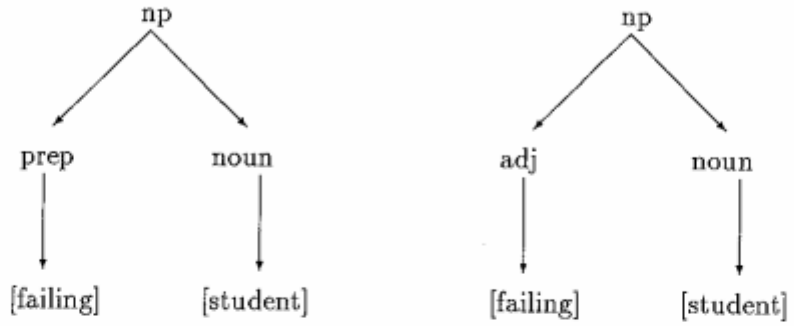
という文を以下のような文法に従って解析している。

```
sentence --> np, vp.          adj  --> [failing].
np       --> adj, noun.       adj  --> [hard].
np       --> prep, noun.      prep --> [failing].
vp       --> verb, adj.       adv  --> [hard].
vp       --> verb, adv.       verb --> [looked].
                              noun --> [student].
```

デ　モ　内　容（2/3）

まず最初に、読み込まれた単語に対するプロセスが、単語間でストリームを張って、起動される。

この図は、既にnp(名詞句) 及びvp(動詞句) まで解析が行われ、それぞれのnpから以下のような部分解析木をvpへストリームを介して送ろうとしているところである。



np及び**vp**が２つづつ存在するのは、*failing* に対して**adj**(形容詞) と**prep**(前置詞) の２つの候補があり、*hard* に対しても **adj**(形容詞) と**adv**(形容動詞) の２つの候補があるからである。

npと**vp**の通信は、以下のように最初に単語間に張られたストリームを階層的に使うことによって行われる。

1. それぞれの**np**は、*student – looked* 間に張られたストリームにそれぞれの部分解析木を送る。
2. **verb**と**adj**及び**adv**も兄弟関係にあるので*looked – hard* 間のストリームを介して通信するが、その時メッセージの中に*student – looked* 間のストリームを入れる。
3. **adj**及び**adv**は**verb**からのメッセージを受け取ると、親ノードに相当するプロセス**vp**を起動し、受け取ったメッセージの中に入っている*student – looked* 間のストリームを渡す。
4. それぞれの**vp**は、*student – looked* 間のストリームを渡され、２つの**np**からのメッセージを受け取ることができる。

このようにして、**np**と**vp**間で通信が行われるが、それぞれの**vp**は２つの**np**からの部分解析木を受け取ると、自分が解析した部分解析木をそれに付加して、親ノード**sentence**を生成し、解析処理を終了する。これにより、この文に対して４種類の構文が適用可能であることがわかる。

デ　　モ　　内　　容　（3／3）

負荷分散方式

PAXでは、以下のような2種類の並列実行が考えられる。

1. 兄弟ノードを形成するまでの処理を並列に実行する。上記の例では、**np**を得るまでと**vp**を得るまでの処理をそれぞれ並列に行える。

2. 複数の候補が考えられるときにそれぞれを並列に実行する。上記の例では、*failing*から**prep**と**adj**が考えられるが、それぞれの処理を並列に行える。

このような並列実行可能な部分をすべて別プロセサに分散させてしまうと、複数の候補からのメッセージが1つのストリームにマージされ、それがまた複数の候補に送られるので、N対1対Mのプロセサ間通信となり、通信量が多くなり過ぎる。

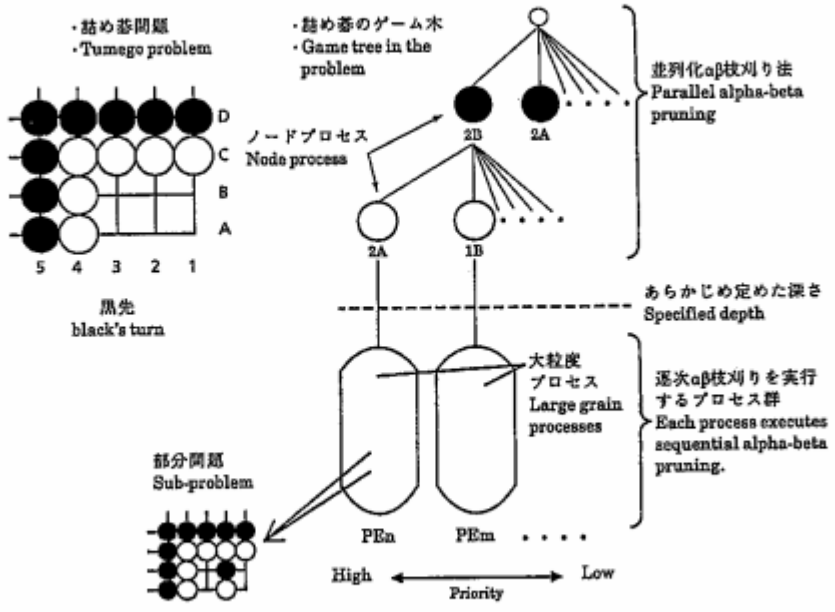
そこで、プロセサ間の通信量を極力抑える1つの方式として、同じストリームからのメッセージを受け取るプロセスは、1つのプロセサで実行するような負荷分散方式を採用した。

まず、最初に起動される各単語に相当するプロセスを異なったプロセサに分散する。そして、単語間に張られているストリームの先頭に、その左の単語が最初に起動されたプロセサの番号を入れる。このストリームを読み込むプロセスは、ストリームの先頭で指定されたプロセサに移動する。これにより、プロセサ間の通信を、N対1に抑えることができる。

デモンストレーションの内容

PAXのデモンストレーションでは、英英辞書“Oxford Advanced Learner's Dictionary of Current English”に記載されている文法に従って、いくつかの例文を構文解析し、以下のような項目をお見せする。

- 解析の結果をウインドウに表示する。
- 解析中の負荷分散の状況をパフォーマンス・メータを用いて示す。
- 単語数の異なる文を解析し、処理時間及び負荷分散状況の変化を示す。
- 複数の文を並列に解析する。
- 1つの文を使用するプロセサの台数を変えて解析し、解析時間を比べることにより、台数効果を示す。
- 逐次版で構文解析を行った場合との解析時間の比較を示す。

題　名	並列応用プログラム（２）　詰め碁
目　的	詰め碁の問題を解くプログラムを例にして、ゲーム木探索向きの並列アルゴリズム，プログラミングパラダイム，及び負荷分散方式についての試作とその評価を行う。
概要及び特徴	与えられた詰め碁問題に対して、その結果（活き／死に／コウ）を判定し、１手目を打着する。 問題の性質　：ゲーム木の完全（読み切り）探索 アルゴリズム：並列化したαβ枝刈り法を使用。探索木を並列展開し、プライオリティを付けて実行。 負荷分散方式：プライオリティ付き大粒度プロセスのランダム配置とアイドルプロセッサへの配置の２通り
構　成	

デ　　モ　　内　　容　（1／2）

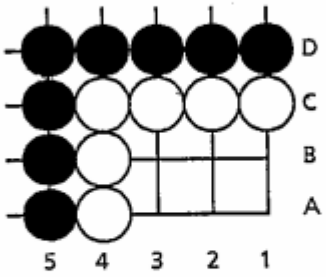
詰め碁問題

右図のように盤面の石の配置と手番（黒番または白番）とが与えられた時、囲まれている石の死活またはコウを判定する問題である。本プログラムでは判定とともに、その探索結果にもとづいて１手目を打着している。

死活を判定するために、ミニマクス型のゲーム木の探索を行い、探索枝の刈り込みにはアルファ・ベータ枝刈り法を並列化したものを用いている。

ゲーム木の探索では、評価関数を使い探索の深さを制限するものがしばしばあるが、ここでは可能な候補手を終端ノードまで探索している。

・詰め碁問題
・Tumego problem

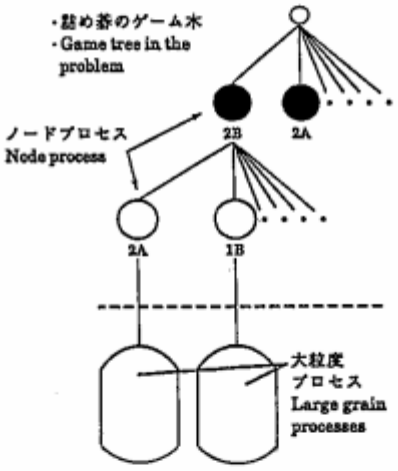


黒先
black's turn

アルゴリズムの概要

右図のようにゲーム木の予め定めた深さまでは並列化αβ枝刈り法を用い、そこから下は逐次型のαβ枝刈り法を実行する。逐次型の部分は各々が大粒度のプロセスとなっており、これらのプロセス単位にプロセッサへの負荷分散を行っている。

白番、黒番の各々の候補手は、”有望な手”の順にソートしたものを用いている。右の図では、探索木の各々の深さにおける左端のノードが最良の手に対応している。このような探索木をordered treeと呼ぶことがある。



ノードにおける処理

①与えられた盤面に候補手を打着する。この際アゲ石があればそれを盤面から取り除く。アゲ石の判定は隣り合う同色の石を全てたどっていき、それらの石の周囲が全て敵の石かを調べる必要があるので、αβ枝刈り法の判定処理に比べ重い処理となる。

②更新された盤面が終端ノードに達したかの判定をする。終端ノードの判定には標的となっている石がアタリ（後１手で取られる状態）かを調べる必要があるが、これも①と同様重い処理になる。

③終端ノードに達していなければ、並列化αβ枝刈りのためのプロセスと子ノードを生成し探索を続ける。

デ　　モ　　内　　容　（2／2）

並列化アルファ・ベータ枝刈り法

ordered treeのためのαβ枝刈り法を次のように並列化した。

通常のαβ枝刈り法では、深さ優先の縦型探索を行うとともに、子ノードから報告される評価値によって、残りの子ノードを探索するかどうかの枝刈りの検査を行っているため、高い逐次性を有する。

これに対して、今回の並列化αβ枝刈り法では、木を並列に展開していくとともに、各ノードには枝刈りの検査及び枝刈りによる処理の中止を関係ノードに知らせるプロセスを配置した。親子関係にあるプロセス間はストリームを介して情報を交換し合う。これにより各ノードは兄弟ノードの結果を待つことなく、並列に処理が実行できるようになった。一方逐次実行に比べて計算量を大きく増加させないためには適切な枝刈りが必須である。そこで探索枝の左側ほど有望な候補である特徴を生かして、効率の良い枝刈りを行うため、探索木の深さによらず常に左側のノードのプロセスほど高い実行優先度を持つようなpriority制御を使用している。これにより、同一プロセッサに割り付けられた複数のノードに対しては、より左側のノードから実行される。

負荷分散方式

大粒度プロセス（逐次αβ）のプロセッサへの配置について、以下の2通りの方式を試作した。

①準静的負荷分散

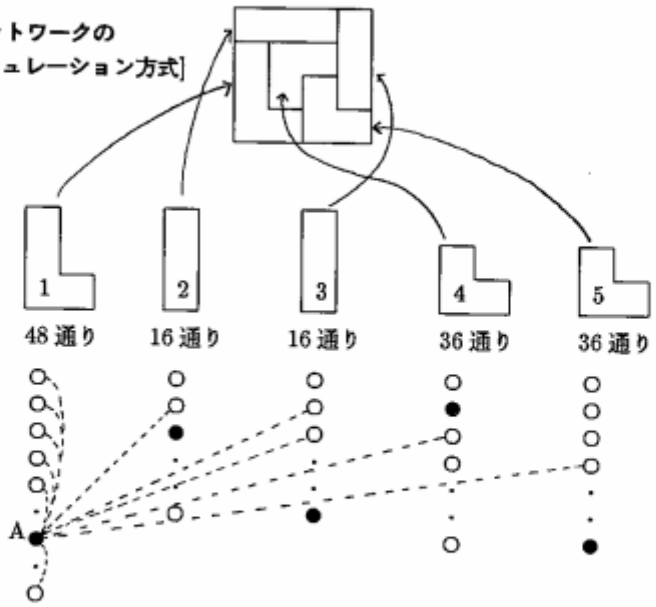
ランダムもしくは巡回的に各プロセッサに配置する。実行時の各プロセッサの負荷情報は使用しない。

②動的負荷分散

アイドル状態のプロセッサに配置する。アイドル状態は、各プロセッサ毎に最低プライオリティのプロセスを生成しておき、そのプロセスからのメッセージの到着によって認識している。

デモの概要

・詰め碁問題をメニューから選択する。
・負荷分散の方式を選択する。
・負荷分散のパラメタとPE台数を入力し探索を開始する。
・負荷分散の様子はパフォーマンスメータにより観察できる。
・探索結果と最初の1手が表示される。
・PE台数を変えて実行することにより、処理性能の台数効果がわかる。
・その他探索手順の逐次表示や、人の打った手のつづきから探索を開始することなどが可能である。

題　名	**並列応用プログラム (3):　　詰め込みパズル**
目　的	詰め込みパズルを異なる 2 通りの方法で解くことにより、それぞれに適する並列アルゴリズム、プログラミング・パラダイム、負荷分散方式について試作し、評価を行なう。
概要及び特徴	詰め込みパズルは、様々な形をしたピースをケースに詰め込むゲームである。詰め込む方法は通常何通りもあり、例えばペントミノと呼ばれる 12 ピースからなるパズルでは 18,712 通りの解がある。これらのパズルを異なる 2 つの方式で解く。 1. **全解探索方式** **プログラム手法:** OR 並列型全解探索を木状にフォークするプロセスで実現 **負荷分散方式:** OR 並列向き準静的負荷分散 2. **ニューラル・ネットワーク*のシミュレーション方式** **プログラム手法:** ニューロンに対応するプロセスを生成し、プロセス間通信でシミュレーションを実現 **負荷分散方式:** 通信主体の問題向き負荷分散方式を試行 *ガウシアン・マシン方式を使用 (協力: 慶応大学安西研究室、相磯研究室)
構　成	**詰め込みパズル [ニューラル・ネットワークのシミュレーション方式]**  ○ ●はニューロンであり、各ピース毎の全ての可能なケースへの置き方に対応する。 --- はニューロン A から見た時の抑制性のリンク。他のニューロンにも同様のリンクあり。 ● は一つの解に収束した時に発火したニューロン

デ　　モ　　内　　容　（1／3）

1. **概要**

デモンストレーションでは、色々なサイズの詰め込みパズルを２つの方式で解き、それぞれについて並列処理の様子や実行時間の台数効果の解説を行なう。

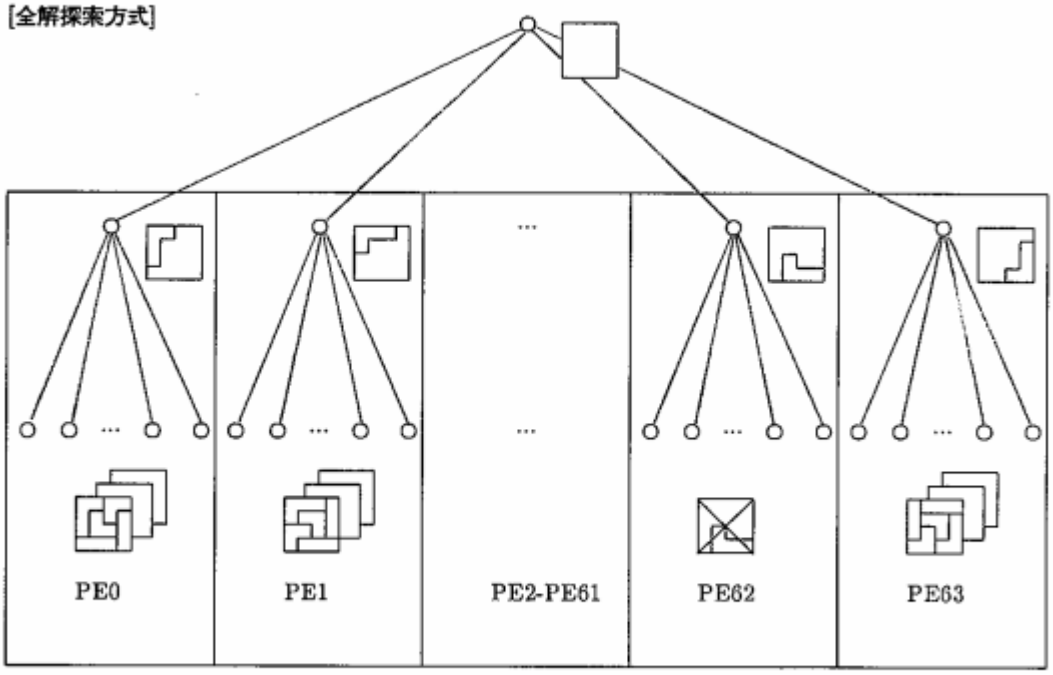
- 詰め込みパズルの問題を表示する。新たに問題を作成することもできる。
- 全解探索方式で解くか、ニューラル・ネットワークのシミュレーション方式で解くかを選ぶ。
- 負荷分散に必要な各種のパラメタを設定し、実行する。
- 実行時の負荷分散の様子は、パフォーマンスメータにて観察する。
- 解が一つ得られると、実行時間とその解を表示する。
- 全解探索方式の場合は、全解が求まった後に解の個数を表示し、必要に応じて特定の解を表示する。
- ニューラル・ネットワークのシミュレーション方式の場合は、エネルギーのグラフも表示する。

なお、12 ピースからなるペントミノと呼ばれるパズルを本デモンストレーションの全解探索方式と同じアルゴリズムで逐次型推論マシン PSI で解かせた場合、全解を求めるのに約４時間かかっている。

2. **全解探索方式**

人間が詰め込みパズルを解く場合、通常はピースを隅の方から一つずつ置いていき、置けなくなったらバックトラックして再度別のピースを置いてみる、という操作の繰り返しで解を導く。全解探索方式はこのアルゴリズムをプログラム化したもので、全解を求めるための一番標準的な方法である。但し、組み合わせの数が爆発的に増えるため、問題が大きくなった場合には解を一つ求めるにも相当な時間がかかるという欠点がある。本デモンストレーションでは、解の存在しない枝は早めに刈り込む枝刈り手法を用いて全ての解を探索し、何通りあるかを表示する。また、必要に応じて解を表示する。

[全解探索方式]



プログラム手法: 全解探索方式では兄弟関係にあるノード以下の探索は互いに独立して実行することができ、典型的なOR並列型の問題である。従って、互いに独立な探索を行なうプロセスを木状にフォークし、それぞれのプロセスから得られた解をマージして全ての解を得る。

負荷分散方式: ある一定の木の深さに達するまで、別々のプロセッサに(例えばランダムに)プロセスをフォークし、その後は自プロセッサ内にて実行する。負荷の分散方式は、サイクリック割り付けやランダム割り付けをプログラム中に記述しているが、実行時のプロセッサの忙しさの情報は用いていないことから準静的負荷分散と呼んでいる。負荷分散を行なう深さ、及びプロセッサの台数は実行前にユーザが与えることができ、これを変えることにより実行時間の変化、及び負荷バランスの変化が観測できる。

3. ニューラル・ネットワークのシミュレーション方式

人間の脳の情報処理の特徴は、10^{10}個以上と言われるニューロン同士の通信によって並列に処理されていることである。従って、ニューラル・ネットワークのシミュレーションは、プロセッサ間の通信が主体となる問題の典型的なものの一つである。特に、マルチPSIのようなプロセッサ台数が高々64台でかつ疎結合の並列マシン上で、通信が主体となるような問題をいかに効率良く実行するかは、並列処理研究の中で最も重要なテーマの一つである。

そこで、ニューラル・ネットワークのシミュレータをマルチPSI上に作成し、その上で詰め込みパズルを一種の最適化問題としてとらえて解かせた。なお、シミュレーションを行なったニューラル・ネットワークのモデルは、慶応大学の安西研究室、相磯研究室の協力により、現在研究中のガウシアンマシン・モデル*を採用した。ガウシアンマシン・モデルは最適化問題の解法で有名なホップフィールド・モデルを改良したもので、同じ系列のボルツマン・マシンと比べて自由度の高い制御を目指したものである。

アルゴリズム:

- 詰め込みパズルの各ピースの盤面上への置き方一つ一つをニューロンとする。図の例では、盤面上にピースが一つも置かれていない時のピース1の置き方は48通りある。同様に2及び3は16通り、4及び5は36通りある。従って、図の例ではニューロンの総数は152個になる。
- 各ニューロン間の結合強度、及びそれぞれのニューロンに固有なしきい値は、ホップフィールドのエネルギー関数と，この問題を解くために今回作成した目的関数との係数比較によって決定される。すなわち、図中でピース1のニューロンAが発火したときには、ピース1のA以外のニューロンの発火を抑制する(抑制性結合)。図中では･･･で現されている。これは、ある形のピースは盤面上に1個しか置けない、という制約条件を意味する。
- Aが発火したとき、その発火は盤面上のある領域を占めることになり、このピースと重なり合うような置き方に対応するニューロンとも、同様に抑制性の結合をする。図中では･･･で現されている。これは、それぞれのピースが盤面上で重なり合わないようにするための、制約条件を意味する。
- このようなネットワークに対して、各ノードに相当するニューロンに適当な初期値を与えてやる。その後はニューロンの動作法則に従って、お互いが通信をし合いながらニューラル・ネットワークのシミュレーションを行ない、詰め込みパズルを解く。なお、解は最初に収束した1つのみを表示する。

負荷分散方式:

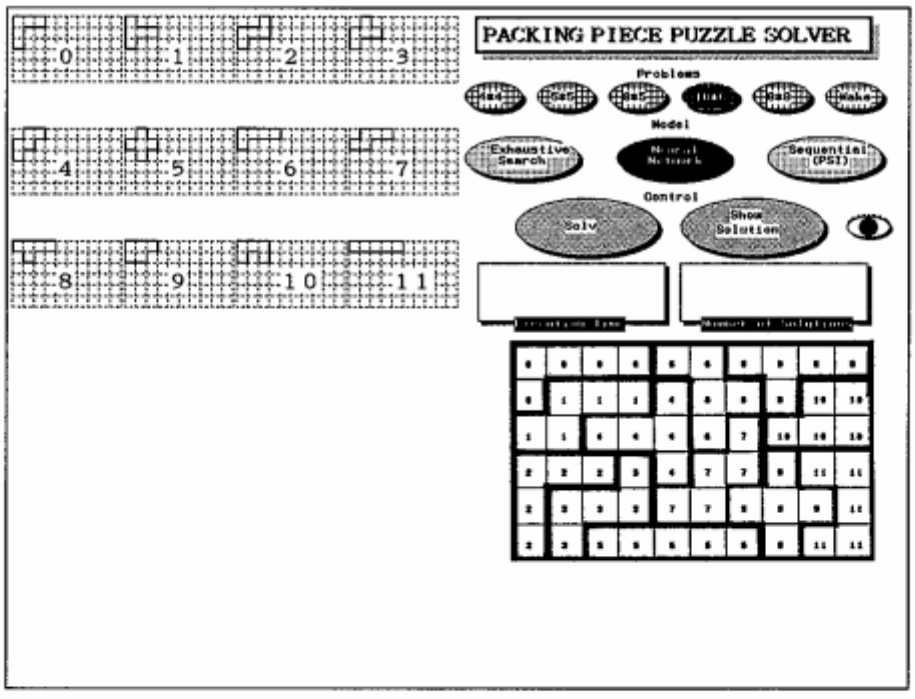
負荷分散方式としては次の2種類を用いて、それぞれの比較を行なう。

デ　　モ　　内　　容　(3／3)

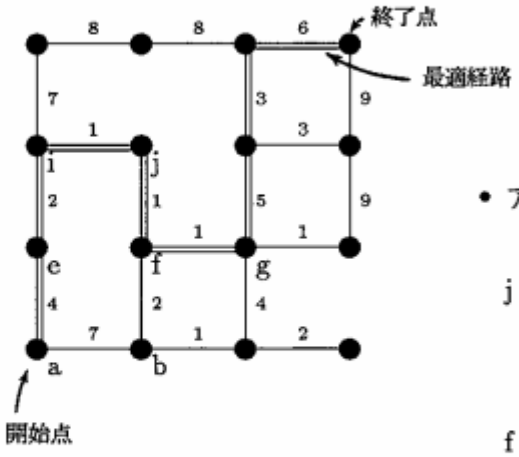
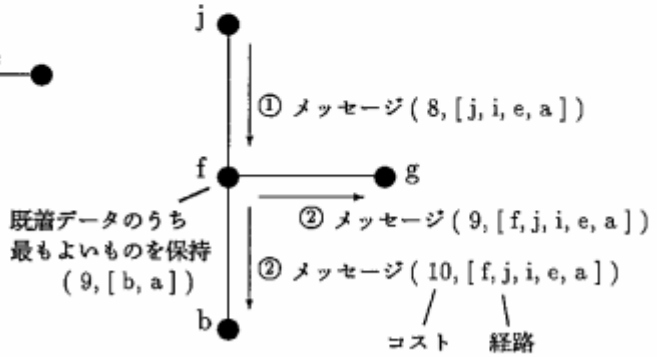
(a) 1問題1プロセッサ方式:
ニューラル・ネットワークのシミュレーション方式では、必ずしも制約条件を満たす解に収束するとは限らない。従って、複数のプロセッサに別々の初期値を与えてやり、複数のシミュレーションを実行することによって正解率を上げる。ただし、この方式はあくまでも正解率を上げるために用いた手段であり、通信が主体となる問題の負荷分散を考えたものではない。

(b) 数10ニューロン1プロセッサ方式:
本方式では、ニューロンを数10から数100ニューロンのグループに分け、グループ単位で各プロセッサに負荷分散させる。グループ化することによってプロセッサ間の通信量を抑える。

* ガウシアンマシン
ホップフィールドモデルに置けるローカルミニマムの問題を解決するために、系全体を温度及び基準活性値と呼ばれるマクロなパラメタによって制御する機能を取り入れた仮想的なマシン。具体的には、系の極小点にトラップされた状態から逃れるために、その時の系のエネルギーに見合ったノイズを印加すると言う方法である。



画面の表示例

題　名	**並列応用プログラム (4)　最適経路問題**
目　的	最適経路問題を例にして、最適解探索向きの並列アルゴリズム、プログラミングパラダイム、及び負荷分散方式についての試作とその評価を行う。
概要及び特徴	ネットワークの各辺にそれぞれ非負の値 (コスト) が割り当てられている時、その上の 2 点間を結ぶ経路のうちコスト最小の経路を求める。 乱数により 1 ～ 数万ノードの大規模ネットワークを生成している。 **問題の性質:** 最適解探索問題 **アルゴリズム:** 各ネットワークノード毎にプロセスを生成し、ノード間メッセージ通信により最適解を求める。 各ノード毎に既着メッセージのうちの最小コストを保持し、探索の枝刈りに使用。 **プログラミングパラダイム:** 固定個数プロセスの相互通信型 **負荷分散方式:** ネットワーク上の通信の偏在個所を複数のプロセッサが担当するように負荷分散を行う。
構　成	• 最適経路問題  • アルゴリズム 

デ　　モ　　内　　容　（1/2）

概要

最適経路問題とは、与えられたネットワークに対して各辺にそれぞれ非負の値（コスト）が割り当てられている時、ネットワーク上のある 2 点間を結ぶ経路のうち最もコストの低いものを求めるという問題である。デモンストレーションでは、乱数により 約 1 万ノードの大規模なネットワークを生成し、

- ある開始点を与えた時、その他のすべての点の各々に対する最適経路を求める
- ある開始点と終了点を与えた時、2 点間の最適経路を求める
  (基本的には前者のアルゴリズムと同様だが、ある種の枝刈りを行う)

という 2 通りの問題で最適経路を求める様子を紹介する。
今回のデモンストレーションでは、各ネットワークノード毎にプロセスを生成し、ノード間のメッセージ通信によって最適解を求めている。これは、木状にプロセスをフォークしてすべての経路を探索する方法に比べ、計算量のオーダは より小さい。

アルゴリズム

各ネットワークノード毎に プロセスを生成し、ノード間のメッセージ通信によって最適解を求めていく。
メッセージには、開始点からそのメッセージを受信したノードまでの経路とコストが含まれている。最初に開始点へコスト 0 のメッセージが送られることによりメッセージ通信が開始される。
また、各ノードは、常にその時点までに受信したメッセージのうちの最小コストとその経路を保持している。最初に保持しているコストは ∞ である (図 1 参照)。

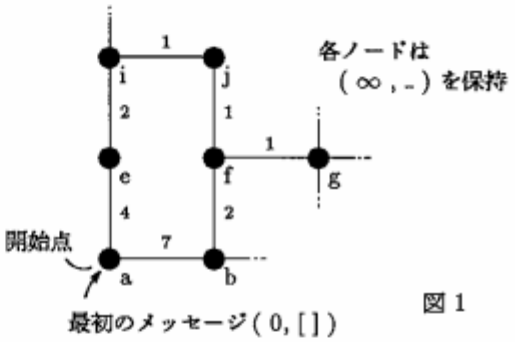


図 1

ノードがメッセージを受信した時、メッセージに含まれる値の方が保持しているものより小さければ、その値の方を保持することにし、すべての隣接ノードへ新しい値と経路を含んだメッセージを送る (図 2 参照)。
もしメッセージに含まれる値の方が保持しているものより大きければメッセージは捨てられ、ノードは何もしない。

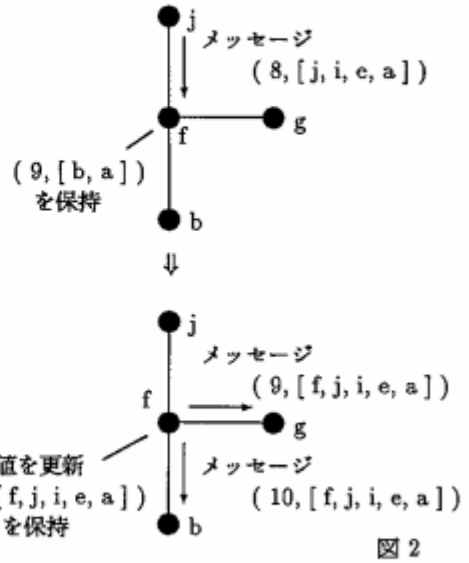


図 2

このようにしてすべてのメッセージがネットワーク上から消滅した時、各ノードは、開始点からそのノードまでの最適経路とそのコストを保持している。

デ　　モ　　内　　容　（2／2）

負荷分散方式

今回のインプリメンテーションでは、各ノードをプロセスで実現しノード間のメッセージ通信によって解を求める。１メッセージ通信当たりのノードの仕事量は比較的軽いため、通信が主体の問題となっている。また通信の流れは、開始点より次第に波状に広がっていくので、ネットワーク上に偏在しがちである。
そこで、以下の考え方に基づく負荷分散を試みている。

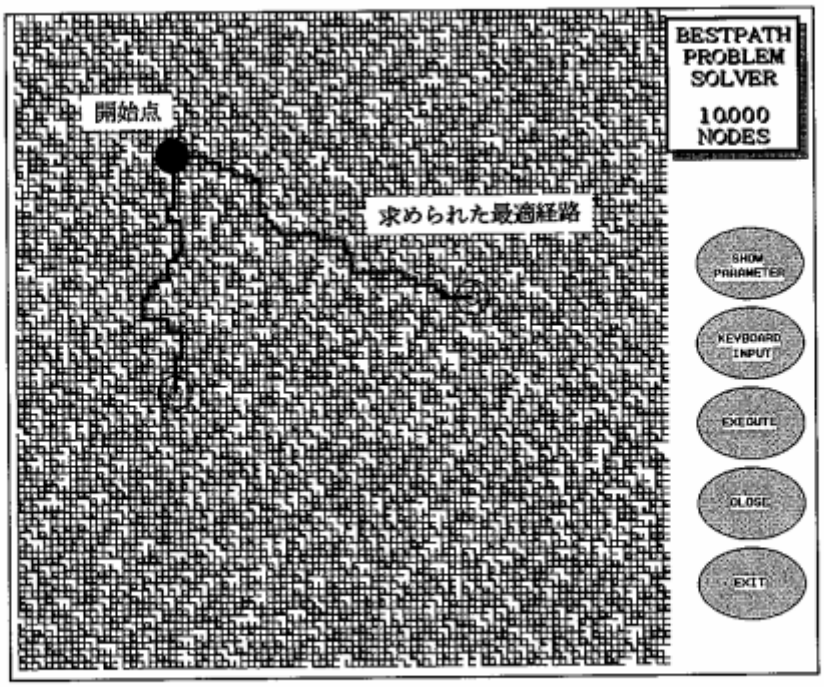
- 全体を部分ネットワークに分割し、部分ネットワーク毎に分散
- 通信の偏在個所を複数のプロセッサが担当するように分散

デモ内容

デモンストレーションでは、以上の条件に従った負荷分散方式をいくつか用意し、ネットワークの分割単位、プロセッサ台数、負荷分散方式を入力することにより実行する。ネットワークの分割単位が非常に小さい場合、各プロセッサの負荷は均一になるが、プロセッサ間にわたる通信が増大するため、通信のための処理時間が大きくなり、実行時間が増大する。逆に分割単位を非常に大きくすると、通信量は減少するが、プロセッサの負荷がばらついてしまうため、やはり実行時間が増大する。このような実行時間の変化を、設定を変更し実行することにより示す。また、負荷バランスについてもパフォーマンス・メータを通して観察することができる。

最適経路の表示例

乱数により生成されたネットワークは、以下のようにウインドウに表示される。各辺は、コストの大小により４段階の太さで表示されており、太い線で描かれたものほどコストは低いことを意味する。



ある開始点からその他のすべての点に対する最適経路を求めた場合