PIE 64 の並列処理管理カーネルのアーキテクチャ†

日高康雄†† 小池汎平†† 田中英彦††

本論文では，高並列推論エンジン PIE 64 の要素プロセッサ内の管理プロセッサによって実行される「並列処理管理カーネル」のアーキテクチャを取り上げて，高並列処理において重要な並列処理管理をどのように扱うかについて述べる．並列処理管理カーネルは，いわゆるオペレーティングシステムの核に相当するが，特に負荷分散とスケジューリングに重点を置いているのが特徴である．並列処理管理カーネルは，負荷分散処理の一部である動的負荷分割を担い，並列度等による分割の判断をして，過剰な並列性を抑制し，通信量を低減させる．スケジューリングでは，ヒープメモリ残量や並列度に応じた動的な優先度，そして，実行開始猶予時間を導入する．これらは，爆発的な並列性による資源の枯渇にプログラマが注意を払う必要をなくすこと，並列度をすみやかに向上させること，サスペンド，コンテクストスイッチのコストを減らすことを目的とする．これらの支援により，プログラムの並列性の向上にのみ，プログラマを専心させることを目指す．そして，コンパイラ等による静的負荷分割と並列処理管理カーネルによる動的負荷分割の効果の違いに関する予備評価を行った．その結果，並列性が要素プロセッサ数を大きく上回る場合は動的負荷分割が，それほど大きく上回らない場合は静的負荷分割が，それぞれ有効であり，両者を複合させるのが最も有効であることがわかった．

1. はじめに

一般に，並列処理オーバヘッドの中では，通信のレイテンシと同期コストが最も大きな問題であると言われている[1]．高並列処理では，これらの問題に加えて，負荷分散やスケジューリングなどの並列処理管理が問題となる．並列処理管理は，低並列処理ではそれほどオーバヘッドとならないが，高い並列性を抽出するために粒度を細かくすると頻繁に要求が生じるため，高並列処理においては不可避のオーバヘッドとなる．

そこでわれわれは，通信処理，同期処理，並列処理管理を本来の計算と並列に処理することにより，並列処理オーバヘッドを効果的に吸収する処理モデルを検討し，現在研究開発を進めている高並列推論エンジン PIE 64 の要素プロセッサに，協調動作する3種類のプロセッサがそれぞれ「計算」，「通信と同期」，「管理」を担うという，複合型アーキテクチャを採用した[15]．

本論文では，この管理プロセッサによって実行される，並列処理管理カーネルについて述べる．並列処理管理カーネルは，いわゆるオペレーティングシステムの核に相当し，PIE 64 の記述言語である Fleng では，記述できない，もしくは，記述するのが適当でない，低いレベルのシステム管理機能を担う．その中でも特に，並列処理に特有の「並列処理管理」に重点をおいている（図 1）．プログラミング環境やユーザ管理などのオペレーティングシステムの上位の機能は，応用プログラムと同様に Fleng で記述，実行されるため，並列処理管理カーネルの機能には含まれない．

まず，並列処理管理に関する定性的な検討を行う．並列処理管理とは，並列処理のスレッドごとに必要となる管理であり，資源管理などとは区別される．オペレーティングシステムの行うシステム管理のうち，何が並列処理管理に含まれるかを検討し，それぞれの特徴をまとめる．その中で，特に実行効率に大きな影響を与える，負荷分散とスケジューリングについて，その要件を整理する．

次に，PIE 64 の並列処理管理カーネルのアーキテクチャについて，特に負荷分散とスケジューリングの戦略に重点を置いて述べる．

PIE 64 では負荷分散を，コンパイラ等による静的負荷分割，並列処理管理カーネルによる動的負荷分割，相互結合網による動的負荷平滑の3つのレベルで扱う．Fleng のプログラムが元々持っている高い並列性と，これら3レベルの負荷分散により，負荷分散の3つの要件である，並列性の抽出，通信量の低減，負荷のバランスを満足させる．

並列処理管理カーネルによるスケジューリングでは，動的な優先度と実行開始猶予時間を導入する．動的な優先度は，ヒープメモリ残量や実行時の並列度に従って変化する優先度であり，爆発的な並列性による資源の枯渇にプログラマが注意を払う必要をなくすことと，並列度をすみやかに向上させることを目的とする．実行開始猶予時間は，コンパイル時に部分的に検

† The Architecture of Parallel Processing Management Kernel of PIE 64 by YASUO HIDAKA, HANPEI KOIKE and HIDEHIKO TANAKA (Department of Electrical Engineering, Faculty of Engineering, The University of Tokyo).

†† 東京大学工学部電気工学科

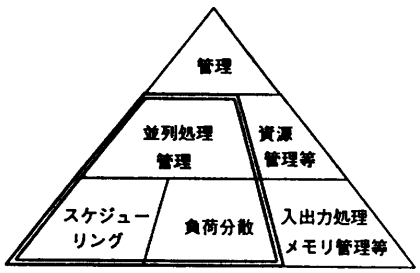

図 1 並列処理管理
Fig. 1 Parallel processing management.

出できるデータ依存関係を用いて，サスペンド，コンテクストスイッチのコストを減らすためのものである．

最後に，どちらも通信量の低減に効果のある，静的負荷分割と動的負荷分割の比較について述べる．静的負荷分割は，並列度と要素プロセッサ台数によらずに，一定の効果があるのに対し，動的負荷分割は，並列度が要素プロセッサ台数を大幅に上回り，並列性を抑制する余地が十分にある場合は非常に高い効果を持つが，並列度と要素プロセッサ台数が近付くと，不十分な並列度を向上させるために頻繁に分割せざるを得ず，通信量が増大して急激に効果がなくなるということを，予備評価の結果に基づいて示す．また，両者の利点を合わせ持つ複合方式が，最も高い効果を持つことも示す．

従来より，負荷分散やスケジューリングに関しては様々な研究が行われてきた[3),8)]．あらかじめプログラムの挙動がわかっている場合や，問題の性質が規則的な数値計算問題の研究例は数多いが，プロセスの数が膨大で，挙動も不規則である高並列記号処理に適用可能な手法は少ない．高並列記号処理では，プログラマが戦略の指定を行い，システムはその指定に従うという方法が一般的に用いられているが[4),5),9)]，これは，プログラマはプログラムの挙動を把握できるという仮定に基づいている．

しかし，高並列記号処理ではこの仮定は必ずしも成り立つとは限らない．適度な高さの並列性を抽出するようにプログラムしたつもりでも，プログラマが意識しなかった依存関係のために，思ったほど並列度が上がらないことがよくある．十分に高い並列性を抽出するためには，プログラマは並列性をできるだけ抽出することに注意を払うべきであり，過剰な並列性を抑制することには注意を払うべきではない．

PIE 64 では，並列処理管理カーネルによる負荷分散，スケジューリングの支援によって，プログラマが過剰な並列性を心配したり，細粒度プロセスのまとめ方を指定したりする必要をなくし，アルゴリズムで制限されてしまう最大並列性を向上させることにのみ，プログラマを専心させることを目指す．

2章と3章では，それぞれ PIE 64 の主要な記述言語である Fleng と PIE 64 について，簡単に説明する．4章では，並列処理管理に関して定性的な検討を行う．5章では，並列処理管理カーネルのアーキテクチャについて，負荷分散とスケジューリングに重点をおいて述べる．6章では，静的負荷分割と動的負荷分割の効果の違いに関する予備評価について述べる．

2. Fleng

PIE 64 の主要な記述言語である Fleng[7)] は，細粒度高並列記号処理向けのプログラミング言語で，コミッティッドチョイス型言語，並列論理型言語の1つである．同種の言語で有名なものには，GHC[6)] がある．

Fleng のプログラムは，次のような形の定義節を，宣言的に書き並べたものである．

Head :- $Goal_1$, $Goal_2$, …, $Goal_n$.

:- の左側をヘッド部，右側をボディー部と言う．

プログラムの実行は，トップゴールの投入によって開始される．与えられたゴールとヘッド部がユニフィケーション可能な定義節の1つが選択され，元のゴールはそのボディー部によって，新たなゴールにリダクションされる．プログラムの実行はこの処理の繰り返しであり，これが並列実行の単位となる．

ゴールの持つ変数は単一代入変数で，未定義変数の確定は，必ずボディー部において行われる．ユニフィケーションの際に，ゴール側の未定義変数の値が確定しなければならない時には，ゴールの実行はサスペンドする．別のゴールのボディー部の実行においてその変数の値が確定すると，中断されていたユニフィケーションが再開される．

PIE 64 のインプリメントでは，ゴールの実行は，述語名とアリティによる実行コードへのディスパッチで開始され，ベクタで表現されたゴール（ゴールフレーム）が検査されて，成功すると新たなゴールフレームが作成される．ゴールフレームは，ヒープメモリ上に動的に作られ，リダクションの際に現れる，新たな変数，リストセル，ベクタも，ヒープメモリ上に動的にその領域が確保される．

3. 並列推論エンジン PIE 64

PIE 64[13] は，64台の要素プロセッサと2系統の相互結合網から構成されている．PIE 64 の要素プロセッサは，推論ユニット (IU—Inference Unit)[15] と呼ばれる．相互結合網[14]は，自動負荷分散機能を持ち，負荷最小の IU を自動的に選択し，そこに接続することができる．

IU の概略を図 2 に示す．IU は，推論処理プロセッサ (UNIRED—Unifier/Reducer)，通信プロセッサ (NIP—Network Interface Processor)，管理プロセッサ (MP—Management Processor) とメモリなどからなっている．

UNIRED[16] は，Fleng のプログラムを実行するプロセッサである．各定義節は UNIRED の機械語にコンパイルされ，各ゴールのユニフィケーションとリダクションは，UNIRED 上の1つのスレッドとして実行される．ヒープメモリは PIE 64 全体で単一アドレス空間を持ち，NUMA (Non-Uniform Memory Access time) 型の分散共有メモリを構成する．また，UNIRED はパイプライン化 MIMD プロセッサ[2]の一種で，そのパイプラインは最大4つのスレッドで共有され，リモートメモリ参照時のレイテンシトレランスの向上がはかられている．ただし，同じスレッドのパイプライン投入間隔が固定である通常の循環パイプラインとは異なり，あるスレッドがリモートメモリを参照した時や並列度が低下した時は，空きスロットを作るのではなく，実行可能なスレッドを連続してパイプラインに投入する．

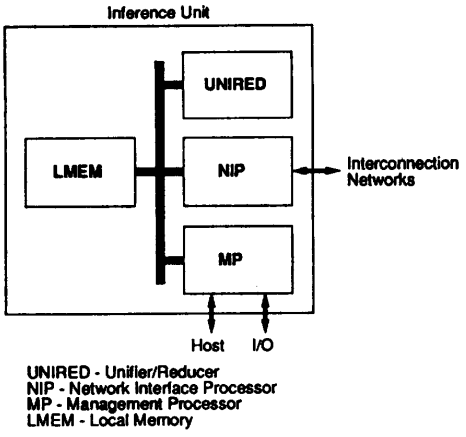

図 2 推論ユニットの概略
Fig. 2 The abstract organization of Inference Unit.

NIP[17] は，相互結合網とのインタフェースを持ち，IU 間の通信処理，Fleng のプロセス間同期処理を実行するプロセッサである．NIP は UNIRED や MP からのコマンドを受け，ハードウェア化されたシーケンサによって通信処理や同期処理を高速に実行する．

MP は並列処理管理カーネルを実行する．MP には，汎用の RISC 型高速マイクロプロセッサの SPARC をコントローラとして使用し，今後のハードウェア化のための基礎データ収集を可能とするためにも，様々な並列処理管理アルゴリズムの実験を可能とする汎用性を重視している．

これら3種類のプロセッサは高速コマンドバスで結ばれ，図 3 に示すようなコマンド／リプライのやり取りによって協調動作する．表 1 に，MP が発行／受信する主要なコマンドをあげる．この協調動作の概略を，以下に述べる．

UNIRED は，MP から reduce コマンドを受けとると，ゴールのユニフィケーションとリダクションを行うスレッドの実行を開始する．スレッドの実行中にリモートメモリ参照が生じると，NIP に deref, read コマンドを発行する．変数の値を確定させる時には，NIP に bind コマンドを発行する．リダクションで作られた新たなゴールは，newgoalコマンドでUNIRED から MP に渡される．未定義変数参照によって，ゴール実行がサスペンドする時には，suspend コマンドが，ゴール実行の終了時には，endreduce, fail コマンドが，それぞれ MP に発行される．

負荷分散のためのゴール転送時には，MP から NIP に writem, writel コマンドが出され，到着した先の IU 上で，NIP は MP に newmsg, newload コマンドを出して，ゴールの到着を伝える．

Fleng の単一代入変数を用いたプロセス間同期処理

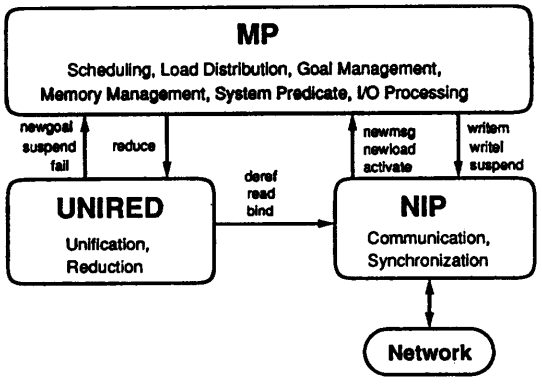

図 3 推論ユニット内の協調処理モデル
Fig. 3 The cooperation model in Inference Unit.

は，概ね以下のように行われる．まず，未定義変数を参照した UNIRED は，suspend コマンドを MP に発行して，実行を中断する．MP はサスペンションレコードを割り当てた後，NIP に suspend コマンドを発行し，それを受けて，NIP は，未定義変数にサスペンションレコードを登録する．一度 MP を介するのは，サスペンド中のゴールを MP が管理し，1つのゴールが複数の変数にサスペンドするのを許すためである．変数の値が確定すると，NIP は bind コマンドを受けとり，NIP はその変数に登録されていたサスペンションレコードをすべて読み出して，activate コマンドを MP に発行する．MP はサスペンションレコードにアクティベイト済みの印をつけた後，そのゴールを再びスケジューリング対象に組み入れる．サスペンド登録，アクティベイトの処理の間に必要な IU 間の通信は，NIP が自動的に行う．

4. 並列処理管理の定性的検討

従来のオペレーティングシステムでも行われている，実行プログラムに何らかのサービスを提供するための管理は，通常，陽に出される要求に従って行われる．これらの管理のための処理量は，要求を出す実行プログラムの性質や，本来の計算の処理量で決まっているもので，並列処理の粒度には左右されない．

これに対して並列処理管理は，プログラムの実行時間の短縮が目的であり，要求に対するサービスが目的ではないため，陽に要求が出されなくても，プログラムの実行を裏から制御し，効率的な並列実行を支援しなければならない．

サービスや資源の管理，保護などを目的とする通常の管理との区別を明確にするために，並列処理管理とは「並列処理のスレッドごとに必要となる管理である」と定義する．ここでスレッドとは，並列処理の処理単位のことで，1つのスレッドの実行は逐次的に行われるものとする．このような並列処理管理のための処理量は，並列処理の粒度が細かくなるにつれて，増大してくる．

並列処理管理の比重の小さい，逐次形の計算機，あるいは大粒度の並列計算機では，本来の計算と管理を同じプロセッサで実行しても，問題は生じなかった．しかし，高並列の並列計算機では処理の粒度が細かくなり，並列処理管理の比重が増大するため，本来の計算と管理を同じプロセッサで実行していてはオーバヘッドの増大を招く．並列処理管理を本来の計算とは別のプロセッサでオーバラップして実行することにより，ユーザプログラムの実行効率を高くすることができる．

並列処理管理には，主に以下のものが考えられる．

1. 負荷分散
   どのようにプログラムをスレッドに分割するかを決定し，各要素プロセッサへのスレッドの割り当てを決定する．

表 1 管理プロセッサが発行／受信する主要なコマンド
Table 1 Commands for Management Processor to send or receive.

MP → UNIRED	
reduce	ゴールのユニフィケーションおよびリダクションを行うスレッドの実行を UNIRED に開始させる．
UNIRED → MP	
newgoal	リダクションの結果，生成されたゴールを MP に渡す．
suspend	ユニフィケーションをサスペンドさせた未定義変数（複数）を MP に伝える．
endreduce	リダクションに続けて再帰的に実行するゴールがないためにスレッドの実行が終了したことを MP に伝える．
fail	ユニフィケーションに失敗したことを MP に伝える．
MP → NIP	
writem	転送先 IU が決定している場合のゴール転送を NIP に行わせる．
writel	転送先 IU が決定していない場合のゴール転送を NIP に行わせる．転送先は，相互結合網の自動負荷分散機能を用いて決定する．
suspend	サスペンションレコードを未定義変数に登録する．
NIP → MP	
newmsg, newload	他の IU から転送されてきたゴールを MP に渡す．
activate	値が確定した未定義変数に登録されていたサスペンションレコードを MP に渡す．

2. スケジューリング
   各要素プロセッサに割り当てられたスレッドの実行順序を決定する.
3. 同期処理
   スレッド間の実行の排他制御や待ち合わせなどを行う.
4. 実行制御
   ユーザの指示による実行の中止や，デバッガによるトレース実行などを行う.
5. アドレス空間の管理
   スレッドごとにアドレス空間が異なれば，その管理が必要である．アドレス空間が共通ならば処理を軽減できる.
6. コンテクストの管理
   レジスタの値やスタックなど，スレッドごとの実行コンテクストの管理．スタックを持たなければ，レジスタ等の値だけ管理すれば良い.

スケジューリング，同期処理，実行制御は，いずれもスレッドの実行状態（待ち，実行可能，実行中）を扱うという点で関連があるが，同期処理は，正しい実行結果を得るために厳密に守らなければならない処理で，スケジューリングは，正確である必要はないが，効率の良い実行を目指してなるべく適切に行うべき処理である．また，実行制御は，プログラムの本来の意図とは異なる例外的な処理であるという点で，他の2者とは異なる.

同期処理は，記述言語で提供する同期プリミティブの種類を決めてしまうと，同期処理の処理内容もかなり決まってしまう場合が多い．このため，アルゴリズムをハードウェア化して，処理を高速化することが，比較的容易である.

Flengで提供している同期プリミティブは単一代入変数であり，PIE 64では3章で述べたように，その処理の主要な部分は NIP の suspend/bind/activate コマンドとして，ハードウェア化されている.

アドレス空間の管理は，一般にかなり重い処理であり，大粒度の低並列計算機でも，スレッド間のアドレス空間を共通にして，スレッド生成時や終了時の処理を軽くしている場合がある．また，並列処理管理の定義からは少し外れるが，スレッドごとのアドレス空間のほかに，要素プロセッサごとのアドレス空間の管理やガーベジコレクションなども必要である．各要素プロセッサが異なるアドレス空間を持つ場合は，アドレスの輸出入表などを用いた管理が必要である．これはかなり重い処理となるが，ガーベジコレクションを局所的に行いやすいという利点を持つ．逆に，アドレス空間を共通にした場合は，通常の処理は高速化されるが，ガーベジコレクションを全要素プロセッサで一括して行う必要がある.

PIE 64は，すべてのIUに渡る単一アドレス空間のヒープメモリを，すべてのスレッドで共有するという方式をとっており，スレッドごとや，IU ごとのアドレス空間の管理は必要ない．ヒープメモリのガーベジコレクションは全 IU で一括して行う.

コンテクストの管理では，スレッドごとにスタックを用意する場合には，その管理が必要になる．しかし，高並列記号処理では，実行の開始時にもスレッドの数は決まっておらず，必要なスタックの深さを知るのも難しい．スレッドごとにスタックを用意しなければ，CPU のレジスタ等の値だけを管理すれば良い.

PIE 64 では，スレッドごとのスタックを持たないため，スタックの管理は必要ない．スレッドごとのコンテクストは，ヒープメモリ上に作られるゴールフレームに格納される．ゴールフレームからレジスタへのコンテクストのロードや，その逆のセーブは，UNIRED 自身が行うため，MP はゴールフレームの先頭アドレスのみをスレッドのコンテクストとして管理する.

以下では，実行効率に与える影響が特に大きいと考えられる，負荷分散とスケジューリングについて，その要件を整理する.

4.1 負荷分散に対する要件

1. 並列性の抽出
2. 負荷のバランス
3. 通信量の低減

すべての要素プロセッサが稼働するように，並列性を抽出しなければならない．また，負荷のバランスをとり，一部の要素プロセッサに負荷が集中しないようにする必要がある．負荷としてはスレッド数のみならず，ヒープメモリの使用量も考慮し，分散配置されたヒープメモリがバランス良く消費されるようにする必要がある．そして，なるべく通信量を低減させる必要がある.

負荷分散を負荷分割と負荷割り当ての2つに分けて考えると，一般には，負荷分割には並列性の抽出と通信量の低減が要求され，負荷割り当てには負荷のバランスと通信量の低減が要求される．要素プロセッサ間の通信遅延時間が均等と見なせる場合は，負荷割り当

てには負荷のバランスのみが要求される．また，並列性の抽出と通信コストの低減は相反するため，それらのトレードオフをとる必要がある．

4.2 スケジューリングに対する要件

1. 同期コストの低減
2. ヒープメモリの枯渇の回避
3. 並列性の抽出
4. ユーザ指定の優先度

同期処理をハードウェアによって支援している場合にも，スケジューリングが早すぎると，サスペンドするためのコンテクストスイッチのコストが発生する．同期コストをさらに低減させるためには，あらかじめ必要なデータが揃ってから起動するように，スレッドのスケジューリングを工夫する必要がある．

ストリーム並列性による並列プログラムでは，プロデューサ-コンシューマ間のストリームのフロー制御を行わないと，ヒープメモリが枯渇して実行終了に至らない場合がある．フロー制御のための要素駆動型やBounded Buffer などのプログラミング手法もあるが，プログラマの負担となり，同期コストの増加にもなる．また，この種のプログラムは，コンシューマ内部で，ストリーム上の各データを処理するスレッドを並列に動作させることにより，非常に高い並列性を抽出できる可能性を持っているのにもかかわらず，これらのプログラミング手法は，抽出し得る並列性を大きく制限してしまう．すなわち，ヒープメモリの残量等を用いて優先度を動的に制御し，適切なスケジューリングによってフロー制御を行うのが望ましい．

並列性の抽出は，負荷分割に対する要件でもあるが，実行時のスケジューリングにおいても，並列度が低い時には，並列度を向上させるスレッドを優先させる必要がある．

そして，優先的処理や投機的計算などを可能とするために，ユーザ指定の優先度も用意する必要がある．

5. 並列処理管理カーネル

並列処理管理カーネルは各 IU 上の MP によって実行され，UNIRED が実行するスレッドの制御などを行う．並列処理管理カーネルの構成を**図 4** に示す．負荷分散とスケジューリングがその中核をなし，メモリ管理によるヒープメモリの消費状況や，NIP から得られる他の推論ユニットの負荷情報などと密接に関連して，適切な負荷分散とスケジューリングを目指す[10]．

5.1 負荷分散

5.1.1 三階層の負荷分散

PIE64 では，負荷分散を並列処理管理カーネル単独ではなく，**図 5** に示すように，3つのレベルで扱う．

1. コンパイラによる静的負荷分割
   コンパイラ等でプログラム中の負荷分散を行い得る箇所（スレッドを生成する箇所とヒープメモリを確保する箇所）における戦略を，並列性を損なわずに，なるべくローカルメモリ参照率

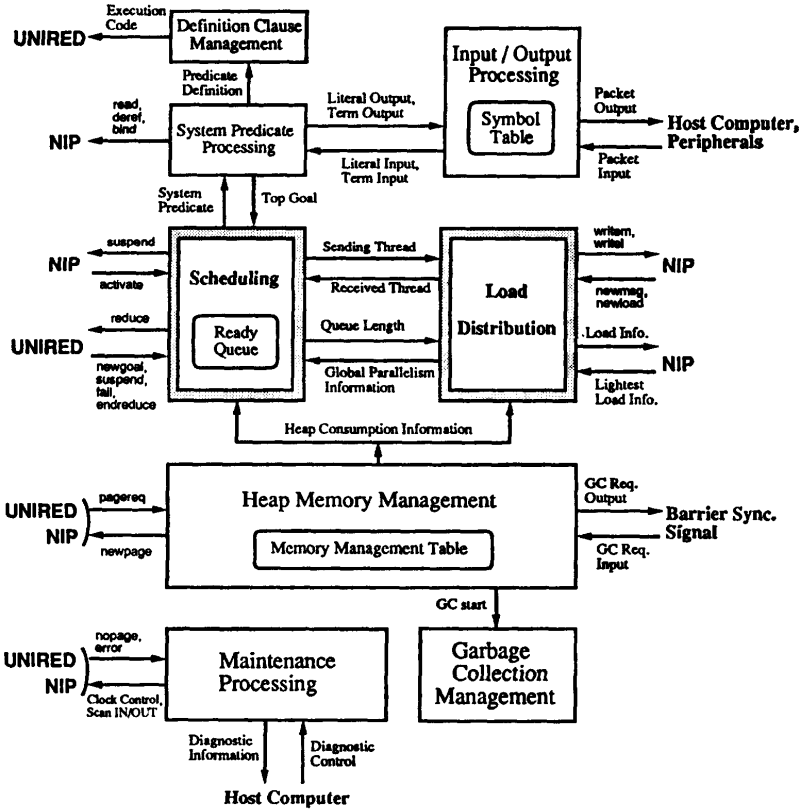

図 4 並列処理管理カーネルの構成
Fig. 4 The organization of Parallel Processing Management Kernel.

を高めるように最適化し，データ依存関係に沿った分割を行う[11]．

2. 並列処理管理カーネルによる動的負荷分割
   並列処理管理カーネルは，その時の並列度やヒープメモリの残量に従って，分割する／しないという簡単な判断を行い，プログラム実行の計算木（制御依存関係）に沿った分割を行う．この動的負荷分割によって，過剰な並列性を抑制して，通信量を低減させる．詳細は後で述べる．
3. 相互結合網を用いた動的負荷平滑
   動的負荷分割で分割すると決まった場合は，相互結合網の自動負荷分散機能[14]を用いて負荷最小の IU に割り当てて，IU 間の負荷のバランスをとる．分割しないと決まった場合は，ローカル IU 内にとどめる．

これは，1により通信量を低減させ，2により過剰な並列性の抑制によりさらに通信量を低減させ，3で負荷のバランスをとるという方針に基づいている．

5.1.2 並列処理管理カーネルによる動的負荷分割

並列処理管理カーネルは，以下のいずれかが満たされる場合，静的負荷分割によって指定された戦略に従い，任意の IU という戦略が指定されている箇所で負荷分割を行う．

- 全体の並列度が低い時．
- 全体の並列度に対して，その推論ユニット内の負荷が非常に高い時．
- ヒープメモリ残量が少なく，その分割箇所でプロデューサ型のスレッドを生成する場合．

上記のいずれも満たされない場合，静的負荷分割により任意の IU という戦略が指定されている箇所では，

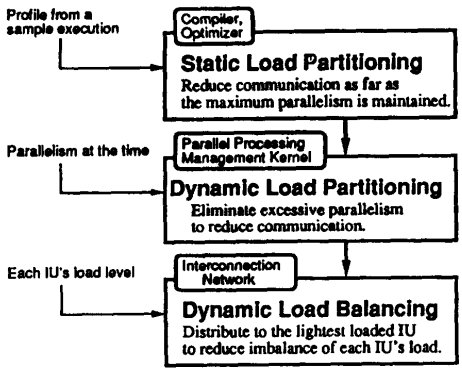

図 5 PIE 64 における三階層の負荷分散
Fig. 5 The three layers of load distribution in PIE 64.

負荷分割をせずにローカル IU を選択する．任意の IU 以外の戦略* が指定されている箇所では，その戦略に従う方法と，その戦略を無視してローカル IU を選択する方法が考えられる．これらの効果の違いについては，予備評価の章で述べる．

2つの相互結合網である PAN (Process Allocation Network) と DAN (Data Access Network) にはそれぞれ，スレッド数に関する負荷情報とヒープメモリ使用量に関する負荷情報を流し，次のように使い分ける．

- スレッドの負荷分散では，
  - 全 IU でヒープメモリ残量が十分にある場合は，PAN を用いる．
  - ヒープメモリ残量が少ない IU が1台以上存在する場合は，プロデューサ型のスレッドには DAN を，それ以外のスレッドには PAN を用いる．
- ヒープメモリ上のデータの負荷分散** では，DAN を使用する．

なお，スレッドを実行中のヒープメモリの確保は UNIRED 上で行われるため，UNIRED のコードにヒープメモリ確保における負荷分割を実行するものとしないものの2つを用意し，MP が出す reduce コマンドのディスパッチ先によって，分割するかどうかを制御する．

5.2 スケジューリング

スケジューリングでは，以下の要素を考慮して次に実行すべきスレッドを選択する．

1. ユーザ指定の優先度．
2. 動的な優先度．
3. 実行開始猶予時間．

ユーザ指定の優先度は，前述のように，優先的処理や投機的計算などを可能とするためのものである．

動的な優先度は，その時の並列度やヒープメモリの消費状況によって変化するもので，そのスレッドの性質によって，次のように変化の仕方が異なる．

- ガーベジコレクション後のヒープメモリ残量が少ない場合は，ヒープメモリ上に多くのデータを生成するプロデューサ型のスレッドの優先度を低くする．
- 並列度が低いときは，多数のスレッドを生成するフォーク型のスレッドの優先度を高くする．

---

* 他のあるデータが配置されている IU と同じ IU を指定する戦略．
** 新しい変数やベクタなどの領域を，他の IU のヒープメモリ上に確保する際の負荷分散．

実行開始猶予時間は，そのスレッドが生成されてから，実行を開始するまでの時間を示すもので，参照する変数の値がすべて確定してから，スレッドの実行を開始させるためのものである．データ依存関係に沿って実行開始猶予時間を静的に決めておくことにより，サスペンド回数を減らし，同期コストを低減させることができる．

データ依存関係がプログラムの実行によって動的に決定されるために，適切な実行開始猶予時間が静的に定まらないスレッドの場合，どの変数を参照するかを調べておく．もし必ず参照する変数があれば，そのスレッドを生成後，すぐに実行可とせずに，その変数に対してまずサスペンド登録を行う．スレッドを実行させる前のサスペンド登録は，UNIRED のコンテクストスイッチを伴わないため，通常のサスペンド処理にくらべて，オーバヘッドが非常に小さく，同期コストを低減できる．

実行可のスレッドを保持するレディーキューは，図6に示すように，スライド型のキューを優先度ごとに設けて構成する．実行開始猶予時間は，正確な時間ではなく世代を単位として表し，猶予時間が0のスレッドがなくなると，キューをスライドさせる．優先度の低いスレッドは，それよりも優先度の高いキューが空の時に限って，スケジュールされる．変数のバインドによりアクティベイトされたスレッドは，該当する優先度の猶予時間0のキューに入れられる．

5.3 資源管理等

並列処理管理カーネルは，負荷分散，スケジューリングのほかに，メモリ管理，入出力処理，定義節管理，システム述語処理，メンテナンス処理なども担う．

ヒープメモリのアドレス空間は，全スレッドで共有されるグローバル物理アドレス空間であり，アドレス変換テーブルはない．各 IU 内に分散配置されるヒープメモリはページ単位に分割して管理される．MP から UNIRED，NIP へのヒープメモリの割り当てはページ単位で行われ，ワード単位の割り当ては，UNIRED，NIP 内部で行われる．

ヒープメモリのガーベジコレクションは，全 IU で同期して行う一括型のガーベジコレクションである[12]．ガーベジコレクションの開始，フェーズの移行は，バリア同期信号を用いて行う．マーキング等の実際の処理は UNIRED で，一時的な間接参照テーブルの作成は NIP で行い，並列処理管理カーネルは，フェーズ移行の管理，UNIRED のコンテクスト管理，

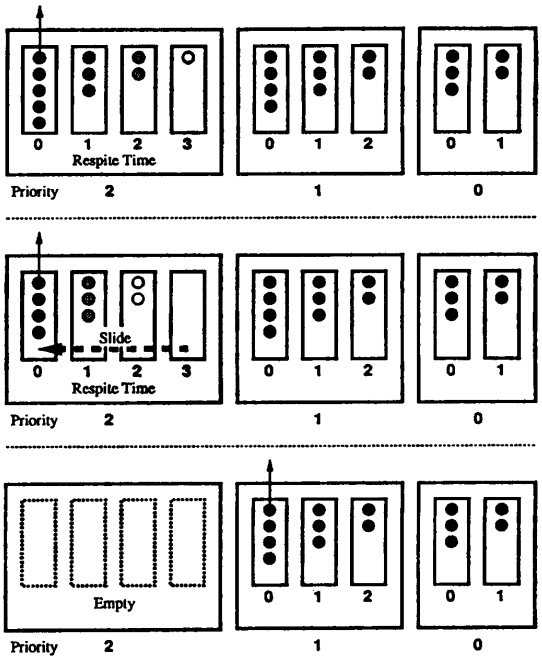

図 6 優先度ごとのスライド型レディーキュー
Fig. 6 The sliding ready queue for each priority.

他 IU からのマーキング依頼の受け付けなどを処理する．

入出力処理では，ホスト計算機や周辺装置との間のパケット入出力を，リテラルやタームのストリームに変換する．定義節管理では，述語ごとにコンパイルされた UNIRED の実行コードの管理を行う．システム述語処理では，Fleng プログラムからの組み込み述語呼び出しを解釈し，カーネル内の様々な処理を呼び出したり，浮動小数点演算の実行を行ったりする．メンテナンス処理では，ホスト計算機上のメンテナンスプログラムと連携して，障害の解析などを支援する．

6. 予備評価

PIE 64 で行われる三階層の負荷分散のうち，コンパイラによる静的負荷分割と並列処理管理カーネルによる動的負荷分割の2つは，共に通信量を低減させる働きを持つ．この2つの負荷分割の効果にどのような違いがあるかについて，簡単な予備評価を行った．

予備評価には，逐次型ワークステーション上の Fleng インタプリタを利用した．ゴールやヒープ上のデータごとに，それがどの IU に存在するかを示す履歴をトレースし，メモリ参照の際に，実行中のゴールと参照されたデータが存在する IU が同じかどうかで，ローカルメモリ参照であるか否かを判定した．ス

ケジューリングは，2つのゴールキューを世代ごとに交互に使った幅優先とし，スケジューリングの特性を実際の並列処理系に近づけ，世代交代時のキューの長さからその世代の最大並列度を求めた．

負荷分散の最終目的は実行時間の短縮であるため，その効果の比較も本来，実行時間を尺度とするのが最も望ましい．しかし，この予備評価では実機の回路を忠実にシミュレーションしていないため，相互結合網内の衝突，メモリアクセスの衝突，並列処理管理の実行時オーバヘッド，コンパイラの特性などを反映させた正確な実行時間を求めるのは困難である．

そこで，負荷分割の要件の1つである通信量の低減に着目して，特定のシステムに依存しない評価を行うことにした．もう1つの要件である並列性の抽出は，並列性が不十分な時に実行時間に影響を与えるが，ここで評価する手法ではどれでも，そのような時には最大並列性を維持する戦略をとるため，その効果に大きな違いはない．

一方，Flengにおける主要な通信には，リモートメモリ参照とゴール転送があるが，ゴール転送は通信のレイテンシの影響をあまり受けない．なぜなら，転送先では他のゴールを処理している限り，そのゴールの到着を待たないからである．しかし，リモートメモリ参照は，その結果が返るまで処理を中断させ，実行時間に大きな影響を与えるため，この予備評価ではローカルメモリ参照率を尺度として評価することにした．

サンプルプログラムには6-Queenを用い，以下の6種類の戦略を使用して，測定を行った．

1. 単純戦略：ヒープ上のデータはすべてローカルIU上にとる．リカージョンのゴール1つをローカルIUで実行し，それ以外のゴールを実行するIUは，1つずつ自動負荷分散で決定する．
2. 静的戦略：プロファイルデータを元に，実行前に静的に定めた戦略に従って実行する．戦略が任意のIUである箇所では，自動負荷分散でIUを決定する．
3. 動的戦略：世代交代時にアクティブなIUの数を調べ，IU台数に足りない数を，次の世代で行う自動負荷分散の最大回数とする（これは，以下の複合戦略でも同様）．最大回数の自動負荷分散を行うまでは単純戦略をとり，それ以後はすべてローカルIUを選択する．
4. 複合戦略-1：最大回数の自動負荷分散を行うまでは静的戦略をとる．それ以後は，静的戦略が自動負荷分散指定の場合のみローカルIUを選択し，それ以外の場合は静的戦略に従う．
5. 複合戦略-2：最大回数の自動負荷分散を行うまでは静的戦略をとる．それ以後は，ゴールを実行するIUにはすべてローカルIUを選択し，ヒープ上のデータを確保する箇所では，静的戦略が自動負荷分散指定の場合のみローカルIUを選択し，それ以外の指定の場合は静的戦略に従う．
6. 複合戦略-3：最大回数の自動負荷分散を行うまでは静的戦略をとり，それ以後は，静的戦略によらず，すべての箇所でローカルIUを選択する．

IU台数を変化させた時に，ローカルメモリ参照率がどのように変化するかを，**図7**に示す．最大並列度の世代平均は，90.4であった．静的戦略は，並列度とIU台数の関係によらずに，一定レベルのローカルメモリ参照率を確保している．動的戦略は，IU台数が少ない場合には，静的戦略よりも高いローカルメモリ参照率を示すが，IU台数が多くなるに従って分割が頻繁に行われようになり，ローカルメモリ参照率は低

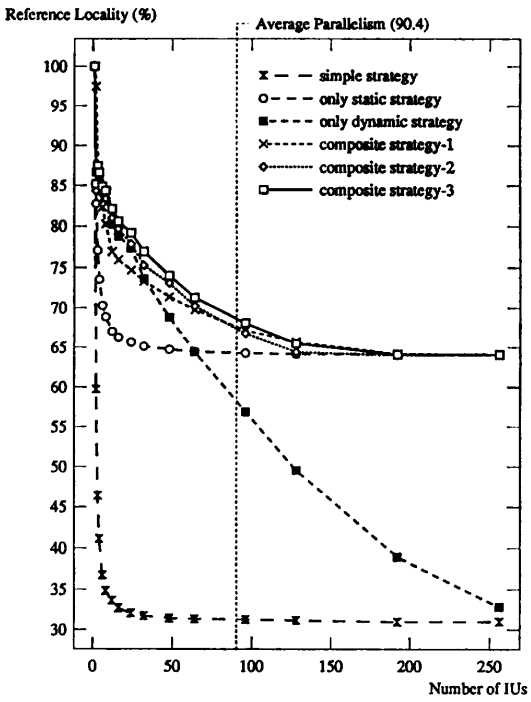

図7 IU台数の変化とローカルメモリ参照率 (6-Queen)

Fig. 7 Number of IUs versus memory reference locality (6-Queen).

下していく．複合戦略はどれも類似の傾向を示すが，複合戦略-1 は制御依存関係に沿った分割の比重が小さいために，IU 台数が少ない場合に動的戦略よりも若干ローカルメモリ参照率が低い．複合戦略の中では，制御依存関係に沿った分割の比重が最も大きい複合戦略-3が，常に最大のローカルメモリ参照率を示している．

図 8 に，世代ごとの最大並列度の変化と，IU 台数が 64 台の時の動的戦略と複合戦略-3 によるローカルメモリ参照率の変化を示す．この図からも，動的戦略は並列度が IU 台数を大きく上回った時にのみ有効に働いていることがわかる．また，複合戦略-3の方はローカルメモリ参照率が大きく落ち込むこともなく，静的戦略と動的戦略の切替がスムーズに行われていることがわかる．

以上より，並列性が IU 台数を大きく上回る場合は動的戦略が，それほど大きく上回らない場合は静的戦略がそれぞれ重要であり，両者の利点を合わせ持つ複合戦略が最も有効であるということがわかる．

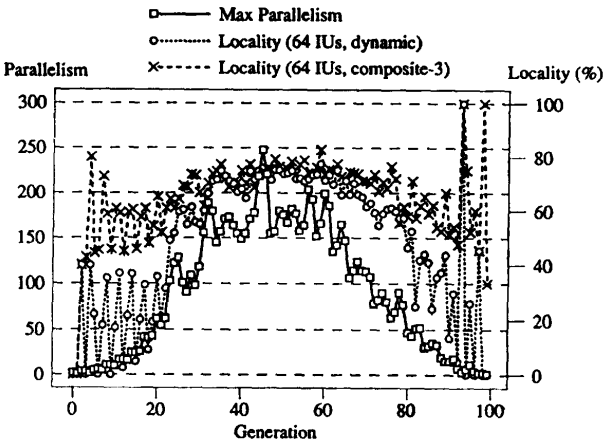

図 8 並列度の変化とローカルメモリ参照率の変化
Fig. 8 Degree of parallelism versus memory reference locality.

7. ま と め

本論文では，並列処理管理の問題点を整理し，PIE 64 の管理プロセッサで実行される並列処理管理カーネルのアーキテクチャについて，負荷分散とスケジューリングを中心に述べた．PIE 64 では負荷分散を，静的負荷分割，動的負荷分割，動的負荷平滑の 3 つのレベルで扱う．並列処理管理カーネルは，このうちの動的負荷分割を担い，過剰な並列性を抑制し，通信量を低減させる．スケジューリングでは，動的な優先度を導入して，ヒープメモリの枯渇の回避と並列性の抽出をはかり，実行開始猶予時間を導入して，同期コストを低減させる．また，静的負荷分割と動的負荷分割の持つ効果が，どのように異なるかを示し，それらを組み合わせた複合方式が最も有効であることを示した．

今後の課題には，

- コンパイル時の静的解析アルゴリズムの開発．
- PIE 64 実機での並列処理管理カーネルの実装．
- PIE 64 上の大規模応用プログラムを用いた評価．
- 管理プロセッサの専用ハードウェア化．

などがある．そして，かつてメモリ管理ユニットハードウェアの導入によって，仮想記憶がオーバレイを駆逐したのと同様に，高並列計算機における負荷分散・スケジューリングの完全な自動化のために導入すべき，「並列処理管理ユニット」の機能と方式を明らかにするのが，本研究の最終的な目標である．

謝辞 本研究は文部省特別推進研究 No. 62065002 の一環として行われた．

参 考 文 献

1) Arvind and Iannucci, R. A.: Two Fundamental Issues in Multiprocessing, Computation Structures Group Memo 226-6, Laboratory for Computer Science, MIT (1987).
2) Smith, B. J.: A Pipelined, Shared Resource MIMD Computer, *Proc. Int'l Conf. on Parallel Processing*, pp. 6-8 (1978).
3) Casavant, T. L. and Kuhl, J. G.: A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems, *IEEE Trans. Softw. Eng.*, Vol. 14, No. 2, pp. 141-154 (1988).
4) Shapiro, E.: Systolic Programming: A Paradigm of Parallel Processing, *Proc. of the Int. Conf. on Fifth Generation Computer Systems*, pp. 458-470 (1984).
5) Takeda, Y., Nakashima, H., Masuda, K., Chikayama, T. and Taki, K.: A Load Balancing Mechanism for Large Scale Multiprocessor Systems and Its Implementation, *Proc. of the Int. Conf. on Fifth Generation Computer Systems*, pp. 978-986 (1988).
6) Ueda, K.: Guarded Horn Clauses, ICOT Technical Report TR-003, Institute for New Generation Computer Technology, Tokyo (1985).
7) Nilsson, M. and Tanaka, H.: Massively Parallel Implementation of Flat GHC on the Connection Machine, *Proc. of the Int. Conf. on Fifth Generation Computer Systems*, pp. 1031-1040 (1988).

8) 坂井：並列計算機におけるスケジューリングと負荷分散，情報処理，Vol. 27, No. 9, pp. 1031-1038 (1986).
9) 新世代コンピュータ技術開発機構第四研究室：KL 1 プログラミング入門編／初級編／中級編，p. 177 (1989).
10) 日高，小池，田中：PIE 64 の並列処理管理カーネルにおけるスケジューリング，負荷分散の検討，第 42 回情報処理学会全国大会論文集，2 H-1 (1991).
11) 日高，小池，舘村，田中：実行プロファイルに基づくコミッティッドチョイス型言語の静的負荷分割手法，情報処理学会論文誌，Vol. 32, No. 7, pp. 807-815 (1991).
12) 小池，田中：分散メモリ並列計算機上でのジェネレーションスキャベンジング GC，並列処理シンポジウム JSPP '90 論文集，pp. 273-280 (1990).
13) 小池，田中：並列推論エンジン PIE 64，bit 臨時増刊並列コンピュータアーキテクチャ，Vol. 21, No. 4, pp. 488-497 (1989).
14) 高橋，小池，田中：並列推論マシン PIE 64 の相互結合網の作製および評価，情報処理学会論文誌，Vol. 32, No. 7, pp. 867-876 (1991).
15) 日高，小池，田中：並列推論エンジン PIE 64 の推論ユニットのアーキテクチャ，コンピュータシステム研究会 CPSY 90-44，電子情報通信学会技術研究報告，Vol. 90, No. 144, pp. 37-42 (1990).
16) 島田，小池，田中：推論プロセッサ UNIRED II：シミュレーションによる評価，並列処理シンポジウム JSPP '91 論文集，pp. 85-92 (1991).
17) 清水，小池，田中：PIE 64 のネットワーク・インタフェース・プロセッサ LSI の詳細，情報処理学会計算機アーキテクチャ研究会資料，87-5 (1991).

(平成 3 年 8 月 5 日受付)
(平成 4 年 1 月 17 日採録)

日高 康雄（正会員）

昭和 38 年生．平成元年東京大学工学部精密機械工学科卒業．平成 3 年同大学大学院工学系研究科情報工学専攻修士課程修了．現在，同博士課程在学中．昭和 59 年（株）創夢設立．現在，同社非常勤取締役．平成 3 年度より日本学術振興会特別研究員．計算機アーキテクチャ，並列処理，CAD 等に興味を持つ．電子情報通信学会，精密工学会，IEEE-CS 各会員．

小池 汎平（正会員）

昭和 36 年生．昭和 59 年東京大学工学部電子工学科卒業．平成元年同大学院工学系研究科情報工学専攻博士課程満期退学．同年東京大学工学部電気工学科助手．工学博士．平成 3 年東京大学工学部電気工学科講師，現在に至る．並列計算機アーキテクチャ，および，並列プログラミング言語に関する研究に従事．本会学術奨励賞受賞．日本ソフトウェア科学会，ACM 各会員．

田中 英彦（正会員）

昭和 18 年生．昭和 40 年東京大学工学部電子工学科卒業．昭和 45 年同大学院博士課程修了．工学博士．同年東京大学工学部講師，昭和 46 年助教授，昭和 62 年教授．昭和 53 年～54 年ニューヨーク市立大学客員教授，現在に至る．計算機アーキテクチャ，並列推論マシン，知識ベース．オブジェクト指向プログラミング，分散処理，CAD，自然言語処理，等の研究を行っている．'計算機アーキテクチャ'，'VLSI コンピュータ I，II'，'ソフトウェア指向アーキテクチャ'（いずれも共著），'情報通信システム' 著．電子情報通信学会，人工知能学会，日本ソフトウェア科学会，IEEE，ACM 各会員．