PIE64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
東京大学 工学部 電気工学科
E-mail: {hidaka,koike,tanaka}@mtl.t.u-tokyo.ac.jp

概要

高並列推論エンジン PIE64 の管理プロセッサが実行するファームウェア－並列処理管理カーネルは，オペレーティングシステムの核に相当するが，細粒度ゆえに重要な並列処理管理に重点をおいている．これには，実行時の並列度・ヒープメモリ残量に応じた負荷分散手法や動的なスケジューリング優先度，実行開始猶予時間などを導入し，プログラマの指示がなくても，的確に負荷分散，スケジューリングを行なうことを目指す．

1 はじめに

高並列計算機において高い台数効果を得るためには，プロセッサ台数に見合う充分な並列性の抽出と並列処理に伴うオーバヘッドの低減が重要である．

一般に，並列処理の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]．我々は，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を重視する．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である．

そこで我々は，通信処理，同期処理，並列処理管理が並列処理オーバヘッドの根本的原因であると考え，それらを本来の計算処理と並列に処理することにより，従来のプロセッサ高速化技術の下で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た．そして，現在研究開発を進めている高並列推論エンジン PIE64 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」を担うというアーキテクチャを採用した．

本論文では，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し，Fleng レベルで記述できない，もしくは，記述するのが適当でない，中核部分の機能を担うが，細粒度並列処理を効果的に支援するために，その中でも特に並列処理管理に重点を置いているのが特徴である (図 1)．プログラミング環境などのオペレーティングシステムの上位の機能は，Fleng のプログラムとして記述され，並列処理管理カーネルの機能には含まれない．

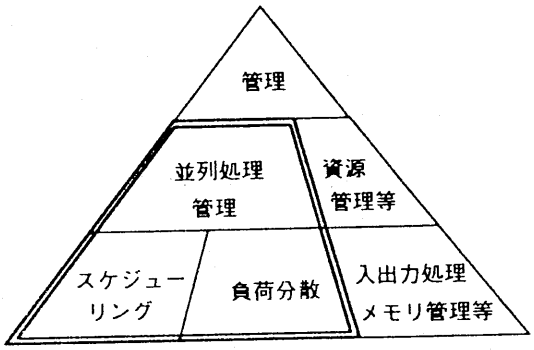

図 1: 並列処理管理

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なわれてきた [2, 5]．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あまりない．このような場合には，プログラマが戦略の指定を行ない，システムはその指定に従うという方法が一般的であるが，これは，プログラマはプログラムの挙動を把握してい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．過剰な並列性を抑制す

The architecture of the parallel processing management kernel of PIE64
Yasuo HIDAKA, Hanpei KOIKE, Hidehiko TANAKA
Department of Electrical Engineering, Faculty of Engineering, The University of Tokyo.

るのは, システムが行なうべきである.

PIE64 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. このうち並列処理管理カーネルが担当する動的負荷分割では, 実行時の並列度等による分割の判断をして, 制御依存関係に沿った分割を行なう. これにより, 過剰な並列性を抑制して通信量を低減させる.

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, 静的に検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.

このように, 並列処理管理カーネルが負荷分散, スケジューリングを支援することによって, プログラマが細粒度プロセスのまとめ方を指定したり, 過剰な並列性を心配する必要をなくし, 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.

2 章と 3 章では, それぞれ PIE64 の主要な記述言語である Fleng と PIE64 について, 簡単に説明する. 4 章では, 並列処理管理に関する定性的な考察を行なう. 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

PIE64 の主要な記述言語である Fleng[4] 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, 並列論理型言語の一つである. 同種の言語で有名なものには, GHC[3] 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.

$$\text{Head} :- \text{Goal}_1, \text{Goal}_2, \cdots, \text{Goal}_n.$$

:- 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一つが選択され, 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.

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

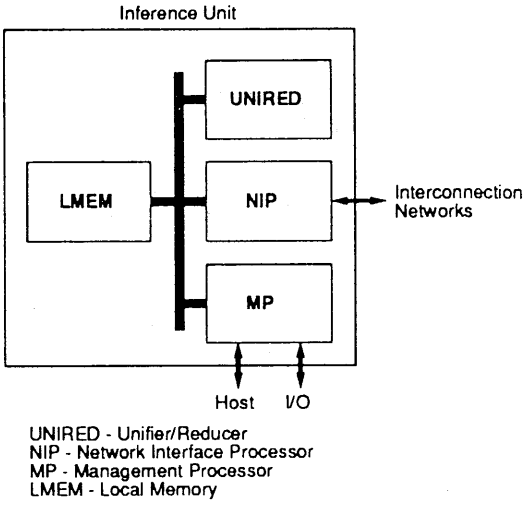

図 2: 推論ユニットの概略

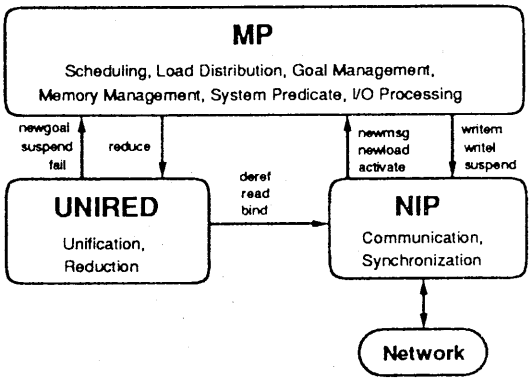

図 3: 推論ユニット内の協調処理モデル

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.

3 並列推論エンジン PIE64

PIE64[8] 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. PIE64 の要素プロセッサは, 推論ユニット (IU – Inference Unit)[10] と呼ばれる. 相互結合網 [9] 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[11] 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, 各ゴールは UNIRED 上の一つのスレ

表 1: 管理プロセッサが発行 / 受信する主要なコマンド

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な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 に行なわせる.
writel	転送先 IU が決定していない場合のゴール転送を NIP に行なわせる. 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.

ory Access time) 型共有メモリを構成する. そして, UNIRED は最大 4 つのスレッドを並行実行し, リモートメモリ参照時のパイプラインの充填をはかっている.

NIP[12] 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.

4 並列処理管理の定性的検討

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るため, 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. 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. 管理のための処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.

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区別するために, 並列処理管理とは, 「並列処理のスレッド毎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, 一つのスレッドの実行は逐次的に行なわれるものとする.

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
2. スケジューリング
3. 同期処理
4. 実行制御
5. アドレス空間の管理
6. コンテクスト, スタックの管理

負荷分散では, 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. スケジューリングでは, 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. 同期処理では, スレッド間の実行の排他制御や待ち合わせなどを行なう. 実行制御では, 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なう. スレッド毎に異なるアドレス空間やコンテクスト, スタックを持つ場合は, それらの管理も必要である.

同期処理, スケジューリング, 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なうべき処理である. 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.

PIE64では, 同期処理の主要な部分は, 単一代入変数に対するsuspend / bind / activateの処理として, NIPにハードウェア化されている. MPは, 一つのスレッドが複数の変数へサスペンドできるようにするために, サスペンドの度に一つのサスペンションレコードを割り当て, 最初のアクティベイト時に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る.

一方, PIE64では, 全てのスレッドは, 全IUに渡る単一アドレス空間のヒープメモリを共有し, スレッド毎のスタックを持たないため, アドレス空間の管理やスタックの管理は必要ない. スレッド毎のコンテクストは, 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自身が行なうため, 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, PIE64の並列処理管理カーネルにおいて特に重要と考えられる, 負荷分散とスケジューリングについて, その要件を述べる.

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

全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二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である場合は, 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過ぎると未定義変数を参照してサスペンドするために, コンテクストスイッチが発生する. 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なわないと, 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求駆動型や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有す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コントロールを行なうのが望ましい.

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なう. 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.

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な

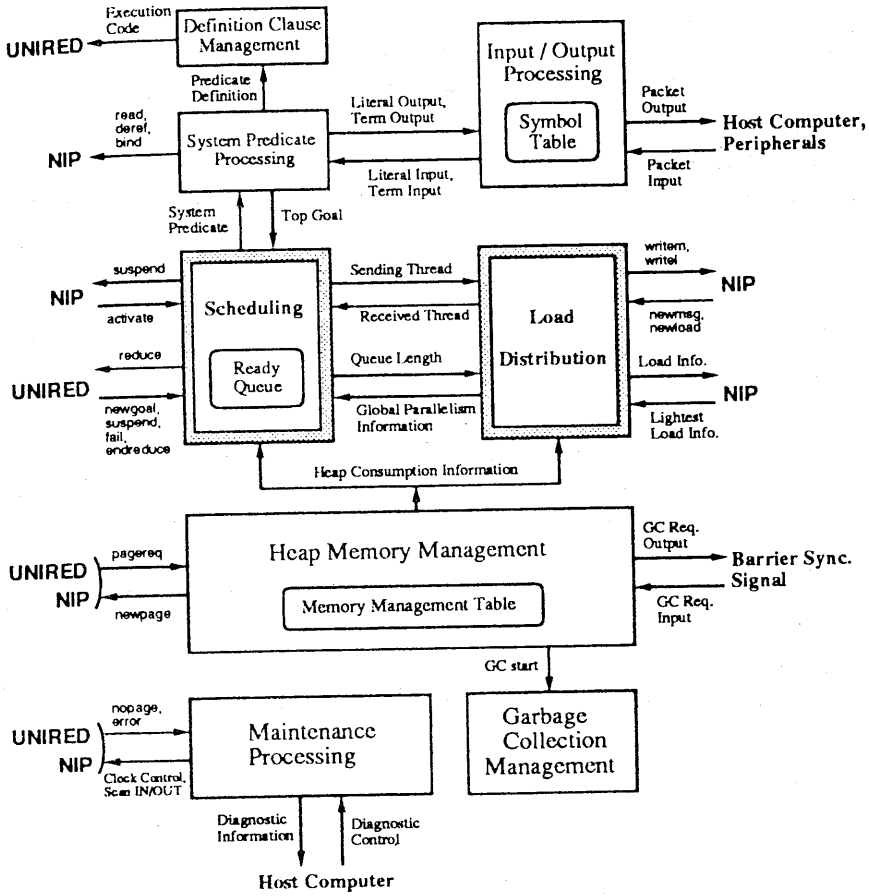

図 4: 並列処理管理カーネルの構成

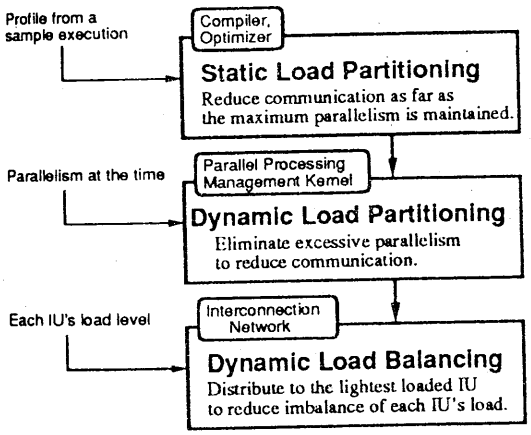

図 5: PIE64 における三階層の負荷分散

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なう [7].

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ない, プログラム実行の計算木 (制御依存関係) に沿った分割を行なう. 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 [9] 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 が指定されている箇所で負荷分割を行なう.

- 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が指定されている箇所では，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，予備評価の節で述べる．

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，ヒープメモリの確保はUNIRED上で行なわれるため，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二つを用意し，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．

- ガベー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．

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全て確定してから，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．

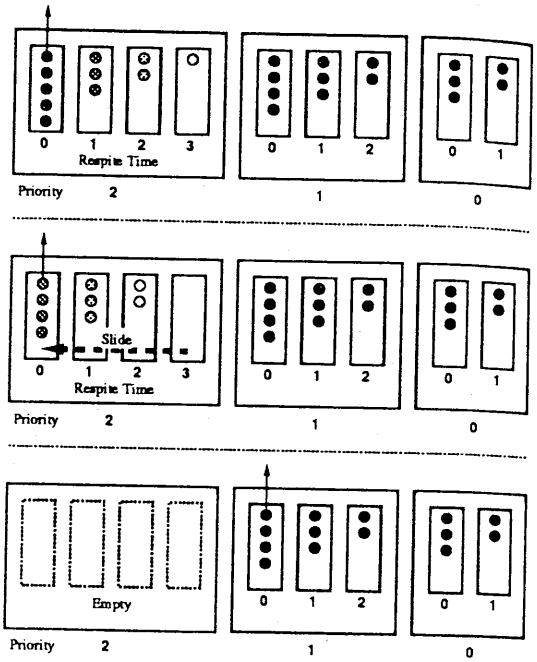

図 6: 優先度毎のスライド型レディーキュー

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みる．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させる．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毎に設けて構成する．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他に，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なわれ，ワード単位の割り当ては，UNIRED，NIP内部で行なわれる．

ヒープメモリのガベージコレクションは，全IUで同期して行なう一括型のガベージコレクションである．ガベー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なう．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で行ない，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，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，述語毎にコンパイルされたUNIREDの実行コードの管理を行なう．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なったりする．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

PIE64で行な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二つは，共に通信量を低減させる働きを持つ．この二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なった．

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毎に，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，二つのゴールキューを世代毎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．サンプルプログラムには，6-Queenを用いた．測定には，以下の6種類の戦略を使用した．

1. 単純戦略：ヒープ上のデータは全てローカルIU上にとる．リカージョンのゴール一つをローカルIUで実行し，それ以外のゴールを実行するIUは，一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，次の世代で行なう自動負荷分散の最大回数とする．最大回数の自動負荷分散を行なうまでは単純戦略をとり，それ以後は全てローカルIUを選択する．
4. 複合戦略-1：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，次の世代で行なう自動負荷分散の最大回数とする．最大回数の自動負荷分散を行な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：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，次の世代で行なう自動負荷分散の最大回数とする．最大回数の自動負荷分散を行な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には全てローカル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：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

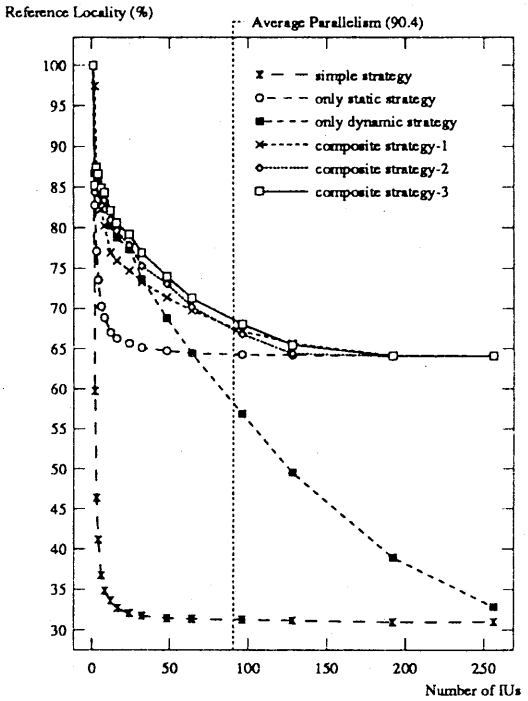

図7: IU台数の変化とローカルメモリ参照率 (6-Queen)

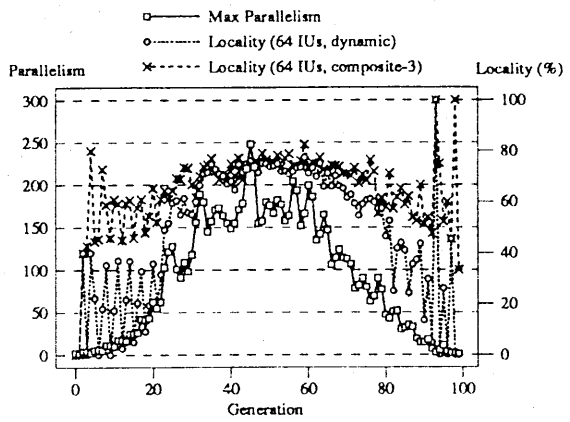

図 8: 並列度の変化とローカルメモリ参照率の変化

行なう自動負荷分散の最大回数とする. 最大回数の自動負荷分散を行なうまでは静的戦略をとり, それ以後は, 静的戦略によらず, 全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なわれるようになり, ローカルメモリ参照率は低下していく. 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, 世代毎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な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.

今後の課題には, 並列処理管理カーネルの実価, 改良などがある.

なお, 本研究は文部省特別推進研究 No.62065 一環として行なわれた.

参考文献

[1] Arvind and Iannucci, R.A.: Two Fund tal Issues in Multiprocessing, *Compu Structures Group Memo 226-6*, Laborat Computer Science, MIT, (1987).

[2] Casavant, T.L. and Kuhl, J.G.: A Tax of Scheduling in General-Purpose Distri Computing Systems, *IEEE Trans. So Eng.*, Vol.14, No.2, pp.141-154 (1988).

[3] Ueda, K.: Guarded Horn Clauses, *ICOT nical Report TR-003*, Institute for New eration Computer Technology, Tokyo (

[4] Nilsson, M. and Tanaka, H.: Massively lel Implementation of Flat GHC on the nection Machine, *Proc. of the Int. Co Fifth Generation Computer Systems*, pp 1040 (1988).

[5] 坂井: 並列計算機におけるスケジューリ負荷分散, 情報処理, Vol.27, No.9, pp.103 (1986).

[6] 日高, 小池, 田中: PIE64 の並列処理管理ルにおけるスケジューリング, 負荷分散の第 42 回 情報処理学会 全国大会, 2H-1 (19

[7] 日高, 小池, 舘村, 田中: 実行プロファイルくコミッティッドチョイス型言語の静的負手法, 情報処理学会論文誌, Vol.32, No.7, (

[8] 小池, 田中: 並列推論エンジン PIE64, bit 刊 並列コンピュータアーキテクチャ, No.4, pp.488-497 (1989).

[9] 高橋, 小池, 田中: 並列推論マシン PIE64 結合網の作成および評価, 情報処理学会 Vol.32, No.7, (1991).

[10] 日高, 小池, 田中: 並列推論エンジン PIE6 論ユニットのアーキテクチャ, コンピューテム研究会 CPSY90-44, 電子情報通信学研究報告, Vol.90, No.144, pp.37-42 (199

[11] 島田, 小池, 田中: 推論プロセッサ UNIR シミュレーションによる評価, 並列処理シウム JSPP'91, 情報処理学会, (1991).