実行プロファイルに基づくPIE64の負荷分散方式

日高 康雄　小池 汎平　舘村 純一　田中 英彦
{hidaka,koike,tatemura,tanaka}@mtl.t.u-tokyo.ac.jp
東京大学 工学部

概要

並列計算機でプロセッサ台数に見合うだけの速度向上を得るためには，負荷分散の方式が一つの鍵となる．本稿では，コミッティッドチョイス型言語であるFlengのプログラムを並列推論マシンPIE64で実行する際の負荷分散戦略を，ゴールとデータの履歴を保持して得た実行プロファイルを用いて最適化する手法について述べる．また，ワークステーション上で行なった本手法の評価についても述べる．

1　はじめに

並列計算機においてプロセッサ台数に見合うだけの速度向上を得るためには，並列処理に伴うオーバヘッドを抑えつつ，どのようにして並列度を向上させるかという，負荷分散の方式が一つの鍵となる．我々が開発中の並列推論エンジンPIE64[1] – Parallel Inference Engine – は，コミッティッドチョイス型言語であるFleng[5]を高速実行する並列計算機で，そのプロセッサ間相互結合網[2]は，プロセッサ間の距離が均等であるという特徴があり，また，負荷が最小のプロセッサを通信時に自動的に選択する自動負荷分散機能を持っている．

負荷分散の問題には，処理をどのように分割し，プロセッサに割り当てるかという問題と，各プロセッサの負荷をどのようにバランスさせるかという問題がある．PIE64では，後者の問題には相互結合網の自動負荷分散機能を使う．本稿では，前者の問題を扱う．

一方，PIE64は分散メモリ構成をとっているため，リモートメモリの参照は，ローカルメモリの参照よりもコストが高い．このため，負荷を分割する際には，ローカルなメモリ参照の割合を多くするように分割する必要がある．

一般に，ローカルメモリ参照の割合を高めると，プログラムの並列度は低下する．極端な例として，1台のプロセッサでプログラムを実行する場合を考えると，メモリ参照は100%ローカル参照になるが，並列度は1でしかない．すなわち，負荷分散戦略の最適化においては，

- ローカルメモリ参照の割合を高くする．
- 並列度を下げない．

ことに関するトレードオフを，プログラムを実行する並列計算機のアーキテクチャに合わせて求めることが重要である．特に，大規模並列計算機で高い並列処理効果を得るためには，後者をより重視する必要がある．

このようなトレードオフでは，プログラム中のどのメモリ参照をローカルにして，どのメモリ参照をリモートにするかを決めなければならないため，プログラムの各部の実行の頻度など，プログラムの動的な特性を考慮する必要がある．しかし，コンパイル時の静的な解析のみで，このような動的な特性を取り出すのは困難である．

本稿では，実行プロファイルに基づいて負荷分散戦略を最適化する方法について述べる．初めは最適化を行なわずに，プログラムをプロファイルモードで実行する．実行時間の長いプログラムは，問題の規模を小さくして実行する．そして，実行時のプログラムの動的な特性をプロファイラによって統計的に収集する．次に，この情報を用いて最適化した負荷分散戦略をコンパイラにフィードバックして，プログラムを再コンパイルする．

2節では，Flengについて，3節では，PIE64について簡単に説明する．4節では，Flengのプログラムの負荷分散戦略として，本稿でどのような戦略を想定しているかについて述べる．5節では，Flengのプロファイラの動作について述べる．6節では，プロファイラの出力データに基づく負荷分散戦略の最適化と，その評価について述べる．

The load distribution method based on the execution profile for PIE64
Yasuo HIDAKA, Hanpei KOIKE, Jun'ichi TATEMURA, Hidehiko TANAKA
University of Tokyo, Faculty of Engineering.

2 Fleng

Flengは，コミッティッドチョイス型言語，並列論理型言語と呼ばれる並列記号処理向けのプログラミング言語の一つである．

プログラムの実行は，トップゴールが与えられることで開始する．与えられたゴールに対して，各クローズのヘッド部との間でそれぞれユニフィケーションが試みられる．ユニフィケーション可能なものが見つかると，そのうち一つのクローズがコミットされ，元のゴールはそのクローズのボディーゴールによってリダクションされる．プログラムの実行は，ゴールとクローズの間でユニフィケーションとリダクションを繰り返すことによって進められる．各ゴールに対するユニフィケーションやリダクションは並列に行なわれる．

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，そのユニフィケーションは中断され，ゴールはサスペンドする．別のゴールの実行において，その未定義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)と呼ばれる．

相互結合網[2]は，両系統とも回線交換方式の3段の多段網である．この相互結合網の特徴として，自動負荷分散機能がある．これは，接続側が接続先のIUを指定せずに，相互結合網が負荷最小のIUを自動的に選択し，そこに接続するという機能である．接続側が接続先のIUを指定する普通の接続も可能である．

2系統の相互結合網は，それぞれPAN (Process Allocation Network) とDAN (Data Allocation Network) と呼ばれる．これらのハードウェアは全く同じものである． PANとDANにはそれぞれ，IUの稼働状況に関する負荷情報とヒープメモリの使用量に関する負荷情報を流す．そして，ゴールの負荷分散にはPANを，データの負荷分散にはDANを用いる．接続先のIUがわかっていて，自動負荷分散機能を使わない場合は，PAN, DANのどちらを使っても構わない．

IUは，UNIRED(Unifier-Reducer)[4], NIP(Network Interface Processor)[3], 管理用マイクロプロセッサ，メモリとからなっている． UNIREDは，Flengのプログラムを実行するための専用プロセッサである．NIPは，相互結合網とのインタフェースを持ち，IU間の通信，Flengのプロセス間同期を支援するコプロセッサである．2系統の相互結合網の経路制御側と被制御側のそれぞれを担当するために，IU1台につき4個のNIPが使われる．管理用マイクロプロセッサにはSPARCを用いる．これには，ホストやIOとのインタフェースがつけられ，スケジューリングなどのゴール管理，システム述語の実行のほか，メモリ管理，ホストインタフェース，入出力インタフェースなどの様々な機能を担う．これら6個のプロセッサはコマンドバスによって結ばれ，コマンドとリプライのやり取りにより協調実行する．

UNIREDは，Flengプログラムの実行におけるユニフィケーション，リダクションを効率良く実行する命令セットを持っている．そして，ほとんどのメモリ参照命令において，ローカルメモリの参照とリモートメモリ(別のIUのメモリ)の参照を自動的に判別し，ローカルであれば直接メモリをアクセスし，リモートであればNIPにリモートメモリをアクセスするコマンドを発行する．すなわち，UNIREDの機械語レベルでは，ローカル参照とリモート参照を区別する必要はなく，PIE64のメモリは仮想的に共有メモリであるとみなすことができる．

リモートメモリをアクセスするためのNIPのコマンドには，通常のメモリアクセスと同様にアドレスを指定する形式の，`read`, `write`コマンドのほか，リモートヒープ割り当てを実現する`writem`, `writel`コマンドがある．`writem`, `writel`コマンドでは，書き込みサイズはコマンドと共に指定するが，書き込みアドレスは指定しない．書き込み先のIU上のNIPが，書き込みサイズを受け取ってヒープメモリを割り当て，書き込みアドレスを決定する．書き込み終了後，書き込んだアドレスがリプライとして返される． `writem`では，書き込み先のIUをコマンドと共に指定するが，`writel`では，書き込み先のIUを自動負荷分散機能で決定する．この`writel`, `writem`コマンドを使うことによって，指定したIUや負荷最小のIUに対するゴールの転送や，構造データなどのリモートヒープ割り当てを，効率良く実行することができる．

次節以後でFlengプログラムの負荷分散戦略について述べていくが，そこではメモリ参照をローカルにすることを考察する．それに先だって，Flengプログラムの実行中に発生するメモリ参照にどのようなものがあるかを，NIPのコマンドと対応させて表1にまとめておく．

4 Flengのプログラムにおける負荷分散戦略

本節では，本稿で想定している負荷分散戦略がどのようなものであるかを述べる．

表 1: Flengプログラムの実行中に発生するメモリ参照

項　目	説　明	対応するNIPのコマンド
デレファレンス	リモート変数の内容をたぐる.	deref
構造データのコピー	リモートリスト/リモートベクタをローカルメモリにまとめてコピーする.	read2, readn, readx
ゴール, 構造データの作成	一時的にローカルメモリに作成し, リモートメモリにコピーする. コピー先は, 後からわかる.	write2, writeln, writelx, writem2, writeml, writemx
変数の作成	UNDEFをリモートメモリに書き込む. 書いた先は, 後からわかる.	write1, writem1
変数へのバインド	リモート変数に値をバインドする.	bind
サスペンド	リモート変数へのサスペンドゴールの登録.	suspend
アクティベイト	ゴールのアクティベイトの通知	activate

4.1 負荷分散ポイントと負荷分散戦略

Flengプログラム中の負荷分散を行ない得る箇所を, 負荷分散ポイントと呼ぶ. これはすなわち, IUを指定する箇所でもある. 負荷分散ポイントとして, 次の二つを考える.

1. ヒープメモリを確保する箇所. (ヒープメモリを確保するIUを指定する.)
2. ゴールを生成する箇所. (ゴールを実行するIUを指定する.)

負荷分散ポイントは, 一つのクローズあたり複数箇所存在する.

この負荷分散ポイントのそれぞれに対して戦略を定める. 負荷分散ポイントにおける戦略は, 静的に決定し, コミットの度に常に同じ戦略を取る. こうすることによって, 戦略を考える際の負荷分散ポイントの数は, プログラムの大きさに比例し, 計算木の大きさにはよらないものとなる.

戦略としては, 次の4種類を考える.

戦略A. 自動負荷分散によって, 負荷最小のIUを選択する.

戦略B. ローカルIUを選択する.

戦略C. リダクション前のゴールの引数として得られるポインタや, 引数の構造データの要素として得られるポインタの指す[1] IUを選択する. (ユニフィケーションの時にゴールからたどることのできる変数や構造データと同じIUを選択する.)

戦略D. 同じクローズ内のヒープメモリに関する別の負荷分散ポイントにおいて, 自動負荷分散によって選択されたIUと同じIUを選択する.

一つの負荷分散ポイントにおける戦略の数は, クローズの大きさに比例し, クローズ毎に一定である.

次に, naive reverseの一つのクローズを用いて, 負荷分散ポイントと戦略の例を示す.

```
nreverse([X|L],R) :-
  append(R1,[X],R), nreverse(L,R1).
```

このクローズの負荷分散ポイントは, 次の4箇所である.

1. 変数R1をどのIUのヒープメモリに置くか.
2. リスト[X]のセルをどのIUのヒープメモリに置くか.
3. ゴールappend(R1,[X],R)をどのIUで実行するか.
4. ゴールnreverse(L,R1)をどのIUで実行するか.

[1] クローズヘッド側が変数である場合は, デレファレンスしていない値を用いる. この値は, ポインタであったり, なかったりする場合があるため, 実行時にポインタであるかどうか調べ, ポインタでなければローカルIUを選択する, などの処理が必要である. 一方, クローズヘッド側が構造データである場合は, ユニフィケーションにおいて必ずデレファレンスを行なうため, デレファレンス後の値を用いる.

また，採り得る戦略には，次の 8 つがある．

1. 自動負荷分散によって決まる負荷最小の IU(戦略 A).
2. ローカル IU(戦略 B).
3. リスト [X|L] のセルと同じ IU(戦略 C).
4. 変数 X と同じ IU(戦略 C).
5. 変数 L と同じ IU(戦略 C).
6. 変数 R と同じ IU(戦略 C).
7. 自動負荷分散で割り当てた変数 R1 と同じ IU(戦略 D).
8. 自動負荷分散で割り当てたリスト [X] のセルと同じ IU(戦略 D).

負荷分散戦略は，'...@*something*' という形式のアノテーションを，以下の規則に従ってプログラムに付加して表記する．

- アノテーションは，負荷分散ポイントに対応する，ボディーゴールやヒープメモリを使用する変数または構造データの右側と，戦略として参照する，ヘッド部の変数または構造データの右側に付加する．
- 戦略 C が参照するヘッド部の変数または構造データには，'...@on(*id*)' というアノテーションを付加する． *id* には，クローズ内の '@on(*id*)', '@any(*id*)' の間で重複しないシンボルまたは数値を用いる．
- 戦略 A(自動負荷分散) を指定する負荷分散ポイントには，'...@any', または '...@any(*id*)' というアノテーションを付加する．後者は，自動負荷分散の結果を他の負荷分散ポイントの戦略 D が参照する場合に用いる． *id* には，クローズ内の '@on(*id*)', '@any(*id*)' の間で重複しないシンボルまたは数値を用いる．
- 戦略 B(ローカル IU) を指定する負荷分散ポイントには，'...@local' というアノテーションを付加する．
- 戦略 C または戦略 D を指定する負荷分散ポイントには，'...@to(*id*)' というアノテーションを付加する． *id* には，'@on(*id*)', '@any(*id*)' で用いたのと同じシンボルまたは数値を指定する．
- 何も指定されない負荷分散ポイントには，システムが適当な戦略を決定する．

例として，アノテーションを付加した naive reverse のプログラムを図 1に示す．

```
append([A@on(iu1)|X@on(iu2)],Y,Z@on(iu3)) :-
    Z = [A|Z1@to(iu1)]@to(iu3),
    append(X,Y,Z1)@to(iu2).
append([],Y,Z) :- Z = Y.

nreverse([X|L@on(iu1)],R) :-
    append(R1@local, [X]@local, R)@local,
    nreverse(L,R1)@to(iu1).
nreverse([],R) :- R = [].
```

図 1: 負荷分散戦略アノテーションを付加した naive reverse のプログラム．

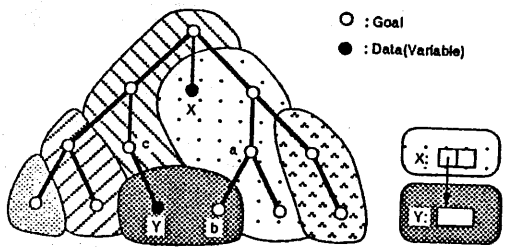

図 2: 計算木と IU の割り当ての関係

4.2 負荷分散戦略とローカルなメモリ参照の関係

戦略 C, 戦略 D を導入することにより，計算木中の離れた場所で作成されるデータとゴールを同じ IU に割り付けることが可能になる．そのような例を図 2に示す．図中の木は，計算木とヒープ上のデータ (または変数) を示したものである．白い円はゴールを，黒い円はヒープ上のデータ (または変数) を表している．また，同じ背景パタン上のゴール，データ，変数は，同じ IU 上に存在することを表している．この例では，計算木の異なる枝にあるゴール b とデータ Y を同じ IU に割り当てていることがわかる．これは以下のような前提と戦略によるものである．すなわち，ゴール a とゴール c は共有変数 X を持っており，c が X を [Y|Z] にバインドする． a は X のバインドを待ってリダクションし，ゴール b を生成する．b を実行する IU は，X の car のポインタの指す IU というように戦略がとられており，その結果 b と Y は同じ IU に割り当てられる． b の引数に Y が渡されて，b の実行時に Y を参照する場合，そのメモリ参照はローカル参照になる．

ここで注意すべきことは，a のリダクションにおいては Y へのポインタを調べるだけで，Y の内容を参照しない，つまり，Y をデレファレンスしない点である．Y の内容が参照されるのは，b の実行においてであり，これは Y と b のある IU でローカルに行なわれる．

一方，Y が構造データにバインドされる場合など，Y

をデレファレンスしなければ，その内容を参照できないことがある．すると，単にaのcarの値からbを実行するIUを定めただけでは，bの実行時にYの内容をローカルに参照できるとは限らない．このような場合は，Yのバインドを行なう側においても，バインド前の変数とバインドする構造データが同じIUになるように，戦略をとっておく必要がある．

5 Flengのプロファイラ

Flengのプロファイラは，負荷分散戦略を最適化するための統計的な情報を出力する．このためプロファイラ内部では，各負荷分散ポイントにおける戦略を固定せず，様々な負荷分散戦略の可能性を同時に調べていく．そして，どのような戦略を取ると，ローカルなメモリ参照が何回発生し，また，並列度の低下がどの程度起きるかを出力する．

5.1 ゴールおよび構造データ，変数の履歴

ゴールや構造データ，変数(以下では，これらをまとめて「オブジェクト」と表記する)が存在するIUが，負荷分散戦略によってどのように決まっているかを考える．簡単のために，IUの数は無限大で，自動負荷分散のたびに新たなIUを割り当てると仮定する．自動負荷分散によって割り当てられるIUを「仮想IU」と呼ぶ．

オブジェクトが存在するIUは，それが生成されるより前のある時点にある負荷分散ポイントにおける戦略Aの自動負荷分散で割り当てられた仮想IUであるか，トップゴール(または，その引数)が存在していたIUであるかのどちらかである．後者は例外的な場合であるので，以下では考慮しない．前者において，オブジェクトがある仮想IU上に存在するためには，仮想IUが割り当てられてから，そのオブジェクトが生成されるまでの間の各所の負荷分散ポイントで，その仮想IUを継承するような戦略(B,C,D)がとられていなければならない．もし途中の負荷分散ポイントで別の戦略を取っていたならば，そのオブジェクトは，別の仮想IU上に存在すると考えられる．

ここで，オブジェクトの「履歴」を導入する．履歴は，負荷分散戦略の条件と，その条件が満たされる時にオブジェクトが存在する仮想IUを示すものである．プロファイラは負荷分散戦略を固定しないため，一つのオブジェクトは，複数の履歴を持つ．

履歴は，仮想IUの識別番号と，そのオブジェクトの生成までその仮想IUを継承するために必要な，一連の負荷分散ポイントにおける戦略のAND条件とからなる．そして，AND条件の中に，同じ負荷分散ポイントの条件が二つ含まれることはない．なぜなら，前述したように，一つの負荷分散ポイントにおいて，異なる二つの戦略をとることはないからである．

一方，自動負荷分散からオブジェクトの生成までの間には，リカージョンなどによって同じ負荷分散ポイントが複数回実行され得るが，それらの条件は，AND条件の一つの項に縮退する．

プロファイラは，オブジェクトの履歴を以下のように管理する．

- オブジェクトのアドレスをキーとするハッシュテーブルを使って，履歴を保持する．
- 一つのオブジェクトの持つ複数の履歴は，仮想IUの識別番号をキーとするsorted listとして表す．
- 履歴の持つAND条件は，クローズの番号と負荷分散ポイントの番号をキーとするsorted listとして表す．
- リダクションが行なわれると，コミットしたクローズ上の負荷分散ポイントの数だけ，新たな仮想IUの識別番号を割り当て，それと元のゴールの履歴，ゴールから参照可能な構造データ，変数の履歴とを基に，新しいオブジェクトの履歴(複数)を算出し，ハッシュテーブルに登録する．履歴の算出時には，同じ負荷分散ポイントの戦略を二つ含めないようにする．但し，計算量を抑えるために，AND条件の項数が多くなった履歴は省く．
- ガベージコレクションによるアドレス変更に従って，存命中のオブジェクトの履歴を別のハッシュテーブルに移す．ガベージコレクション終了後，元のハッシュテーブルに残っている履歴の使用メモリを解放し，ハッシュテーブルを取り換えて処理を続行する．

5.2 ローカルメモリ参照の条件

あるゴールの実行中のメモリ参照がローカルであるためには，そのゴールの存在する仮想IUと，参照した構造データ，変数の存在する仮想IUが同じであれば良い．

ゴールと構造データ(または変数)が存在する仮想IUが同じであるための条件は，それぞれが持つ一連の履歴から，同じ仮想IUを持つ履歴の組み合わせを求め，その二つの履歴のAND条件から自動負荷分散の項を除いて作る．このような条件は，一般に複数求められる．また，同じ負荷分散ポイントの戦略が二つになるような条件は省く．

自動負荷分散の項を除くのは，仮想IUが同じであるための条件には必要ないからである．同じオブジェクトの履歴のAND条件同士は排他的であるが，ここで求める条件は自動負荷分散の項を除くために，条件の間に含意関係が生じる．ローカルなメモリ参照の条件の場合は，帰結の側の条件は冗長であるから省く．

5.3 並列度が低下する条件

並列度の低下は，同時に実行可能な複数のゴールが同じ仮想IUに存在する時に発生する．

並列度の低下の条件を調べるために，アクティブゴールキューをエンキュー用とデキュー用の二つに分け，デキュー用のゴールキューが空となる度にそれらを交換する．キューの交換から次の交換までの期間を世代と呼び，キューの交換を世代交代と呼ぶことにする．そして，世代交代の際に，アクティブゴールの全ての組合せについて，それらが同じ仮想IUに存在する条件を，ローカルメモリ参照の条件と同様に求める．但し，並列度の高い時の計算量を抑えるため，先に全てのゴールから同じ仮想IUの履歴を集めておき，次にそれらの組合せから条件を求めるようにする．

一方，上述のように調べただけでは，全ての並列度の低下が同じように扱われてしまう．計算木上の近傍のゴールに対しても，同じ仮想IUに存在することを回避しようとすると，メモリ参照のローカリティを上げることができなくなる．

そこで，並列度低下の猶予期間を設ける．仮想IUの識別番号と共に，それを割り当てたリダクションの世代番号を持たせる．並列度の低下の条件を調べる際に，仮想IUが割り当てられてから，猶予期間の過ぎていない仮想IUにおける条件は，並列度の低下の条件から省くことにする．メモリ参照のローカリティを上げることと，並列度の低下を避けることのトレードオフは，適切な猶予期間を設定することに相当すると考えられる．

5.4 プロファイラの出力

プログラムの実行に沿って，ローカルなメモリ参照の条件，並列度の低下の条件を求め，その出現回数をカウントする．これには，条件をキーとするハッシュテーブルを使う．実行終了後，それらの条件とその出現回数を出力する．

図3に，出力データの一部を示す．行頭の数字は，その行で示す条件の出現回数である．[数-数-数]は，ある負荷分散ポイントの戦略の条件であり，3つの数字はそれぞれ，クローズの番号，クローズ内の負荷分散ポイントの番号，戦略の番号を意味する．ローカル参照の条件と並列度の低下の条件について，それぞれこのような形式のデータが出力される．

```
171 : [10-1-5] [10-7-7] [11-1-7]
361 : [10-1-1] [14-0-4]
38 : [10-3-3] [10-8-2] [14-0-4] [14-2-5]
434 : [7-5-5] [10-0-7] [10-4-4] [14-2-5]
2046 : [10-0-12]
26 : [7-1-3] [7-5-8] [10-3-4] [10-5-1]
28 : [7-0-9] [7-2-7] [7-5-3]
73 : [10-3-8] [10-5-2] [14-0-10]
```

図3: プロファイラの出力データの一部

6 負荷分散戦略の最適化とその評価

前節までに述べた負荷分散戦略，プロファイラの有効性を調べ，自動的な最適化アルゴリズムの指針を得るために，開発中のPIE64に先だって，ワークステーション上のFlengインタプリタを用いた評価を行なった．負荷分散戦略の最適化は，プロファイラの出力データを見るための簡単なプログラムを用いて，人手で行なった．このプログラムは，フィルタ，ソータ，ページャの機能を持つもので，以下の特徴を持つ．

- 負荷分散ポイントとその戦略を指定し，それに反する条件を除き，確定した条件の[，]を省いて表示する．
- 負荷分散ポイントを指定し，その負荷分散ポイントの条件を含むものだけを表示する．
- ソートのキーには，出現回数(降順)，未確定の条件数(昇順)を用いる．第一キーには，どちらを選ぶこともできる．

このプログラムを用いて，以下の手順で最適化を行なった．

1. ローカル参照の条件から，どの負荷分散ポイントの戦略を決めるかを考える．
2. 1で選んだ負荷分散ポイントを指定し，関連するローカル参照の条件，並列度低下の条件を見る．
3. 並列度低下の条件が全て確定しないような戦略で，ローカル参照が最も多く，条件の項数の少ないもの[2]を選び，その戦略を指定する．

[2]履歴の計算でAND条件の項数の多いものを省略できるのは，このためである．

```
qu(N,A) :-
    gen(N,L@local)@any, qu(L,[],[],A,[])@any.

gen(N,L) :- gen(true,N,L)@any.

gen(true,N,L) :-
    L = [N|L1@local]@local, sub(N,1,N1@local)@any,
    gt(N1,0,R@any(1))@any,  gen(R,N1,L1)@to(1).
gen(false,N,L) :- L = [].

qu([P|Lu@on(1)],Ls,Lp@on(2),A0,A) :-
    append(Lu,Ls,Lr@to(3))@to(3),
    check(P,1,Lr,Lp,Lp,A0,A1@any(3))@to(2),
    qu(Lu,[P|Ls]@to(1),Lp,A1,A)@to(1).
qu([],[P|L],Lp,A0,A) :- A0 = A.
qu([],[],Lp,A0,A) :- A0 = [Lp|A]@local.

check(P,D,L,[Q|Lp0],Lp,A0,A) :-
    add(Q,D,Sum@any(1))@to(1),
    equal(Sum,P,R1@any(2))@to(1),
    sub(Q,D,Dif@to(2) )@to(2),
    equal(Dif,P,R2@to(2) )@to(2),
    chk(R1,R2,P,D,L,Lp0,Lp,A0,A)@to(2).
check(P,D,L@on(1),[],Lp,A0,A) :-
    qu(L,[],[P|Lp]@local,A0,A)@to(1).

chk(true,_,P,D,L,Lp0,Lp,A0,A) :- A0 = A.
chk(_,true,P,D,L,Lp0,Lp,A0,A) :- A0 = A.
chk(false,false,P,D,L@on(1),Lp0,Lp,A0,A@on(2)) :-
    add(D,1,D1@to(2))@to(1),
    check(P,D1,L,Lp0,Lp,A0,A)@any.

append([],Y,Z) :- Z = Y.
append([A|X],Y,Z) :-
    append(X,Y,Z1@any(1))@to(1), Z = [A|Z1]@local.
```

図 4: 負荷分散戦略を最適化した n-queen のプログラム

4. ローカル参照の条件に未確定な条件がなくなるまで, 1 から 3 を繰り返す.
5. 残った負荷分散ポイントに対して, 並列度低下の条件が全て確定する場合は自動負荷分散の戦略を, そうでなければローカル IU の戦略を指定する.

このようにして最適化した n-queen のプログラムを図 4に示す.

一方, 最適化した負荷分散戦略を評価するために, プロファイラに評価モードを設けた. 評価モードでは, 負荷分散戦略を指定したものに固定して, ゴール, 構造データ, 変数が存在する仮想 IU(常に唯一つに定まる) を追跡する. そして, メモリ参照が起きた時は, 実行中のゴールと参照した構造データ変数が存在する仮想 IU を比較し, 一致すればローカル参照とする. また, 世代交代の際には, 異なる仮想 IU に存在するアクティブゴールの個数を, その世代における並列度とする. 平均並列度は, IU 台数を無制限とした場合の値と, 64 台に制限した場合の値を求める. 前者は, 全世代の並列度を単純に平均した値であり, 後者は, 並列度 (=n) が 64 を超える世代を, 並列度 64 が n/64 世代だけ続くものと見なして得られる平均値である. IU 台数を制限すると, 仮想 IU が異なっていても同じ IU に割り付けられる場合があるが, それはここでは考慮しない.

最適化した戦略の評価の対照として, 以下の単純戦略を用いる.

- 構造データ, 変数は全てローカル IU にとる.
- リカージョンのゴール一つは, ローカル IU で実行し, それ以外のゴールを実行する IU は, 自動負荷分散で決める.

評価のプログラムには, Primes と Queen を用いた. プロファイラのデータは, Primes-100, 6-Queen に対してとり, それを元に戦略を決定した. プロファイラのパラメタには, 履歴の AND 条件の項数は 3, 並列度低下の猶予期間は 2 を指定した. プロファイラの実行に要した時間は, SUN4-260 を使用して, Primes-100 で 235.4 秒, 6-Queen で 3875.9 秒であった. 決定した戦略の評価は, Primes-100, 6-Queen のほか, Primes-1000, 8-Queen に対しても行なった.

ローカルメモリ参照に関する結果を表 2に示す. また, 6-Queen, 8-Queen についての並列度に関する結果を図 5, 図 6に示す. ローカルメモリ参照の割合は, 単純戦略では 35% 前後であるが, 最適化によって 65% 前後まで上げることができた.

6-Queen の平均並列度は, IU 台数が無制限の場合に, 90.4(単純戦略) と 66.9(最適戦略) であり, 64 台に制限した場合に, 53.0(単純戦略) と 48.6(最適戦略) であった. また, 8-Queen の平均並列度は, IU 台数が無制限の場合に, 1107.8(単純戦略) と 806.1(最適戦略) であり, 64 台に制限した場合に, 63.6(単純戦略) と 63.3(最適戦略) であった. 単純戦略では, リカージョン以外のゴールには新しい仮想 IU を割り当てるため, その並列度は, 最大の並列度を示すことを考慮すると, この程度の並列度の低下は相応だと考えられる.

戦略の最適化は, 6-Queen, Primes-100 のプロファイルデータを用いて行なったが, その戦略は 8-Queen, Primes-1000 においても同様に有効であった. これは, これらのプログラムでは, 各クローズのコミット回数が, 問題の規模によらず同じような傾向を示すためと考えられる.

表 2: ローカルメモリ参照に関する結果

プログラム名	全メモリ参照回数	ローカルメモリ参照回数(率)	
		単純戦略	最適戦略
6-Queen	18278	5650(30.9%)	11696(64.0%)
8-Queen	390916	122305(31.3%)	249520(63.8%)
Primes-100	6471	2555(39.5%)	4674(72.2%)
Primes-1000	225041	82772(36.8%)	161222(71.6%)

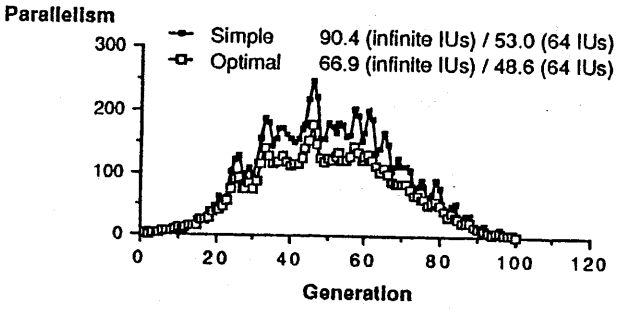

図 5: 並列度に関する結果 (6-Queen)

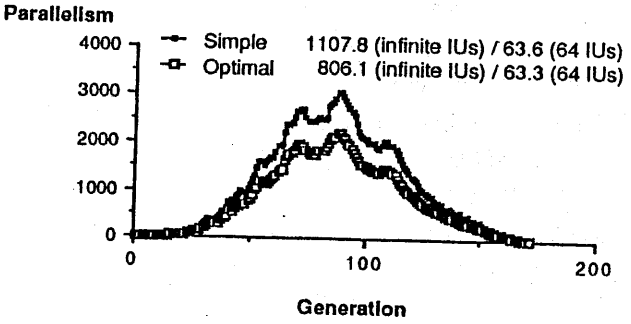

図 6: 並列度に関する結果 (8-Queen)

これらの結果より，本稿で述べた最適化手法を用いて，並列度を落すことなく，ローカルメモリ参照の割合を単純戦略の約2倍まで上げられることがわかる．しかし，依然として比較的多くのリモートメモリ参照も発生することもわかる．このことから，大規模並列処理において並列度の低下を避けるためには，リモートメモリ参照を避けることは困難であり，相互結合網に高い性能が要求されるということがわかる．

7 まとめ

本稿では，プロファイラにより Fleng のプログラムの動的な特性を調べ，負荷分散戦略を最適化する手法について述べた．また，ワークステーション上のインタプリタを用いて，その手法の有効性を確認した．今後の課題として，

- 相互結合網のトラフィックには，ゴールその他の転送などもあるため，それらに関する考察．
- 寿命に関する考察と，ゴールのスケジューリングへのプロファイラの適用．
- プログラムの静的解析との組み合わせによるプロファイラの高速化．
- 実機上での実用的なプログラムを用いた検証．

などが考えられる．

謝辞

本研究は，文部省特別推進研究 No.62065002 の一環として行なわれている．

参考文献

[1] 小池, 田中: “並列推論エンジン PIE64”, 並列コンピュータアーキテクチャ, bit 臨時増刊, Vol.21, No.4, 1989, pp. 488-497.

[2] 高橋, 小池, 田中: “並列推論マシン PIE64 の相互結合網の作成および評価”, 並列処理シンポジウム '90 A1-1, 情報処理学会, May 1990(予定).

[3] 清水, 小池, 田中: “並列推論マシン PIE64 の推論ユニット間通信”, 計算機アーキテクチャ研究会 79-4, 情報処理学会, Nov. 1989.

[4] 島田, 下山, 清水, 小池, 田中: “推論プロセッサ UNIREDII の命令セット”, 計算機アーキテクチャ研究会 79-5, 情報処理学会, Nov. 1989.

[5] 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, 1988. p1031-1040.