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†

日高康雄†† 小池汎平††
舘村純一†† 田中英彦††

従来，コミッティッドチョイス型言語の負荷分散方式には，プログラムにアノテーションを付加して，配置戦略を指示する方式が使われてきたが，指定できる戦略の自由度が低いために，プログラムを書き換える必要があり，その自動付加も困難であった．本稿では，より自由度の高い負荷分散戦略の提案とその自動付加の試みを示す．自動付加する戦略の目標は，「プログラムに内在する並列性をすべて抽出した上で，高いローカルメモリ参照率を示すこと」とし，この上で，必要以上に得られた並列性の抑制は，実行時に行うものとする．まず，「ゴール」と「データ」両方の配置を指定可能とし，「関連のある既存のデータが置かれている PE（要素プロセッサ）と同じ PE」を戦略として用意することを提案する．この柔軟な負荷分散戦略によって，計算木（制御依存関係）に沿った分割だけでなく，データ依存関係に沿った分割が可能となり，ゴールやデータを適切な PE に送り込むような負荷分散戦略を簡潔に表現できるようになる．次に，実行に沿った「履歴」を保持して，負荷分散戦略の最適化に必要な情報を集めるプロファイラについて述べる．このプロファイラの出力により，ヒューリスティクスによらない最適化が可能となる．そして，ワークステーション上に試作した処理系で定量的な評価を行い，実際の並列処理システムでの実行時間短縮への貢献を定性的に検討して，本手法の有効性を示す．

1. はじめに

並列計算機における重要な問題の一つに，負荷分散問題がある．負荷分散とは，データや処理などの負荷を分割する箇所と，各要素プロセッサ（PE）への割り付けを決定することである．負荷分散が適切でないと，負荷の集中や通信のボトルネックのために，並列計算機の目的である台数効果が得られなくなる．

負荷分散に関しては，今までにも多くの研究がなされてきた[1),8)]，例えば，プログラムをタスクグラフとして表現し，グラフの分割，PE への割り当て問題として解く手法[2)]や，規則的な数値計算問題を分割する手法[3)]などがある．しかし，コミッティッドチョイス型言語（CCL—Committed Choice Language）には，これらの手法の適用は困難である．前者はグラフがあらかじめ得られていることを前提とするが，CCL では粒度が細かいためにグラフの規模が非常に大きく，グラフ全体を得るのは非現実的である．また後者は，CCL が対象とする問題の性質が非常に不規則であるため応用できない．

従来，CCL の負荷分散方式では，プログラム上の負荷分散を実行する部分に，その配置戦略を指示するアノテーションをつける方式が主に使われてきた．これまでに提案されている戦略は，PE 番号の直接指定のほか，タートルグラフィックスに倣った PE 番号の相対指定[4)]や，ローカリティを生かすことを狙った仮想的な平面分割指定[5)]などで，戦略を指定できるのはゴールだけであった．しかし，これらの戦略の自由度は低く，計算木（ゴール間の制御依存関係）に沿った分割しかできないため，負荷の集中や過度の分散を招いてうまくいかないことが多かった．適度に負荷分散させるためには，プログラマがプロセッサ数を意識して，プログラムを書き換える必要があった[10)]．また，アノテーションの自動付加も困難であった．

我々は，「静的負荷分割によって，プログラム中の並列性をすべて抽出した上でローカルメモリ参照率*を向上させ，実行時の動的負荷分割で過剰な並列性を抑制し，動的 PE 割り当てで負荷のバランスをとる」というアプローチをとる．本稿では前者を取り上げ，「静的負荷分割による負荷分散戦略」について述べる．

† A Static Load Partitioning Method Based on Execution Profile for Committed Choice Languages by YASUO HIDAKA, HANPEI KOIKE, JUN'ICHI TATEMURA 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

* 分散メモリ型並列計算機上の CCL における PE 間の主要な通信には，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，転送先の PE 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処理の中断を招き，実行時間に響く．

本稿ではまず，「ゴール」と「データ」両方の配置を指定可能とすること，そして，「関連のある既存のデータ等が置かれている PE と同じ PE」を戦略として用意することを提案する．この負荷分散戦略により，「データ依存関係に沿った分割」が可能となり，ゴールやデータを適切な PE に送り込む負荷分散戦略を簡潔に表現できるようになる．

次に，負荷分散戦略の自動付加の試みとして考えた，「プロファイラ」を用いた負荷分散戦略の最適化手法を述べる．本手法の特徴は，プロファイラが単なる統計的データを集めるだけでなく，実行に沿った「履歴」を保持して，十分な情報を収集することにある．その結果，この情報を利用する最適化アルゴリズムは，ヒューリスティクスによらない正確かつ単純なものとなっている．

この負荷分散戦略と最適化手法は，我々が開発中の並列推論エンジン PIE 64[11] と，その記述言語である Fleng[7] を対象として考えられたものである．PIE 64 は本手法を効率良く支援するハードウェアを持っているが，他の並列計算機や CCL に対しても同様の手法が適用できると考えられる．

2. Fleng

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，CCL，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[9] がある．

Fleng のプログラムは，下のような形のホーンクローズを，宣言的に書き並べたものである．

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

:- 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クローズの一つが選択され，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．

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が検査されて，成功すると新たなゴールを作る．リダクションの際に現れる，新たな変数，リストセル，ベクタは，ヒープメモリ上に動的にその領域が確保される．CCL では，このようなインプリメントが一般的である．

3. 並列推論エンジン PIE 64

PIE 64 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) と呼ばれる．ここでは，本手法を効率よく支援するアーキテクチャ上の特徴を述べる．

相互結合網[12]の特徴として，自動負荷分散機能がある．これは，接続時に相互結合網が負荷最小の IU を自動的に選択する機能であり，負荷分散戦略として「負荷最小の IU」が指定されている箇所では，この機能を用いて IU の割り当てを行う．

IU は，UNIRED (Unifier-Reducer)，NIP (Network Interface Processor)，管理プロセッサなどからなっている．UNIRED は，Fleng のプログラムを実行するプロセッサで，その機械語レベルではローカルメモリ参照とリモートメモリ参照を区別する必要はない．NIP は，相互結合網とのインタフェースを持ち，IU 間の通信を支援するプロセッサである．

次章で，他の IU のヒープメモリ上に「データ」を配置する戦略を提案するが，NIP はこの「リモートヒープ割り当て」を効率良く支援する[13]．リモートヒープメモリの割り当ては，割り当て先 IU 上の NIP によって直ちに，管理プロセッサの処理を経ずに行われ，また，UNIRED の処理も妨げない．

4. Fleng における負荷分散戦略

ここでは，本稿で提案する負荷分散戦略について述べる．

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い得る箇所を，負荷分散ポイント (Load Distribution Point) と呼ぶ．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．(変数や構造データを配置する 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．)

負荷分散ポイントのそれぞれに対して負荷分散戦略 (Load Distribution Tactics) を指定する．同じ負荷分散ポイントでは，リダクションの度に同じ戦略をとる．

戦略としては，次の 4 種類を指定する．

戦略 A．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（以後，引数ポインタ）*の指す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，戦略Aによって選択されたIUと同じIUを選択する．

次に，naive reverseの一部を用いて例を示す．

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

負荷分散ポイントは，次の4箇所である．

1. 変数R1を確保する箇所．
2. リスト[X]のセルを確保する箇所．
3. ゴールnreverse(L, R1)を生成する箇所．
4. ゴールappend(R1, [X], R)を生成する箇所．

また，とり得る戦略には，次の八つがある．

1. 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. 戦略Aで割り当てたR1と同じIU（戦略D）．
8. 戦略Aで割り当てた[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で表記する．意味は，以下のとおりである．

- @any, @any(*id*)は，戦略Aを表す．後者は，他の戦略Dから参照される場合である．

```
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) :-
    nreverse(L,R1)@to(iu1),
    append(R1@local, [X]@local, R)@local.
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
Fig. 1 A naive reverse program with load distributing annotations.

* クローズヘッド側が変数である場合，デレファレンスしていない値を用いる．この値は，ポインタでない場合があるため，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うため，デレファレンス後の値を用いる．

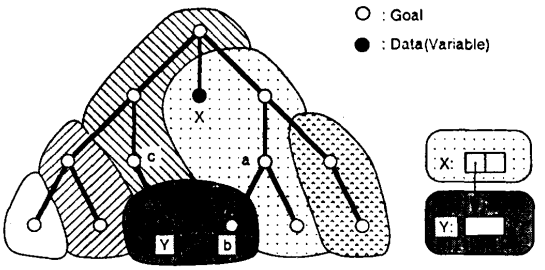


図2 戦略C, Dにより可能となるIUの割り当ての例
Fig. 2 A computation tree partitioned by data oriented tactics.

- @localは，戦略Bを表す．
- @on(*id*)は，戦略Cで参照されるデータを表す．
- @to(*id*)は，戦略Cまたは戦略Dを表し，同じ*id*を持った@on(*id*)や@any(*id*)を参照する．

アノテーションを付加した例を**図1**に示す．

4.2 負荷分散戦略とローカルメモリ参照の関係

戦略C，D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に割り当てることが可能になる．その例を**図2**に示す．図中の木は，計算木とヒープ上のデータを示し，白い円はゴールを，黒い円はデータを示す．また，同じ背景パタン上のゴール，データは，同じIU上に存在することを示す．この例は，以下の過程を示している．

1. aとcは，共有変数Xを持つ．
2. cが，Xを[Y|Z]にバインドする．
   c (X @on (iu1)) :- X = [Y|Z] @to (iu1).
3. aはXのバインドを待ち，bを生成する．
   a ([Y @on (iu2)|Z]) :- b (Y) @to (iu2).
4. bはYをローカルに参照する．

3では，[Y|Z]のcarにかかれたYのアドレスを参照するだけで，Yの内容を参照しない．bはYのアドレスが示すIUに投げられ，そこでYをローカルに参照する．

一方，変数が構造データにバインドされた場合，構造データまでローカルに参照するには，バインドする側の戦略で，変数と構造データを同じIUにする必要がある．2の戦略がこれに相当し，3で[Y|Z]をローカルに参照するのを助けている．

5. Fleng のプロファイラ

本章で述べるプロファイラは，戦略がどのような場合にローカルメモリ参照が何回発生し，また，どのよ

うな場合に並列性の低下が起きるかを出力する．このローカルメモリ参照や並列性低下の条件を正確に調べるために，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る．

5.1 オブジェクトのラベル，履歴

まず，プロファイラの管理データに関する用語の定義を行う．本研究は，ATMS[6]のアルゴリズムに着目したことに端を発しており，この用語の一部はATMSの用語に由来する．

- **オブジェクト** (Object)
  負荷分散ポイントで生成される，ゴール，構造データ，変数を「オブジェクト」と呼ぶ．
- **仮想 IU** (Virtual IU)
  簡単のために，IUの数は無限大で，戦略Aのたびに新たなIUを割り当てると仮定し，割り当てられたIUを「仮想IU」と呼ぶ．
- **戦略仮説** (Tactics Assumption)
  負荷分散ポイントとそこでの戦略の組を「戦略仮説」と呼ぶ．戦略仮説は，そのポイントでその戦略をとるという「仮定」を意味する．
- **戦略環境** (Tactics Environment)
  戦略仮説の集合を「戦略環境」と呼ぶ．これは，戦略仮説のAND条件を表し，それらの戦略仮説を同時に仮定することを意味する．戦略環境は，同じ負荷分散ポイントに関する戦略仮説を二つ以上含まないように管理される．なぜなら，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できず，戦略が同じならば，それらを一つの戦略仮説に縮退できるからである．
- **履歴** (History)
  戦略環境と仮想IUの組を「履歴」と呼ぶ．これはラベルの一要素としてオブジェクトにつけられ，その戦略環境が成り立つ場合に，オブジェクトが存在する仮想IUを示す．
- **ラベル** (Label)
  履歴の集合を「ラベル」と呼ぶ．これはオブジェクトに対してつけられ，どのような戦略環境が成り立つ場合に，そのオブジェクトがどの仮想IUに存在するか 示す．ラベルの持つ各履歴の戦略環境同士は排他的であり，すべての負荷分散ポイントにおける戦略を固定すると，オブジェクトが存在する仮想IUは一意に決定される．

図3は，オブジェクト，負荷分散ポイント，負荷分散戦略，仮想IUの関係を示したものである．この図に従って，オブジェクトが存在するIUが，戦略によってどのように決まっているかを示す．O1〜O4はオブジェクト，P1〜P4はオブジェクトを生成した負荷分散ポイント，T1A〜T4Aは戦略A，T1B, T1C等は戦略A以外の戦略，IU1〜IU4は仮想IUをそれぞれ表す．

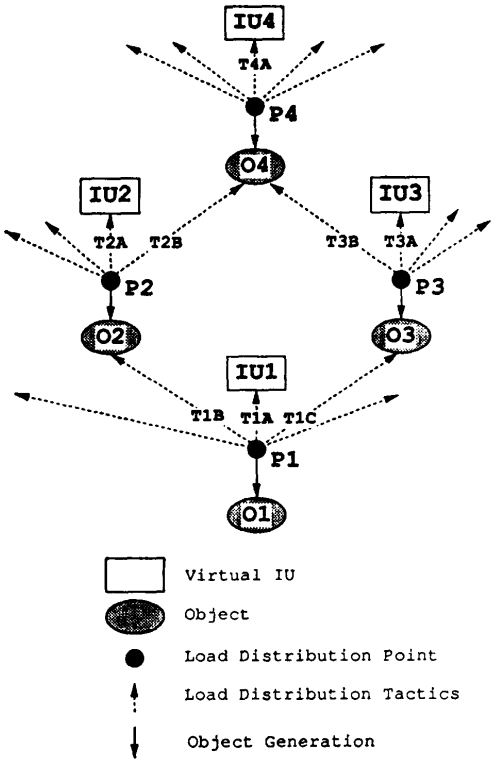


図3 オブジェクト，負荷分散ポイント，負荷分散戦略，仮想IUの関係
Fig. 3 Relation between objects, load distribution points, load distribution tactics and virtual IUs.

O1がどの仮想IU上に存在するかを考える．もし，P1, P2での戦略がそれぞれ，T1B, T2Aであれば，O1はIU2上に存在する．また，P1, P3, P4での戦略がそれぞれ，T1C, T3B, T4Aであれば，O1はIU4上に存在する．すなわち，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が戦略Aで割り当てられて」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．

図3中の各オブジェクトのラベル，履歴の様子を，**図4**に示す．オブジェクトの持つラベルから，どのよ

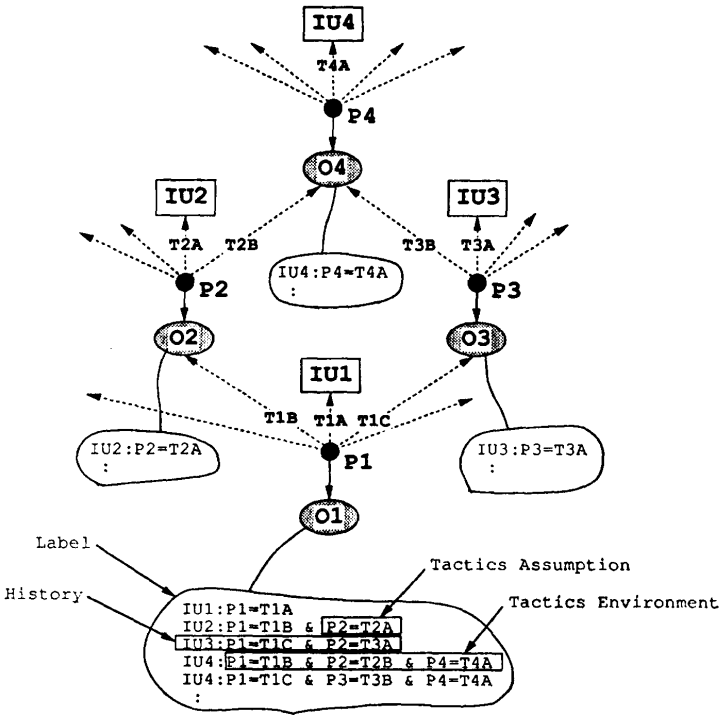


図 4 オブジェクトのラベル，履歴
Fig. 4 Labels and histories of objects.

うな戦略の時に，どの仮想 IU にそのオブジェクトが存在するかがわかり，また，ある仮想 IU に存在するための，戦略に対する条件を調べることもできる．

オブジェクトのラベルは，プロファイラによって管理される．リダクション時に，コミットしたクローズで生成される全オブジェクトのラベルが作られ，オブジェクトの回収時に，ラベルも回収される．これにより，ラベル一つに使われるメモリ量を制限することで，同時に生きているオブジェクト数に比例した量のメモリで，ラベルを管理できる．

リダクション時のラベル生成アルゴリズムの概略を以下に示す．

1. 生成するオブジェクトの数だけ，新たな仮想 IU を割り当てる．
2. 元のゴールのラベルと引数ポインタの指すデータのラベルを取り出す．
3. 各オブジェクトのラベルを，上記の仮想 IU とラベルに，そのオブジェクトを生成する負荷分散ポイントの戦略仮説を加えて，算出する．

ただし，多数の戦略仮説を持つ履歴は省略して，使用メモリ量と計算量を抑える．

5.2 ローカルメモリ参照の条件

メモリ参照がローカルであるためには，実行中のゴールが存在する仮想 IU と，参照したデータが存在する仮想 IU が同じであれば良い．プロファイラはメモリ参照の度に，ゴールのラベルと参照したデータのラベルからこの条件を求めて，その回数を数える．そのアルゴリズムの概略を以下に示す．

1. ゴール，データ，双方のラベルから，同じ仮想 IU の履歴の組み合わせをすべて求める．
2. 双方の履歴の戦略環境をマージする．
3. 戦略Aの戦略仮説を省く*．
4. 戦略環境の間に含意関係があれば，前提側の戦略環境は冗長であるので省く**．これで，戦略環境同士は排他的になる．
5. 以上で得られた一連の戦略環境は，ローカルメモリ参照の条件を表しており，一つの戦略環境が AND 条件を，一連の戦略環境で OR 条件を表している．
6. 各戦略環境に対応するカウンタを1ずつカウントする***．
7. 実行終了後，各戦略環境とその回数を出力する．

5.3 並列性低下の条件

並列性の低下は，同時に実行可能な複数のゴールが同じ仮想 IU に存在する時に起こる．

単純化のため，すべてのゴールリダクションには同一の時間がかかるというモデルを採用し，プロファイルをとる時は，世代単位のスケジューリングを行う．

---

* 仮想 IU が同じであるためには，戦略Aは必要ない．

** 例えば，図4で，O1 と O2 が共に IU2 にあるための条件は P1=T1B, P2=T2A で，共に IU4 にあるための条件は P1=T1B, P2=T2B, P4=T4A で，含意関係はないが，戦略Aを除くと含意関係を生じる．帰結側の条件 (P1=T1B) が満足されれば，このメモリ参照がローカルであることは保証されるため，前提側の条件 (P1=T1B, P2=T2B) を残す必要はない．前提側が冗長であるのは，ローカルメモリ参照の条件が満たすための条件であるからで，並列性低下の条件では，前提側の条件を省くことはできない．

*** OR 条件を記録する必要がないのは，戦略環境同士が排他的になっているためで，一つの戦略環境を確定させると他の戦略環境は必ず偽となり，どの戦略環境が何回分のローカルメモリ参照に貢献するかが正確にわかるからである．並列性低下の条件では排他的にならないが，すべての戦略環境を偽とするように使われるため，OR 条件は必要ない．

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．

プロファイラは世代交代の際に，アクティブゴールのすべ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（ただし，4の前提側の省略はしない）．

一方，計算木上の近傍のゴールが同じ仮想 IU に存在することをも避けようとすると，ローカルメモリ参照率を上げられない場合がある．これは，処理の軽いゴール同士でも並列性低下の対象となって別の IU に分離されるため，共有変数をローカルにアクセスできるゴールが一つだけに制限されるからである．

そこで，「並列性低下の猶予期間」を設ける．並列性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，並列性低下の条件から省くことにする．これによって，1，2回のリダクションで処理を終了するようなゴールを，並列性低下の対象から外すことができる．猶予期間に処理を終了しない重いゴールが特定の IU に集中することこそ，大幅な並列性の低下を招くと考えられるため，猶予期間を導入しても並列性の低下は十分に避けることができる．

5.4 プロファイラの試作

本手法の有効性を調べるために，開発中の PIE 64 に先だって，逐次型ワークステーション上にプロファイラを試作した．プロファイラが情報を得る Fleng 処理系には，既存のインタプリタを用いたが，プロファイラ自身には，本手法のアルゴリズムをインプリメントした．

インタプリタとプロファイラは別のプログラムとして動作する．ソケットを通じて送られる実行トレース情報は十分に抽象化されており，実際の並列処理系において送られる情報と同じものである．また，ゴールのスケジューリングは世代単位となっており，スケジューリングの特性は実際の並列処理系と同じである．

一方，プロファイラ自身は負荷分散戦略を固定しないため，実際の並列処理系であっても，サンプル実行時にとった負荷分散戦略に依存しない結果を出力する．つまり，実行が1台で行われても，プロファイラの出力結果は有効である．

この試作処理系における限界は，インタプリタをベースとしている点であり，コンパイラを用いた場合と比べて，ヒープメモリ参照の特性が若干異なることである．このため，試作処理系で得られた最適戦略は，類似のメモリ参照特性を持つ並列処理系に対しては有効であるが，メモリ参照特性が異なる場合は，プロファイルをとり直す必要がある．

以上より，試作処理系は，実際の並列処理系でも有効な最適戦略を求められるとは限らないものの，本手法の有効性の確認には十分であるといえる．

5.5 負荷分散戦略の最適化

負荷分散戦略の最適化は，並列性低下の条件を避けつつ，多数回のローカルメモリ参照の条件が満たされるように，戦略仮説を一つずつ確定させていくことで行われる．最適化のアルゴリズムの概略を以下に示す．初めは，すべての戦略仮説が未確定な状態にある．1と2で多数回のローカルメモリ参照に関係のある負荷分散ポイントを一つ選択し，3でその負荷分散ポイントでとり得る戦略から並列性低下を起こす戦略を取り除き，4と5でローカルメモリ参照をなるべく増やすように戦略を決定し，その戦略仮説を一つ確定させる．そして，6で，この戦略仮説に反するために，もはや確定できなくなった戦略仮説を含む戦略環境を削除する．ローカルメモリ参照の条件に未確定な戦略仮説が残る限り，1から7を繰り返し，それで決定されなかった負荷分散ポイントに対する戦略を8から 11 で決定する．なお，CL はローカルメモリ参照の条件，CP は並列性低下の条件を示す プロファイラの出力データで，戦略環境と回数の組の集合で表される．

1. CL 中の未確定な戦略仮説を含む戦略環境の中から，回数が最多のものを一つ選び，Eとする．
2. Eに含まれる未確定な戦略仮説の負荷分散ポイントの中から，それに関する戦略仮説を含んだ戦略環境が CL 中で回数の上位に多くある負荷分散ポイントを一つ選び，これを P1 とする．
3. P1 でとり得る戦略のそれぞれに対して，CP 中の戦略環境の中に，P1 とその戦略の組み合わせによる戦略仮説だけが唯一未確定で，他の戦略仮説は既に確定しているような戦略環境が存在するか否かを調べる．存在する場合はその戦略は並列性低下を起こすため，候補から外す．存在しない場合に，P1 とその戦略の組み合わせを，確定させる戦略仮説の候補とする．

4. CL 中の戦略環境で，3で求めた戦略仮説の候補を含み，回数が最多*の戦略環境の中から，未確定の戦略仮説の数が最少**の戦略環境を選ぶ．同数の戦略環境がいくつかある時は，その一つを任意に選ぶ．この戦略環境の中の，3で候補とした戦略仮説を確定させて，P1 においてその戦略をとることを決定する．
5. 3で残った候補が少ないために，4で戦略環境を一つも選べない場合は，戦略Bの戦略仮説が候補にあれば戦略 B，なければ戦略Aを P1 に対して決定し，それらの組み合わせによる戦略仮説を確定させる．
6. 確定した戦略仮説と，負荷分散ポイントが同じで戦略が異なるような戦略仮説は，もはや確定し得ないため，そのような戦略仮説を含む戦略環境をすべて，CL，CP から取り除く．
7. CL 中に未確定な戦略仮説を含んだ戦略環境がなくなるまで，1から6を繰り返す．
8. 戦略が決定していない負荷分散ポイントを任意に選び，これを P2 とする．
9. CP 中の戦略環境の中に，P2 と戦略Bの組み合わせによる戦略仮説だけが唯一未確定で，他の戦略仮説は既に確定している戦略環境が存在するか否かを調べる．存在する場合は，戦略Aを，存在しない場合は，戦略Bを P2 に対して決定し，それらの組み合わせによる戦略仮説を確定させる．
10. 6と同様に，確定した戦略仮説と，負荷分散ポイントが同じで戦略が異なるような戦略仮説を含む戦略環境をすべて，CP から取り除く．
11. すべての負荷分散ポイントに関する戦略仮説が確定するまで，8から 10 を繰り返す．

本手法の結果最適化された *n*-Queen のプログラムを**図5**に示す．

```
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5 負荷分散戦略を最適化した *n*-Queen のプログラム
Fig. 5 An *n*-Queen program with the optimized load distributing strategy.

* ただし，1，2回の違いは同数と見なす．
** ラベル生成時に履歴を省略できるのは，このためである．

6. 評　価

負荷分散戦略の最適化の効果を調べるために，試作システムのプロファイラに評価モードを設けた．評価モードで動作するのはプロファイラだけで，インタプリタの動作はプロファイル時と変わらない．このため，実行のトレース情報はプロファイル時と同じものである．評価モードのプロファイラでは，各オブジェクトのラベルは，固定された戦略に従う履歴のみを常に唯一持ち，そのオブジェクトが存在する仮想 IU を正確に追跡する．

メモリ参照時には，実行中のゴールと参照したデータが存在する仮想 IU を比較し，一致すればローカル参照とする．世代交代時には，アクティブゴールが存在する仮想 IU の数を数えて，その世代の並列度とする．平均並列度は，IU 台数を無制限とした単純な平均値と，64 台に制限して，並列度 (=*n*) が 64 を超える世代を，並列度 64 が *n*/64 世代続くとした平均値を求める．自動負荷分散による IU 割り当てや動的負荷分割の効果は考えない．

最適化した戦略の比較の対象として，以下の単純戦略を用いる．

- 構造データ，変数はすべてローカル 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．

この単純戦略は，最大の並列性を抽出する．

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．ラベル生成時には戦略仮説を四つ以上含む場合を省略し，並列性

表 1 ローカルメモリ参照に対する最適化の効果
Table 1 The optimization effect on the rate of local memory access.

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	18,278	5,650 (30.9%)	11,696 (64.0%)
8-Queen	390,916	122,305 (31.3%)	249,520 (63.8%)
Primes-100	6,471	2,555 (39.5%)	4,674 (72.2%)
Primes-1000	225,041	82,772 (36.8%)	161,222 (71.6%)

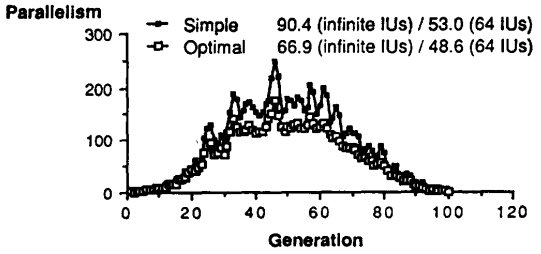


図 6 並列度に対する影響 (6-Queen)
Fig. 6 The optimization effect on the parallelism of 6-Queen.

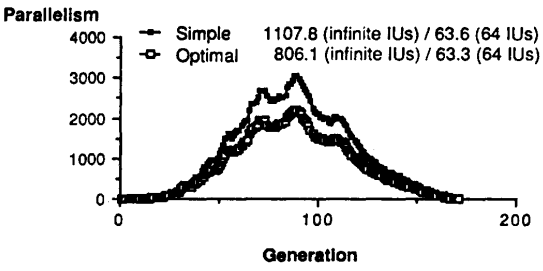


図 7 並列度に対する影響 (8-Queen)
Fig. 7 The optimization effect on the parallelism of 8-Queen.

低下の猶予期間を2世代までとした．SUN 4-260 上のプロファイラの実行時間は，Primes-100 が 235.4 秒，6-Queen が 3875.9 秒であった*．最適化した戦略の評価は，Primes-1000，8-Queen に対しても行った．ローカルメモリ参照に対する最適化の効果を**表 1** に示す．また，6-Queen，8-Queen の並列度に対する影響を**図 6**，**図 7** に示す．

若干並列度が低下している原因には，並列性低下の猶予期間を設けていること，ラベル生成アルゴリズムで履歴の省略を行っていることなどが考えられる．図5の8番目のクローズで作られるゴールは，猶予期間内の近傍のゴールにあたり，同じ IU に配置されて並列性を3だけ低下させる．8-Queen の場合，これとクローズのコミット回数 (17,204) の積は 51,612 であり，図7の並列性低下量 (51,897) の大部分を占め，ほとんどが近傍のゴールによるものであることがわかる．

以上，プロファイラの出力結果を用いて最適化を行うと，自然な並列性を損なわずに，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．これより，本手法が，静的負荷分割の最適化に役立つことが結論づけられる．

7. 考　　察

並列処理研究の最終的な目標は，実際の並列処理システムにおける実行時間の短縮であり，一つの要素技術の改善だけでなく，他の要素技術との関係を検討する必要がある．そこで，本手法と他の要素技術との関係，そして，本稿で対象とした静的負荷分割と実行時間の関係について，定性的な検討を行う．

まず，本手法と他の要素技術との親和性は，かなり良いと言える．本手法は，静的分割による負荷分散戦略を，並列性を損なわずに最適化するものであり，プロセッサへの割り当てを実行時に行い負荷のバランスをとる動的 PE 割り当てや，実行時の並列度に合わせて過剰な並列性を抑制する動的負荷分割などの技術を制限するものではない．これらの技術を用いた上で，さらに本手法を適用し，実行時間を短縮することができる．

本稿では CCL を対象としたが，一つ目の提案である「データを参照する戦略，データ配置の戦略」は，並列処理言語一般に適用できると言える．二つ目の提案であるプロファイラは，CCL の特徴の中で，「分割し得る箇所が自明で，オーバヘッドを無視すればすべての並列性を容易に抽出できる」という細粒度高並列指向言語の特徴と，「クローズ単位で，あらかじめある程度の処理がまとめられている」というリダクションモデルの特徴を利用している．CCL 以外では，細粒度の関数型言語をリダクションモデルで処理する場合などにも，本手法を適用できる可能性がある．

次に，静的負荷分割の改善が，実行時間の短縮に貢献する程度を考える．静的負荷分割の技術は動的負荷分割の技術と相補関係にあるため，動的分割の技術が有効に働く対象領域と働かない対象領域とでは，実行

* インタプリタは LAN で結ばれた別のワークステーションで実行している．

時間短縮への貢献の程度が変化する．

動的分割の技術は，時間変動する並列性が，プロセッサ台数を大幅に上回る場合は有効に働くが，並列性とプロセッサ台数が近付くと分割が頻繁に行われ，有効に働かなくなるという問題点を持っている．これに対して静的分割の技術は，並列性とプロセッサ台数がたとえ等しい場合にも有効な戦略を与える技術である．すなわち，動的分割の技術が有効に働かなくなる高並列処理の領域においては，静的分割の技術が実行時間に与える影響が，非常に大きくなってくると言える．

8. まとめ

本稿では，まず，「データを参照する戦略」，「データ配置の戦略」という柔軟な負荷分散戦略の提案を行った．次に，「履歴」を保持して十分な情報を収集する「プロファイラ」を導入して，ヒューリスティクスによらずに戦略を最適化する手法を示した．そして，負荷分散戦略の改善を定量的に評価し，実際の並列処理システムにおける実行時間短縮への貢献について定性的に考察して，本手法の有効性を確認した．試作処理系は Fleng を対象としたが，Fleng 以外の CCL に対しても，同様の手法を適用できると考えられる．今後の課題として，本手法により得られた知見を元にした，プログラムの静的解析と処理の軽いプロファイラとを合わせた実用的な手法の開発などが考えられる．

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

参考文献

1) 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).
2) Kasahara, H. and Narita, S.: Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing, *IEEE Trans. Comput.*, Vol. C-33, No. 11, pp. 1023-1029 (1984).
3) Berger, M. J. and Bokhari, S. H.: A Partitioning Strategy for Nonuniform Problems on Multiprocessors, *IEEE Trans. Comput.*, Vol. C-36, No. 5, pp. 570-580 (1987).
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) de Kleer, J.: An Assumption-Based TMS, *Artif. Intell.*, Vol. 28, pp. 127-162 (1986).
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) 淵(監修)，古川，溝口(共編)：並列論理型言語 GHC とその応用，p. 277，共立出版，東京 (1987).
10) 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).
11) 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).
12) 高橋，小池，田中：並列推論マシン PIE 64 の相互結合網の作成および評価，並列処理シンポジウム JSPP '90 論文集，pp. 89-96 (1990).
13) 清水，小池，田中：並列推論マシン PIE 64 の推論ユニット間通信，情報処理学会計算機アーキテクチャ研究会資料，79-4 (1989).

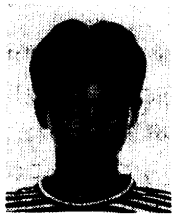
(平成 2 年 8 月 20 日受付)
(平成 3 年 3 月 4 日採録)

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．計算機アーキテクチャ，並列処理，機械系CAD，形状モデリングに興味を持つ．電子情報通信学会，精密工学会，IEEE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，現在にいたる．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．



舘村　純一（正会員）

昭和42年生．平成元年東京大学工学部電子工学科卒業．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．並列処理，並列プログラミング言語とその環境，並列オブジェクト指向，ユーザインタフェース等に興味を持つ．

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．