事例ベース推論による並列プログラミング支援システム

山崎 勝弘 古川 知之

１９９４年１１月１１日

Abstract

過去の並列プログラムを極力再利用して並列プログラミングの負担を軽減させる方法について述べる。事例ベース推論は知識獲得の隘路を軽減させる手法として、裁判や故障診断などの実規模の問題に適用されてきた。本研究では並列プログラムの構造を解析して、その骨格を示すスケレトンを作成し、それに細部の肉付けを行って、並列プログラムを作成する手法について検討する。スケレトンにはタスク分割、同期、並列化手法など並列プログラムの最も重要な部分が含まれる。事例はスケレトン、プログラム本体、及びインデックスから成る。新たな問題に対して、類似したスケレトンを事例ベースから検索し、それに自動修正／派生的類推による修正／ユーザによる修正を行って並列プログラムを生成する。

Abstract

A parallel programming support system, which utilises past typical programs as much as possible to reduce the burden of parallel programming, is proposed. Case-Based Reasoning is expected to relieve the bottlenecks of knowledge acquisition, and has been applied to several practical problems. Cases consist of program itself, indices and skeletons. Skeletons show the basic structure of parallel programs, while indices illustrate their features. When a new problem is given, the system retrieves the most relevant case from the case base using indices and adapts it to the given problem. Automatic case adaptation, derivational analogy and presentation of relevant cases are proposed as case adaptation methods.

1　はじめに

近年、仮想共有メモリ方式の並列マシンが商用化され、分散メモリ型並列プログラミングの困難さを解決する手段として注目されている。単一の仮想アドレス空間、自動アドレス変換、及びデータ管理の透過性はスケーラビリティを実現する上で極めて魅力的である。しかし、仮想共有メモリプログラミングの特長を十分に引き出すためには、並列プログラミングモデル、問題の分割、論理アドレスから物理アドレスへのマッピング、コンパイラ最適化、デバッグ、テストなどを支援するトータルな並列プログラミング支援環境が必要不可欠である。

一方、専門家システムを開発する上で最も困難な問題点の一つは、知識獲得である。いかなる問題解決においても、専門家の知識が再利用可能な形で保持されていなければ、後続の開発者は全く同様の苦労をしなければならない。事例ベース推論（CBR:Case-Based Reasoning）は知識獲得の隘路を減少させる手段として、近年、裁判や故障診断などの実規模の問題に適用され、成果を上げてきた。過去の類似した判例に基づいて現在の裁判の判決を下すHYPOシステム[1]は、CBRによる最も成功したシステムの一つである。

本研究では、過去の類似した並列プログラムを極力再利用して、並列プログラミングの負担を軽減させる方法について検討する。並列プログラミングでは、問題のタスク分割、タスク間の情報授受が並列処理の性能を決定する重要な要因である。すなわち、複数プロセッサの負荷均衡を図り、かつプロセッサ間の情報授受のオーバーヘッドを極力減少させなければならない。現状では逐次プログラミングをマスターしたユーザであっても、並列プログラミングの訓練に多大の労力を要している。また、熟練した並列プログラマであっても、対象マシンが替わるとアーキテクチャの詳細を学ばなければ、高性能の並列プログラムを開発できない。

本研究で提案する手法は対象マシンや言語に独立であるが、ここでは仮想共有メモリ並列マシンKSR 1上でのFortran/Cプログラミングを仮定する。並列プログラミングでは、タスク分割や共有変数の相互排除と共に、そのマシン独自の並列化手法を熟知している必要があり、また困難である。本手法では並列プログラムの骨格をスケレトンとして用意する。また、類似したスケレトンを検索するために、その問題の特徴をインデックス付する。インデックス、スケレトン、プログラム本体を一つの事例として、事例ベースに格納する。新たな問題に対して、再利用可能な類似事例を事例ベースから検索し、スケレトンへの肉付けを自動/手動で行ってプログラムを生成することを目指す。類似事例の自動検索を可能とするために、並列アルゴリズムを分類する。以下、並列プログラミング、並列アルゴリズムの分類、事例の表現、及び事例の検索と修正について述べる。

2　並列プログラミング

2.1　並列プログラミングの分類

並列プログラミングは共有メモリ並列プログラミング、分散メモリ並列プログラミング、データ並列プログラミング、オブジェクト指向並列プログラミング、関数型データフロープログラミングなどに分類される[2]。

分散メモリ並列プログラミングでは、ホーアの提唱したCSPモデルが代表的であり、プロセス間のセンド、レシーブを陽に指定する。各プロセッサにおいて、演算時間を最大に、通信時間を最小にする必要がある。このために、計算と通信をオーバーラップさせ、通信オーバーヘッドを極力減少させる工夫が要求させる。共有メモリ並列プログラミングでは、共有変数の相互排除を実現するために、ロック、アンロックを陽に指定する。共有変数による各プロセッサ間の情報授受は、分散メモリ並列プログラミングにおけるセンド、レシーブに対応する。

データ並列プログラミングでは、大規模なデータセットに対して同一演算を行い、細粒度の並列性を実現する。大規模データに対する演算を逐次プログラムと同じ制御フローで実現できるので、前二者よりもプログラミングは容易であ

ると考えられる。

分散メモリ並列プログラミングと共有メモリ並列プログラミングはSIMD、SPMD、MIMDいずれにも適しているが、データ並列プログラミングはSIMD、SPMDに適している。どれが最も適しているかは、通信コスト、負荷均衡、プロセッサスケジューリングなどに依存する。

並列プログラミングの要点は以下のように要約される。

- 各プロセッサにおける負荷均衡を実現するために、いかにタスクを分割するか。
- 各プロセッサ間の情報授受のためのオーバーヘッドをいかに減少させるか。

代表的な分散メモリ型マイクロプロセッサであるトランスピュータ用のParallel Cにはスレッドが用意されている。スレッドとは軽いプロセスであり、コード、ヒープ、静的・外部データメモリを他のスレッドと共有する。各スレッドはデータを共有でき、チャネルまたは共有メモリによって通信できる。スレッドを用いることにより、計算と通信をオーバーラップできる。

共有メモリ型並列マシンの一種として、仮想共有メモリ方式が近年注目されている。この方式ではメモリは実際には分散しているが、単一の仮想アドレス空間を実現し、そこでのプログラミングを可能とするものである。マルチスレッドによる並列プログラミングでは、単一のプログラム（SPMD）を用意して、各スレッドを各プロセッサで実行する。スレッド間の同期、及び共有変数の相互排除をいかに実現するかが要点である。

2.2　並列プログラミング支援システム

図１にシステム構成を示す。システムは複数のインデックスを用いて最も類似した事例を検索する。完全にマッチする類似事例が検索できないときは、検索条件を緩めて、とにかく類似した事例を探す。問題の解析では事例検索のための十分な特徴付を行わねばならない。ユーザは応用の種類、仕様、並列化アルゴリズムの種類、及びプログラムの特徴を指定する。プログラムの特徴には変数、終了条件、同期方法、並列化手法などが含まれる。変数はさらに、共有、局所、リダクション、コモンに分類される。

最善の場合、非常に類似した事例が検索され、与えられた問題への変換が自動的になされる。一方、最悪の場合でも、何らかの関連性のある事例が検索されるので、ユーザによる並列プログラミングの手助けになる。中間の場合には、スケレトンへの情報の追加が自動あるいは手動でなされる。スケレトン内には、スレッドの使用法、共有変数の同期、タスク分割などと共に、変数初期化、計算、結果の回収などの一連のプログラミングプロセスが記述されている。

最悪の場合でもユーザに類似事例を提示するために、事例ベースは過去の代表的な並列プログラムのみならず、並列化手法の代表的な使用方法をできるだけ多数含まねばならない。システムの要点は、過去の類似したプログラムとプログラミング過程を用いて、どれだけプログラムを自動的に生成できるかである。従って、本システムは並列CASE(Computer-Aided Software Engineering)と考えられる。

3　並列アルゴリズムの分類

基本的なデータ操作、数値計算、記号処理、ビジネスデータ処理など広範な応用分野において、並列アルゴリズムを開発するための努力がなされてきた。最近では、１９９０年代のグランドチャレンジとして、気象予測、プラズマモデリング、財政モデリング、乱流計算など非常に大規模な問題が対象になっている[3]。

3.1　Rabhiによる分類

Rabhiは並列アルゴリズムを四つのパラダイムに分類している[4]。

- 分割統治法 (Divide and Conquer)
  問題は下位の部分問題に分割され、それら自身がさらに分割されて再帰的に解かれる。最終結果は部分問題の解を再帰的に結合することにより得られる。

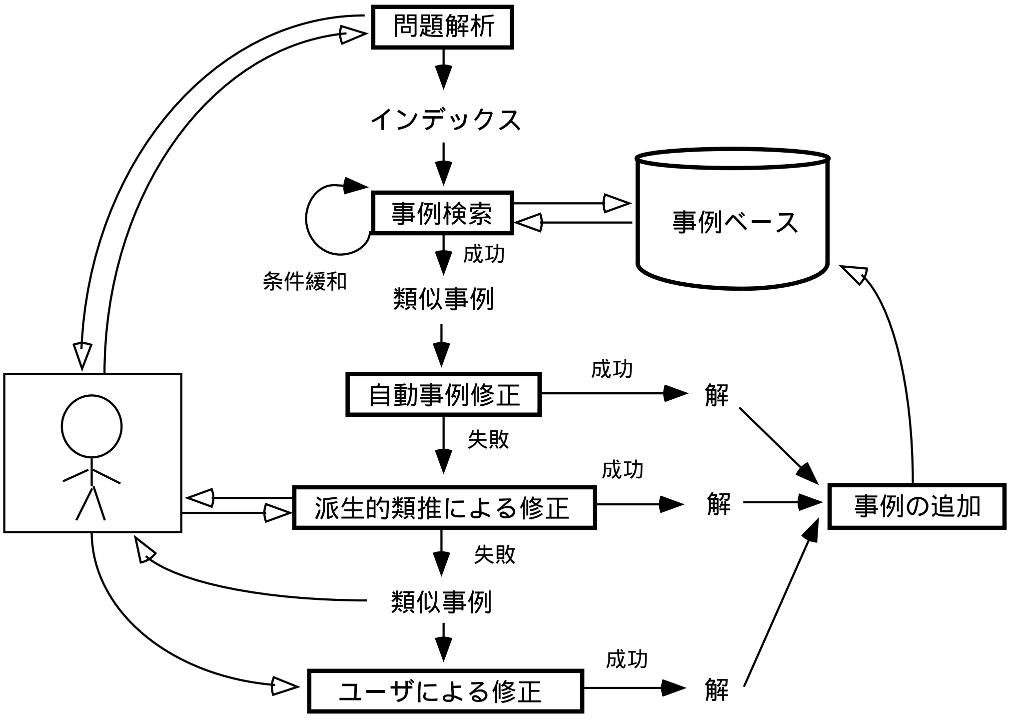

図 1: システム構成

- プロセスネットワーク (Process Networks) 計算を複数のステージに分け、データがステージを流れる。入力データの有無によって、ステージは並行に動作する。

- プロセッサファーム (Processor Farms) 問題は複数の独立な計算に分割され、それらの結果が統合される。全体の制御は一ヶ所で行われ、独立した計算の数はプログラマあるいは使用可能なスレーブ数によって決まる。

- 繰り返し変換 (Iterative Transformation) オブジェクトは複数の繰り返しステップにより変換され、局所・広域・結合オペレータが適用される。局所・広域オペレータはデータの種類により分類される。結合オブジェクトでは、オブジェクトのグループが他のオブジェクトの集合を形成する。

3.2 BACS による分類

BACS(Basel Algorithm Classification Scheme) [5] はアルゴリズムの分類、複雑さの解析などの方法論的側面を明かにするために提案された。BACS は主にアルゴリズムのトポロジー、プロセス構造、相互作用の方式、データ分散、及び実行構造の観点から並列アルゴリズムを分類する。アルゴリズムトポロジーは規則性と均質性から分類される。規則的・均質なトポロジーとして、木、トーラス、メッシュ、ハイパーキューブが、規則的・不均質なトポロジーとして、ワーカーがある。不規則なトポロジーとは各プロセスが異なった構成ルールをもつものである。プロセス構造には静的と動的がある。相互作用は 2 ノード間か (直接) / 3 ノード以上間か (広域)、及び陽になされるか / 陰になされるかの両点から分類される。例えば、ミューテックス (mutex) による相互排除は広域かつ陰である。実行構造として、微細 (microscopic) と巨視 (macroscopic) がある。微細構造は計算、相互作用、及びデー

モンコールの基本要素によるプロセスの合成順序を示し、並列アルゴリズムのスケルトン生成の基礎となる。一方、巨視構造はアルゴリズムの全てのイベントの列である。殆どの並列アルゴリズムの実行構造は規則的であり、それらはイベントの同一の繰り返し系列から成っている。

3.3 KSR 1上での並列化手法

スレッド (Posix thread) とはプロセス内での制御の逐次フローで、他のスレッドと協力して問題を解く。KSR 1にはパラレルリージョン (parallel region)、パラレルセクション (parallel section)、タイリング (tiling) の三つの並列化手法がある [6]。

- パラレルリージョン
  コードセグメントの複数の初期化を並列に実行する。すなわち、複数のスレッドが同じコードセグメントを並列に実行する。
- パラレルセクション
  一つのスレッドが一つのセクションに対応し、全セクションを同時に実行する。
- タイリング
  ループの実行が複数のタイルあるいは繰り返しのグループに分割され、ループ全体が分割されて並列に実行される。

スケジューリング、及び私有 / 共有変数はプログラマにより管理されねばならない。スレッドはミューテックス、バリア (barrier)、コンディション変数 (condition variable) により相互作用する。ミューテックスは共有変数の相互排除を可能とする。バリアは複数のスレッドが同期をとるためのものである。コンディション変数は他のスレッドから信号を受け取るまで、ミューテックスをブロックしたり解放する。

4 事例の表現

事例はインデックス、スケルトン、プログラム本体、及び履歴から成る。優先度を持つ複数のインデックスが類似事例の検索に利用される。応用、仕様、アルゴリズム、変数、及び終了条件はマシン独立なインデックスであり、一方、同期と並列化手法は Posix 標準に準拠している。スケルトンはプログラムの骨格を示し、もし必要な情報が補われれば、目標のプログラムが生成される。履歴はその事例がどのように修正されたか、それが成功したか失敗したかを示す。

4.1 一般的表現

1. インデックス
   - 応用
     データ操作、数値計算、記号処理、ビジネスデータ処理
   - 仕様
     問題の定義
   - アルゴリズム
     分割統治、プロセスネットワーク、プロセッサファーム、繰り返し変換
   - BACS による分類

   - 変数
     共有 / 局所、リダクション、コモン
   - 終了条件
     一定、条件式
   - 同期
     ミューテックス、バリア、コンディション変数
   - 並列化手法
     パラレルリージョン、パラレルセクション、スレッドライブラリ
2. スケルトン
3. プログラム
4. 履歴

4.2 事例の例

4.2.1 マンデルブロー

(1) インデックス

- 数値計算
- $Z_{i+1} = Z_i^2 + C \quad i = 0, 1, 2, ...$
- プロセッサファーム
- $Fix_w(I^t_{lock \cdot unlock}, C^t).$
- 共有、局所、リダクション
- 一定
- ミューテックス
- パラレルリージョン

(2) スケルトン

```
変数初期化
pthread_mutex_init(mu1)
parallel region (スレッド数、局所変数)
do
    スレッド番号によるタスク割り当て
    do
        計算
        pthread_mutex_lock(mu1)
            共有変数への書き込み
        pthread_mutex_unlock(mu1)
    enddo
enddo
end parallel region
```

4.2.2 レイトレーシング

(1) インデックス

- 数値計算
- $At^2 + Bt + C = 0$ (球)

  $A = {E_x}^2 + {E_y}^2 + {E_z}^2 = 1$

  $B = 2E_x(v_x - x_0) + 2E_y(v_y - y_0) + 2E_z(v_z - z_0)$

  $C = (v_x - x_0)^2 + (v_y - y_0)^2 + (v_z - z_0)^2 - r^2$
- プロセッサファーム
- $I^t_{barrier}, Fix_w(C^t).$
- 共有、局所
- 一定
- バリア
- スレッドライブラリ

(2) スケルトン

```
pthread_barrier_t A;
画像データ用変数の宣言;
main()
{
    変数の初期化;
    pthread_barrier_init(&A, プロセッサ数);
    for(スレッド数)
      pthread_create(ray, スレッド番号);
    ray(0);
    画像データのファイルへの出力;
}
void *ray(スレッド番号)
{
      pthread_barrier_checkin(&A);
      スレッド番号によるタスク割り当て;
      for(スレッド番号による計算範囲){
            レイトレーシングの計算部;
            画像データ用変数へのデータの格納;
      }
      pthread_barrier_checkout(&A);
}
```

5 事例の検索と修正

5.1 事例の検索

インデックスは抽象的なものから具体的なものへと並べられ、優先付されている。抽象化のレベルの観点 [7] から見ると、変数と終了条件はプログラムレベルに対応し、同期と並列化手法は実現レベルに対応する。事例検索では全ての

インデックスが合致するもの、さもなければ検索条件を緩めて、できるだけ多数のインデックスが合致するものを検索する。

5.2　自動的な事例の修正

新しい問題と検索された事例の仕様が式のレベルで非常に似ていれば、自動的な修正の可能性がある。例えば、マンデルブローの変形で、右辺が Z_i の関数である式が与えられれば、システムは右辺の計算部分だけを修正すれば良い。

5.3　派生的類推による修正

プロセッサファームはデータ並列計算を含んでいる。データ並列計算の代表的な例は粒子シミュレーション、流体フローモデリング、情報検索、コンピュータグラッフィクスなどである[2]。これらの問題は同一サイズのタスクに分割され、結果を格納するための共有・リダクション変数が必要である。従って、マンデルブローのスケレトンがこれらの問題に使用できる可能性がある。スケレトン内の変数初期化、タスク割り当て、計算、共有変数への書き込みの部分は各プログラム毎に異なる部分である。従って、スケレトンをユーザに提示し、これらの部分への書き込みをシステムとユーザが会話的に行うのが、現実的な修正法であると考えられる。

6　関連した研究と課題

Cole はアルゴリズムの特定のスタイルの構造を記述するアルゴリズムスケレトンを提案した[8]。分割統治スケレトン、繰り返し結合スケレトン、クラスタスケレトン、及びタスクキュースケレトンを提案し、高いレベルで良いアルゴリズムスタイルを記述することができる。

BACS によるアルゴリズムの分類に基づいて、スケレトン指向プログラミング（SOP:Skeleton-Oriented Programming）が提案されている[5]。アルゴリズムスケレトンはそこに計算部分を補うことによって、最終プログラムとなる実行、同期のパターンである。プロセス間の同期はアルゴリズムの主要構造に埋め込まれている。まず、プログラマは主要なアルゴリズムスケレトンを保持したデータベースを概観して、適当なスケレトンを選ぶ。アルゴリズムの記述はスクリプト言語によっている。次に、その記述はスケレトンジェネレータにより、プログラムスケレトンに変換される。プログラムスケレトンは仮想マシン上のある言語に対するアルゴリズムを示す。最後に、プログラマによりプログラムスケレトンに必要な情報が補われて、プログラムが完成する。

SOP は本稿で提案した手法と極めて近いが、プログラムの移植性とスケレトン検索の点で異なっている。SOP ではプログラムの移植性を実現するために、仮想マシン上のある言語に対するアルゴリズムを示すプログラムスケレトンが生成されるが、ユーザがデータベースからスケレトンを手動で探さねばならない。本手法では事例ベースからのスケレトンの自動検索と自動修正に重点を置いており、プログラム移植性はその次の段階の目標と考えている。

7　おわりに

本稿では過去の代表的な並列プログラムの構造をスケレトンとして事例ベースに保存し、新しい問題に対する類似したスケレトンを検索・修正して並列プログラムを生成する手法について検討した。並列プログラムにおけるタスク分割、共有変数の同期、並列化手法はプログラミングが最も困難であるが、同じ種類のアルゴリズムに対しては、同一の構造を適用できる。本研究ではこの性質を利用して、新しい問題に類似したスケレトンを検索し、そこへの肉付けを自動/手動で行って並列プログラムを生成し、並列プログラミングの困難さを緩和させることを目指している。現在、種々の並列プログラムを記述して、そのスケレトンを作成する作業を進めている。

今後の課題として、以下の点があげられる。

- 代表的な並列プログラムを保持する事例ベースの構築

- 類似したスケレトンに自動／手動で肉付けを行って、並列プログラムを作成するための事例の修正法

謝辞

本研究は山崎がマンチェスター大学コンピュータサイエンス学科に留学中に構想を練ったものである。すばらしい研究環境と本研究へのご協力を頂いたJohn R. Gurd教授に深謝する。また、KSR1を使用させて頂いているキャノン・スーパーコンピューティングS.I.株式会社に感謝する。

参考文献

[1] Ashley, K.D. and Rissland, E.L., “A Case–Based Approach to Modeling Legal Expertise”, *IEEE EXPERT*, Fall, Vol.3, No.3, pp.70–77, 1988.

[2] Levis, T.G. and El-rewini, H., “Introduction to Parallel Computing”, *Prentice-Hall International*, 1992.

[3] Siegel, H.J. et al., “Report of the Purdue Workshop on Grand Challenges in Computer Architecture for the Support of High Performance Computing”, *Journal on Parallel and Distributed Computing*, 16,pp.199–211, 1992.

[4] Rabhi, F.A., “Exploiting Parallelism in Functional Languages:a Paradigm-Oriented Approach” , *Second Workshop on Abstract Machine Models for Highly Parallel Computers* , pp.1–14, April 1993.

[5] Burkhart H. et al., “BACS:Basel Algorithm Classification Scheme”, *Technical Report 93-3*, Universitat Basel, 1993.

[6] Kendall Square Research, “KSR Parallel Programming”, 1991.

[7] Gurd, J.R. et al., “A Framework for Experimental Analysis of Parallel Computing”, *Technical Report Series*, UMCS-93-2-3, Department of Computer Science, University of Manchester, 1993.

[8] Cole, M., “Algorithmic Skeletons:Structured Management of Parallel Computation”, *Pitman*, 1989.

[9] Chandy, K.M. and Kesselman C., “Parallel Programming in 2001” , *IEEE Software* , pp.11–20, November 1991.