事例ベース並列プログラミングシステム

山崎 勝弘　　安藤 彰一　　朝倉 啓之

立命館大学理工学部

概要

類似した並列プログラムの構造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システムについて述べる。並列アルゴリズムを四つのクラスに分類し、各クラス毎に並列プログラムを作成・収集して、事例ベースを作成する。事例はインデックス、スケレトン、プログラム、並列効果、及び履歴から成る。スケレトンにはタスク分割、同期、相互排除、並列化手法、スレッド使用法など並列プログラムの骨格となる最も重要な部分が含まれる。インデックスは並列プログラムの特徴を示し、並列効果は速度向上を示す。３次スプライン、エッジ抽出、細線化、包装アルゴリズムの問題に対して、類似した事例を事例ベースから検索し、それを修正・肉付けして並列プログラムを生成し、事例の再利用度を評価する。また、それらの実行構造を可視化して、デバッグと性能改善に利用する。

A Case-Based Parallel Programming System

Katsuhiro Yamazaki, Shoichi Ando and Hiroyuki Asakura

Faculty of Science and Engineering, Ritsumeikan University

Abstract

This paper describes how to reduce the burden of paprallel programming by utilizing releveant parallel programs. Parallel algorithms are devided into four classes and a case base for parallel programming is developed by retrieving parallel programs in each class. Cases consist of indices, a skeleton, a program, parallelization effects and a history. Skeletons include the most important issues such as task division, synchronization, mutual exclusion, parallelization methods and threads. Parallel programs for three dimensional spline, edge detection, thinning and package wrapping algorithm are developed by retrieving the most relavant case and adapting it to the given problem. Their execution structures are visualized for debugging and tuning.

1　はじめに

流体計算、気象予測などの大規模計算を適切な時間内で解くためには並列処理が必須であるが、並列処理は未だ専門家レベルにとどまっている。並列処理を広く普及させるためには、並列プログラミングを容易にするためのブレークスルーが必要不可欠であり、並列プログラムの再利用性と移植性を指向した種々の研究が進められている。

並列プログラミングでは、複数プロセッサ間の負荷均衡を図り、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ることが最も重要である。また、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る。現状では、C/Fortran に並列構文を追加した言語が用いられており、ユーザ自身がタスク分割や並列化を指定しなければならない。従って、逐次プログラムから並列プログラムへの変換、及び高性能の並列プログラムの開発に多大の労力を要している。

本研究では、事例ベース推論を並列プログラミングに適用して、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。すなわち、新たな問題に対して、類似した並列プログラムの構造を極力再利用し、不足する部分を補って、並列プログラムを作成する。この考え方は、初心者のみならず熟練プログラマでも一般的に行っている方法であり、これを体系化することにより、実質的に役立つシステムの構築を目指す。また、並列プログラムのデバッグとチューニングを支援するために、各プロセッサの稼働状況、同期などの実行状況を可視化する。

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 KSR１上での C プログラミング [1] を仮定する。仮想共有メモリ方式では、単一の仮想アドレス空間が与え

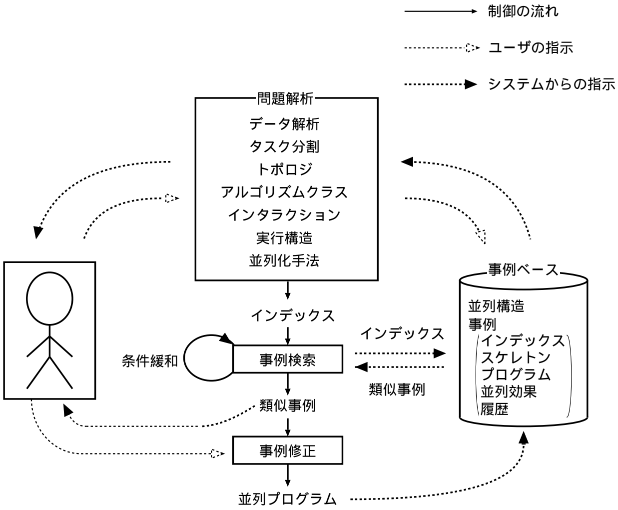


図 1: システム構成

られるので、分散メモリプログラミングの困難さを軽減させることができる。

本手法では並列プログラムの骨格をスケルトンとして用意する。スケルトンには、タスク分割、同期、相互排除、並列化手法、スレッド使用法など並列プログラムの最も重要な部分が含まれる。また、類似事例を検索するために、その問題の特徴をデータ構造、タスク分割、終了条件、アルゴリズム、同期、並列化手法などの観点からインデックス付けする。さらに、プロセッサ数を変化させたときの速度向上を実測し、並列効果とする。インデックス、スケルトン、プログラム、並列効果、及び履歴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、最も類似した事例を事例ベースから検索し、スケル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。

並列アルゴリズムは一般的に、分割統治法、プロセッサファーム、プロセスネットワーク、繰り返し変換の四つのクラスに分類される [2]。我々は各クラス毎に並列プログラムを作成して、並列プログラミング用事例ベースの構築を進めてきた [5][6]。また、類似事例の検索 修正法について具体的に検討し [4][7]、かつ、並列プログラムの実行状況を可視化した [9]。

本論文では、四つの並列アルゴリズムクラスに対する並列実行構造を示した後、インデックスの作成、類似事例の検索 ·修正法について述べる。また、3 次スプライン、細線化、エッジ抽出、包装アルゴリズムの並列プログラムを類似事例を修正して作成し、その実行状況を可視化する。さらに、事例の再利用度を評価し、検討する。

2 事例ベース並列プログラミング

図 1にシステム構成を示す。問題解析では、まず応用分野、仕様を規定し、データ依存性の有無、ループの終了条件を明らかにする。これらを基に、並列アルゴリズムのクラスを決める。各クラス毎に代表的な並列構造が事例ベースに格納されているので、決定された並列アルゴリズムのクラスに対する並列構造をユーザに提示する。ユーザは同期、並列化手法を基に、使用する並列構造を選択する。その構造に対する BACS 表現がシステムから与えられる。BACS[3] は主にアルゴリズムのトポロジ、プロセス構造、実行構造、相互作用の方式、及びデータ分散の観点から並列アルゴリズムを分類する。

システムはこれらのインデックスを用いて、最も類似した事例を検索する。類似事例が検索されると、それを構成するインデックス、スケルトン、プログラム、並列効果がユーザに提示される。スケルトン内には、タスク分割、スレッドの使用法、同期などと共に、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過程が記述されている。

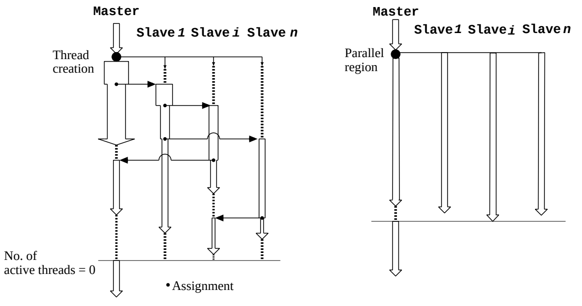
ユーザはスレッドの起動、スレッド間の同期などスケルトン内で再利用できる部分を極力再利用し、タスク分割など修正を要する部分を修正する。また、逐次プログラムが作成されている場合には、初期化、単位計算に利用する。

3 並列アルゴリズム

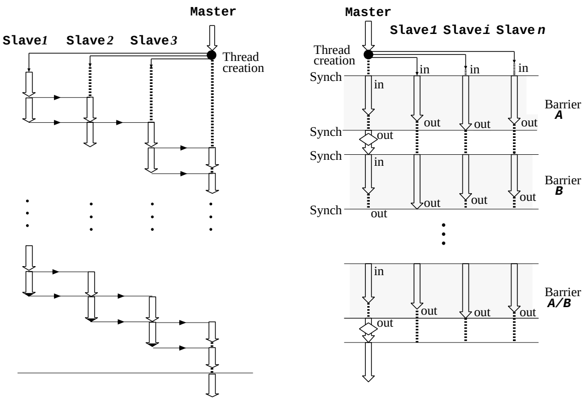
3.1 並列アルゴリズムの分類

Rabhi は並列アルゴリズムを四つのパラダイムに分類しており [2]、これが最も一般的な分類である。

- 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。部分問題の解は再帰的に結合され、最終結果が得られる。
- 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。マスターが問題を複数の独立したタスクに分割し、各タスクを各スレーブに実行させて、それらの結果を回収して統合する。
- 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。パイプライン処理に相当する。
- 繰り返し変換 (Iterative Transformation)
  各オブジェクトにオペレータ f を繰り返し適用し、終了条件を満たすまで各オブジェクトを変換する。



(a) 分割統治法　(b) プロセッサファーム



(c) プロセスネットワーク　(d) 繰り返し変換

図 2: 並列実行構造

3.2 並列実行構造

KSR 1 のスレッドは POSIX スレッドに準拠している。スレッドとは軽量プロセスであり、生成のためのオーバーヘッドが小さい。複数スレッドによる並列処理が可能であり、通常 1 スレッドが 1 プロセッサで実行される。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 [1]。パラレルリージョンでは、複数のスレッドが同一のコードセグメントを実行する。パラレルセクションでは、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。すなわち、前者は SPMD、後者は MPMD と考えられる。また、スレッドライブラリを用いて、ユーザ自身が直接スレッドを生成したり、消滅させることもできる。

3.1 の各アルゴリズムを KSR 1 上で実現すると、それらの並列実行構造は図 2のようになる。

分割統治法では、マスターからパラレルリージョンを起動し、複数のスレーブは他からタスクを割り当てられるのを待っている。マスターは自分のタスクの半分をスレーブ 1 に渡す。以下、同様にして、アイドルスレッドがあれば、各スレッドは自分のタスクの半分をそれに渡す。

プロセッサファームではマスターからパラレルリージョンを起動し、各スレーブで全く同一のタスクを実行する。プロセスネットワークではマスターから、top、middle、tail スレッドをスレッドライブラリで生成する。任意のスレッドは上位スレッドと下位スレッドとの同期が必要である。

表 1: 事例一覧

(a) 分割統治法	クイックソート 積分計算 コッホ曲線
(b) プロセッサファーム	マンデルブロ 文字列検索 (KMP 法) レイトレーシング
(c) プロセスネットワーク	LU 分解 バブルソート
(d) 繰り返し変換	高次方程式の解法 (DKA 法) 有向グラフの推移閉包

繰り返し変換ではスレッドライブラリを用いて複数スレッドを起動し、バリアチェックインして全スレッドで同一の計算を行う。バリアチェックアウトしてマスターが終了条件を判定する。2 つのバリアを用いているのは、マスターがチェックイン中は、計算を終了したスレーブスレッドがチェックアウトして、再度チェックインできるためである。分割統治法、プロセッサファームではスレッドライブラリを用いて、直接各スレッドを生成させる構造も可能である。

4 類似事例の検索・修正法

4.1 事例ベースの構築

表 1 に示すように、まず各アルゴリズムクラス毎に複数の事例を用意した [4][5][6]。これらの事例を用いて、事例修正による並列プログラミングの可能性について具体的に検討した [7][8]。すなわち、分割統治法ではクイックソートの事例を用いて、巡回セールスマン問題のプログラムを作成した。プロセッサファームでは、KMP 法から 3 次スプラインを、さらに 3 次スプラインからエッジ抽出のプログラムを作成した。プロセスネットワークではバブルソートから細線化を、繰り返し変換では DKA 法から包装アルゴリズムのプログラムを作成した。これらのうち、プロセッサファーム、プロセスネットワーク、繰り返し変換について、事例修正による並列プログラミングについて 5 章で述べる。

4.2 インデックスの作成

ユーザは以下の順に、次の各項目を作成する。

1. 応用
2. 仕様
3. データ構造
4. タスク分割
5. 終了条件
6. トポロジ
7. アルゴリズム
8. 並列化手法
9. インタラクション
10. BACS

まず、応用、仕様でその問題の定義などを記述する。データ構造、タスク分割、終了条件、トポロジは代表的なものを図と説明を用いてシステムが表示し、その中から選択するか、あるいはユーザが記述する。データ構造として配列（1次元、2次元など）、構造体が考えられる。タスク分割ではデータの分割（2次元配列なら行方向、列方向、メッシュ）と割り付け（コピー、ブロック、サイクリック）を決めなければならない。アルゴリズム、並列化手法、インタラクションは代表的な構造がシステムから提示されるので、その中から選択する。データ依存性の有無、ループの終了条件を基に、並列アルゴリズムのクラスを決める。例えば、データ依存、終了条件が共に無ければ、プロセッサファームを適用できる。データ依存はあるが、終了条件が無ければ、プロセスネットワークを使用できる。終了条件のみならば、分割統治法か繰り返し変換が適用できる。

各クラス毎に代表的な並列構造が事例ベースに格納されているので、決定された並列アルゴリズムクラスに対する並列構造をユーザに提示する。ユーザは同期、並列化手法を基に、使用する並列構造を選択する。その構造に対するBACS表現がシステムから与えられる。各項目ともシステムが提示した例からユーザが選択するだけでインデックスを記述できるようにし、簡単化を図る。

4.3　類似事例の検索法

システムはインデックスを基に類似事例の検索を行う。類似事例が見つかれば、その事例を表示し、完全にマッチする事例がない場合は、条件を緩和する。その際、類似事例が検索されるまで、次の順番で各項目を削除していく。

1. 応用
2. 仕様
3. 終了条件
4. データ構造
5. タスク分割
6. インタラクション
7. 並列化手法
8. BACS
9. トポロジ
10. アルゴリズム

この順番は各項目の重要性とスケルトンへの影響から決定されており、並列プログラムの構造に最も影響があるもの程、後ろになっている。最悪の場合でも、最後のアルゴリズムの項目は4つから選択するようになっているので、必ず、いずれかの事例にマッチするようになる。

表 2: 3次スプラインのインデックス

応用	数値計算
仕様	複数の座標点の補間を行う.
△ ソースデータ	構造体1次元配列 point
△ 結果データ	構造体1次元配列 result
△ タスク分割	point, 要素ブロック, ブロック result, 要素ブロック, ブロック
○ 終了条件	なし
○ トポロジ	マスター&ワーカー
○ アルゴリズム	プロセッサファーム
○ 並列化手法	パラレルリージョン
○ インタラクション	なし
○ BACS	C^t

4.4　類似事例の修正法

スケルトン内には、タスク分割、スレッドの使用法、同期などと共に、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過程が記述されている。スレッドの起動と終了、スレッド間の同期などはスケルトン内の情報を再利用できる。従って、変数初期化、タスク分割、計算、結果回収などを主に修正する必要があり、これは通常ユーザにより手動で行われる。逐次プログラムが作成されている場合には、変数初期化、単位計算に逐次プログラムの対応部分を埋め込むことができる。

5　事例修正による並列プログラミングと可視化

5.1　3次スプライン

5.1.1　問題の定義

複数の座標点の各区間を補間して、滑らかな曲線となるような座標点を求める。

5.1.2　並列化の解析

ソースデータ、結果データを実数型構造体の1次元配列とする。それらをスレッド数分のブロックに分割し、各々を各スレッドに割り当てる。各スレッドでは、補間を行う各区間で3次パラメータ表現した座標点の1・2次微係数が一致し、かつ座標点を通ることを用いて、各座標点の3次パラメータ式を求める。スレッド間の同期、終了条件は無いのでアルゴリズムはプロセッサファームを使用できる。並列化手法はパラレルリージョン、全スレッドで同じ計算を行うのでBACS表現は C^t (全スレッドでの同一計算) となる。表2に3次スプラインのインデックスを示す。類似事例として文字列照合 (KMP法)

が検索される。○ がついた項目は KMP 法と同じ、▵ がついた項目は類似していることを示す。

5.1.3　事例の修正

KMP 法の事例を修正して 3 次スプラインの並列プログラムを作成する。以下に KMP 法のスケレトンを示す。

```
main
{  input the number of threads;
   get a team ID;
   activate a parallel region( team ID, work );
   print the results;
}

work
{  get a thread no.;
   len = length of TEXT for each thread
   for ( the no. of PATs  )
   { for ( k = 0; k <= len + length of PAT -1; k++ )
         {  CALCULATION;
            store the results into searchbox;
} } }
```

main からパラレルリージョンを起動して、全スレッドで work を実行する。work ではタスク分割の後、単位計算を行う。スレッド、同期はスケレトンをそのまま再利用でき、単位計算は逐次プログラムを殆ど再利用している。タスク分割では KMP 法の重複ブロックをブロックに修正し、新規作成も必要とした。

5.2　エッジ抽出

5.2.1　問題の定義

2 値化画像から輪郭となる部分を抽出する。

5.2.2　並列化の解析

ソースデータ、結果データを整数型 2 次元配列とする。複数行から成るブロックに分割し、ブロック毎にスレッドに割り当てる。値が 1 の点の上下左右の値がすべて 1 であるならその点はエッジではなく、値が 0 の点が 1 つでもあればその点はエッジとなる。スレッドではそれぞれ割り当てられた範囲でエッジを抽出する。各スレッドは独立に計算でき、スレッド間の同期は必要なく、また終了条件もないので、アルゴリズムはプロセッサファームを使用できる。並列化手法はパラレルリージョン、BACS 表現は C^t となる。類似事例として 3 次スプラインが検索される。表 3 にエッジ抽出のインデックスで、3 次スプラインのそれと異なる項目のみを示す。

表 3: エッジ抽出のインデックス

応用	画像処理
仕様	2 値化画像の輪郭を抽出する。
▵ ソースデータ	整数型 2 次元配列 indata
▵ 結果データ	整数型 2 次元配列 outdata
▵ タスク分割	indata　, 行ブロック, ブロック
	outdata , 行ブロック, ブロック

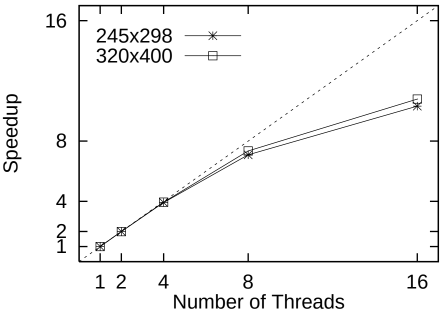


図 3: エッジ抽出並列効果

5.2.3　事例の修正

表 2 と表 3 はデータ構造とタスク分割が異なるので、タスク分割の一部を修正し、単位計算を肉付けすることにより作成できた。図 3 にエッジ抽出の並列効果を示す。

5.3　細線化

5.3.1　問題の定義

図 4に示すように、2 値画像の線を細くすることにより、その中心線を求める。

5.3.2　並列化の解析

ソース・結果データ（data）は整数型 2 次元配列である。17 通りのマスクパタンを用意し、それとマッチする場合には、中央の点を 1 から 0 に変換する [8]。data を行方向または列方向のブロックに分割する。各スレッドで 1 つのブロックを図 5に示すように、下向き、右向き、上向き、左向きにマッチし、これを繰り返す。ウェーブフロント（左上から右下に波状に計算を進める）を用いる。あるマッチでは以前のマッチの結果を用いるので、プロセスネットワークを用いる。トポロジーはパイプである。スレッドライブラリでスレッドを生成し、signal/wait でスレッド間の同期をとる。この問題に最も類似し

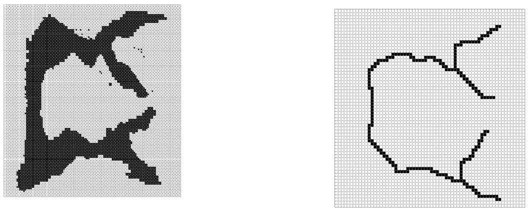


図 4: 細線化

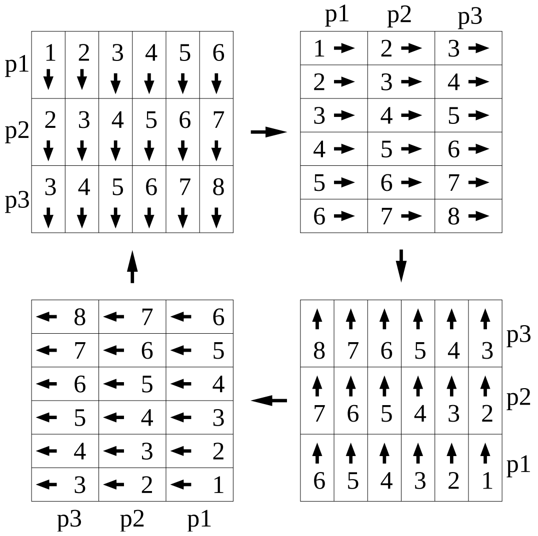


図 5: タスク分割

た事例として、バブルソートが検索される。細線化のインデックスを表 4 に示す。

5.3.3 事例の修正

main ではタスク生成部分を除いて、スケレトンを殆どそのまま再利用できる。タスク分割では、スレッドの生成順序を変更する。すなわち、バブルソートでは tail、middle、top の順にスレッドを生成しているが、これを top、middle、tail の順に変更する。細線化の初期設定や結果回収は逐次プログラムを再利用する。上位、下位スレッドとの同期部分は再利用できる。タスク分割では一部修正が必要である。生成された細線化プログラムの並列効果を図 6に示す。

5.4 包装アルゴリズム

5.4.1 問題の定義

ある点集合において、全ての点を包み込むことができる複数の点を直線で結んだ凸多角形を求める。

表 4: 細線化のインデックス

応用分野	画像処理
仕様	画像データから中心線を抽出
△ ソースデータ	整数型 2 次元配列 $data$
△ 結果データ	整数型 2 次元配列 $data$
△ タスク分割	
$data$, 行/列のブロック , ブロック	
○ 終了条件	なし
○ トポロジ	パイプ
○ アルゴリズム	プロセスネットワーク
○ 並列化手法	スレッドライブラリ
○ インタラクション	シグナル&ウェイト
○ BACS	
top_thread	$Fix\{C, I^{p(lower\_thread)}\}$
middle_thread	
$Fix\{I^{p(upper\_thread)}, C, I^{p(lower\_thread)}\}$	
tail_thread	$Fix\{I^{p(upper\_thread)}, C\}$

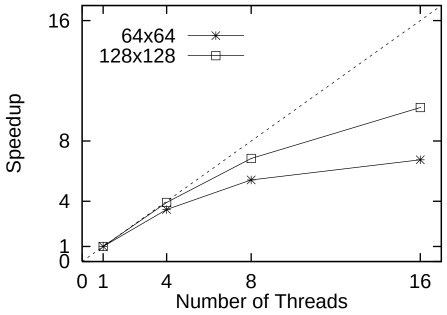


図 6: 細線化並列効果

5.4.2 並列化の解析

ソース ·結果データ (point) は実数型構造体配列である。任意の点と他の全ての点のなす角度は独立に計算できるので、point をサイクリックに分割する。各スレッドでは計算範囲は異なるが全く同じ計算を行う。また、始点と終点が一致するまで計算を繰り返すので、繰り返し変換を用いる。パラレルリージョンを用いてマスタから複数のスレッドを生成し、スレッド間の同期をバリアでとる。この問題の類似事例として DKA 法が検索される。包装アルゴリズムのインデックスを表 5 に示す。× の項目は DKA 法と異なることを示す。

5.4.3 事例の修正

main ではデータ入力とデータ出力を除いて、DKA 法のスケレトンをそのまま利用できる。worker ではバリアチェックイン、バリアチェックアウト、スレッド番号取得を再利用できる。タスク分割は類似事例

表 5: 包装アルゴリズムのインデックス

応用分野	数値計算
仕様	全ての点を包み込むように点を直線で結んだ凸多角形を計算
△ ソースデータ	構造体 1 次元配列 $point$
△ 結果データ	構造体 1 次元配列 $point$
○ タスク分割	$point$, 要素 , サイクリック
× 終了条件	始点=終点
× トポロジ	部分照合
○ アルゴリズム	繰り返し変換
○ 並列化手法	パラレルリージョン
○ インタラクション	バリア
○ BACS	
$Cond\{I^t_{barrier(A/B)}\{C^t\},C^{p(master)}\}$	

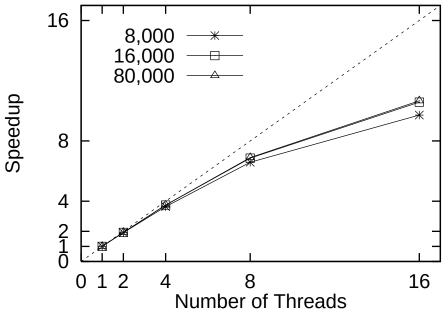
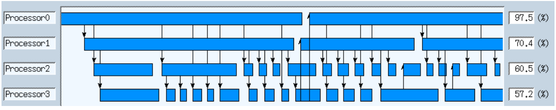


図 7: 包装アルゴリズム並列効果

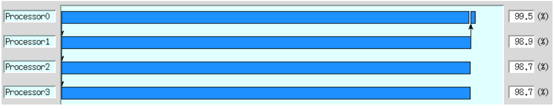
と同じサイクリック分割であるので、容易に修正できる。単位計算では、共有変数を各スレッド毎の局所変数に変換し、それに伴う修正を行う。生成された包装プログラムの並列効果を図 7に示す。

5.5 並列実行の可視化

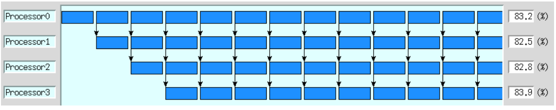
図 8にクイックソート [6]、エッジ抽出、細線化、包装アルゴリズムの可視化を示す。これらは図 2 に示した各々の並列実行構造に対応する。いずれも 4 個のスレッドで実行している状況であり、右端に各プロセッサの稼働率を示す。また、各図の右下に表示時の倍率を示している。クイックソートでは問題が分割されていく様子が分かる。エッジ抽出では、パラレルリージョンを用いて、全スレッドで同一プログラムを実行するので、オーバーヘッドが最も少ない。細線化ではパイプラインが規則正しく動いている。包装アルゴリズムでは、繰り返し計算に対して、バリア同期のオーバーヘッドが大きい。一般的に、分割統治法（クイックソート）と繰り返し変換（包装アルゴリズム）ではプロセッサ稼働率が高くなく、同期のオーバーヘッドをいかに低減させるかが重要である。このように、各アルゴリズムの挙動の理解を深め、どの部分をチューニングすれば良いかが分かる。



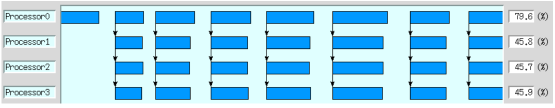
(a) クイックソート (x500)



(b) エッジ抽出 (x2)



(c) 細線化 (x1000)



(d) 包装アルゴリズム (x2000)

図 8: 可視化

6 事例修正並列プログラミングの評価

6.1 事例の再利用度

今までの四つのプログラムについて、スケレトン再利用、スケレトン修正、新規作成の各々の行数を表 6 に示す。スレッドと同期は 3 次スプライン、エッジ抽出、包装アルゴリズムでそのまま再利用できている。細線化のスレッドの修正 4 は、スレッドの生成順序を middle、top、tail から top、middle、tail に変更している。また、同期の新規 4 は、4 方向各々の計算終了確認のために追加された。

タスク分割はエッジ抽出と細線化で一部を再利用できているが、修正と新規作成が必要である。エッジ抽出では 3 次スプラインとデータ構造が近いので、一部を再利用している．単位計算は殆どを再利用できている。包装アルゴリズムでは、共有変数を各スレッドの局所変数に変更するような変数に関連した修正と新規作成を行っている。

並列プログラミングではどの並列アルゴリズムを適用できるか、及びそのアルゴリズムをどのような実行構造で実現するかが最も重要である。すなわち、図 2に示した並列実行構造を決定し、それらをスレッドや同期機構でいかに実現するかが最も重要、かつユーザにとって困難な部分である。細線化のスレッ

表 6: 事例の再利用度の評価

（行数）

		スレッド	同期	タスク分割	単位計算
3次	再利用	8	1	0	107
	修正	0	0	11	3
	新規	0	0	12	0
エッジ	再利用	8	1	6	60
	修正	0	0	0	5
	新規	0	0	15	0
細線化	再利用	1	36	19	26
	修正	4	0	22	0
	新規	0	4	8	0
包装	再利用	10	7	0	67
	修正	0	0	2	4
	新規	0	0	0	18

ド生成では、この事例を新たに登録することにより、そのまま再利用できる。従って、事例が充実すれば、スレッドと同期についてはスケレトンをそのまま再利用でき、並列構造の作成を支援できる。これらの構造が決まれば、後は単位計算、変数初期化を逐次プログラムから埋め込めばよい。タスク分割は最も修正を要する部分であり、ユーザが苦労するところである。現在までの実験では、プロセッサファームの実行構造が最も単純であり、かつ並列化しやすいので、事例修正が最も容易である。

6.2 事例検索

事例検索時に、インデックスの優先順位を導入した。本論文で示した四つのプログラムでは、アルゴリズム、並列化手法、インタラクション、BACS は全て一致した。アルゴリズムクラスと並列実行構造を決めることが最も重要であり、これにより並列プログラムの骨格が決まる。一方、データ構造とタスク分割は修正が必要であった。並列化のためのそれらの可能性は限定されているので、全ての方式を準備し、ユーザから選択させることが可能であろう。現在、事例数が少ないが、事例ベースの充実と共に、それらの問題点を軽減できると考えられる。

6.3 並列効果

並列効果は作成された並列プログラムをプロセッサ１台で実行した場合に対する速度向上を示している。生成された並列プログラムは、プログラマによる修正が行われているので、プログラマ自身が作成したプログラムと比較しても遜色がない。

7 おわりに

本論文では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、新しい問題に対して類似した事例を検索・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。３次スプライン、エッジ抽出、細線化、包装アルゴリズムの並列プログラムを類似事例を修正して作成し、スレッド、同期は再利用でき、タスク分割は修正が必要であることを示した。現在、事例ベースの充実を図ると共にシステム全体を構築中である。今後の課題として、タスク分割の簡単化、可視化機能の充実、及び事例の自動修正の検討などが必要であろう。

謝辞

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. 株式会社に深謝する。本研究に関して、有益なご助言を頂いている新實 治男先生（京都工芸繊維大学）、John Gurd 先生（マンチェスター大学）、上嶋 明氏（本学研究生）に感謝する。

参考文献

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

[2] 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.

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

[4] 松田浩一, 山崎 勝弘, “並列プログラミング用事例ベースの作成と類似事例の検索・修正法”, 情処学第 *52* 回全大 *3L–4*, 6–113, 1996.

[5] 安藤 彰一, 松田浩一, 山崎 勝弘, “並列プログラミング用事例ベースの作成”, 第 *40* 回システム制御情報学会研究発表講演会, 4002, 1996.

[6] 山崎 勝弘, 松田浩一, 安藤 彰一, “並列プログラミング用事例ベースの構築”, 信学技報 *CPSY96–39*, pp.1–6, 1996.

[7] 山崎 勝弘, 松田浩一, 安藤 彰一, “類似事例の修正による並列プログラミング”, 情処学第 *53* 回全大 *2E–4*, 1–327, 1996.

[8] 山崎 勝弘, 安藤 彰一, 朝倉 啓之, “事例ベース並列プログラミング”, 第 *38* 回プログラミングシンポジウム, pp.155–165, 1997.

[9] 朝倉 啓之, 安藤 彰一, 山崎 勝弘, “事例修正による並列プログラミングと並列実行の可視化”, 情処学第 *54* 回全大 *6P–5*, 1–373, 1997.