卒 業 論 文

事例ベース並列プログラミングの実験と評価 (I)

氏　　名 ： 徳山 美香
学 籍 番 号 ： 2210940225-0
指 導 教 員 ： 山崎 勝弘 教授

提　出　日 ： 1998 年 2 月 20 日

立命館大学 理工学部 情報学科

摘要

本論文では、過去の類似した並列プログラムを極力再利用して、並列プログラム事例ベース作成を行った。

並列処理の応用は、様々な分野でみられる。大量の数値処理装置として、物理計算、シュミレーション、画像処理等に応用されている。しかし、並列処理の困難さとして、仕事の分割とプロセッサの割り当て、仕事間の相互作用の実現の問題が、加わる上に、仕事間の関係とプロセッサの接続の対応に関する問題がある。

本研究では、事例ベース推論の考え方を用いている。類似した事例を利用して並列プログラミングの負担を軽減させようというものである。

並列アルゴリズムは、プロセスネットワーク、繰り返し変換、プロセッサファーム、分割統治法の４つに分けられる。

そのうち、繰り返し変換、プロセッサファームで並列プログラムを作成した。作成したプログラムに対し、インデックス、スケレトン、プログラム本体を一つのの事例とする。並列プログラムの骨格をスケレトンとして用意する。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により、表されている。また、インデックスにより、その問題の応用、仕様、終了条件、並列アルゴリズム、データ構造、タスク分割、インタラクション、トポロジ、並列化手法を示している。さらに、事例の再利用の評価をおこなっている。再利用の評価では、類似したスケレトンと逐次プログラムを用いて評価をおこなっている。スレッド、同期、タスク分割において、再利用、修正、新規それぞれプログラムの行数によって、評価を行い利用度を検討する。

目 次

1 はじめに **1**

2 並列プログラミングと並列アルゴリズム **3**
2.1 並列アルゴリズムの分類 3
2.1.1 一般的分類 3
2.2 並列実行構造 6
2.2.1 インタラクション 6
2.3 並列プログラミング 8
2.3.1 並列構造の分類 8
2.4 BACS 表現 11

3 事例ベース並列プログラミングシステム **12**
3.1 システム概要 12
3.2 事例ベースの構築 13

4 事例ベース並列プログラミング **14**
4.1 BM 法 14
4.2 ロンバーグ積分法 16
4.3 n クイーン 19

5 事例の再利用度の評価 **20**
5.1 KMP 法を類似事例とした再利用度評価 20
5.1.1 評価と考察 21
5.2 DKA 法を類似事例とした再利用度評価 21
5.2.1 評価と考察 22

6 おわりに **23**

謝辞 **24**

参考文献 **25**

図 目 次

1 プロセッサファーム 4
2 繰り返し変換 4
3 プロセスネットワーク 5
4 分割統治法 5
5 barrier 同期 6
6 シグナルウエイト 7
7 mutex による相互排除 7
8 並列構造 (プロセッサファーム) 8
9 並列構造 (分割統治法) 9
10 並列構造 (繰り返し変換) 10
11 並列構造 (プロセスネットワーク) 10
12 システム構成 12
13 タスク分割 14
14 BM 法並列効果 16
15 ロンバーグ積分 17
16 ロンバーグ積分法の並列効果 19

表 目 次

1 BM 法インデックス 15
2 ロンバーグ積分インデックス 18
3 KMP 法再利用度 21
4 DKA 再利用度 22

1 はじめに

情報処理の世界では処理内容が高度かつ多様化するにしたがい、広く分散処理が行われるようになった。これまでのように個々の演算回路の高速化を追及するよりも、多数のプロセッサに仕事を分担させて並列処理を行うことによってそれほどコストをかけずに高性能化を行う並列計算機が注目されるようになった。

並列処理にはこれまでの逐次処理とは根本的に違った考え方が必要であり、現実の問題を並列計算機にかけて並列処理するには数々の新しい手法を使用しなければならない.
単一の問題をどのように分割すればよいか、それらを個々のプロセッサにどのように割り当てればよいかなど、いずれもこれまでの計算機では経験しなかったことが問題になる。また、並列プログラミングでは、プロセス間の同期、相互排除、スレッドなどに熟知している必要がある。並列処理によって、その効果を大きなものにするには、応用の対象を明確にして、アルゴリズムの記述、プロセッサの構成を、それに適合したものにすることが大切である。問題に内在する並列性を、効率よく、物理レベルのプロセッサに写像するには、対象とするデータの論理的な構造と、処理手順としての機能の関係をよく把握する必要がある。あるプロセッサ構成をもつ並列処理システムが与えられたとき、並列処理の効果を、大きくするには、その構成に適した問題を扱うか、問題を解くアルゴリズムをその構成に適するように工夫するすることが要求される。

そこで、事例ベース推論 [2][3] (CBR :Case-Based Reasoning) をもとに並列プログラミングの負担を軽減させる方法を提案している。獲得した知識を再利用可能な事例として保持されており、新たな問題を解決しようというものである。事例ベース並列プログラミングシステムでは、新たな問題に対して過去の類似事例を事例ベースから検索し、肉付け、修正を行い、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ている。

本研究では、並列プログラムを作成する際に、類似した並列プログラムを再利用し、出来上がったプログラムを事例のひとつとして事例ベースに加える。そして、類似事例と新たに作成した事例とを比較し、再利用度の評価を検討する。作成したプログラムに対し、インデックス、スケレトン、プログラム本体を一つのの事例とする。並列プログラムの骨格をスケレトンとして用意する。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により、表されている。また、インデックスにより、その問題の応用、仕様、終了条件、並列アルゴリズム、データ構造、タスク分割、インタラクション、トポロジ、並列化手法を示している。さらに、プロセッサ台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. 事例の再利用の評価手順として過去の類似事例のスケレトン、逐次プログラムなどから、再利用可能な部分、修正必要な部分、新規作成の各々の行数を示す。

本論文では、第 2 章で、並列プログラミングと並列アルゴリズムの分類 [5] を説明する。第 3 章では、事例ベースの構築と並列プログラミングシステムの説明を述べ、第 4 章では、作成した事例をアルゴリズムごとに紹介する。第 5 章では、事例の再利用度を評価し検討する。

第 4 章の事例ベースの作成において、仮想共有メモリ型並列マシン KSR-1[6] で BM 法、ロンバーグ積分法、n クイーンを類似事例を用いて並列プログラムを作成をし、実行している。

BM 法 [9] は、文字列照合の方法のひとつである。KMP 法と同じように PAT の構造情報を利用するもので、PAT の後ろから照合させていく。タスク分割の対象となるのは、TEXT データである。KMP 法を類似事例としてプロセッサファームで並列化をおこなった。

ロンバーグ積分法 [8] では、数値積分法のひとつで、ある値を終了条件として定め、その範囲内で繰り返し計算を行う方法である。台形公式により初期値を求め、その値をプロセッサ台数分で分割し、ロンバーグ補間によって並列に計算を繰り返し行っている。DKA 法を類似事例として繰り返し変換で並列化を行った。

2　並列プログラミングと並列アルゴリズム

2.1　並列アルゴリズムの分類

2.1.1　一般的分類

処理手順の分割法
並列アルゴリズムとは、次のような性質をもつ計算手順をいう。

1. 問題を解決する手順がいくつかの仕事に分割されている。
2. それぞれの仕事の計算手順が定義されている。
3. 仕事間の実行の順序関係、とくに同時に実行できる関係が示されている。

ここでキーポイントになるのは、できるだけ多くの仕事が同時に実行できるように手順を分割することである。

- データの並列性 (負荷分割） 対象とするデータを分割して、かく部分に同質のプロセッサが負荷を分担することにより、同一の操作を同時に行わせる。

ハル大学の Rahi[5] は計算手順をデータ参照パターンの規則性により、並列アルゴリズムを分割統治法、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 4 つのアルゴリズムに分類している。

1. プロセッサファーム

   図 1 に示すように、プロセッサファームは、初期段階で仕事の分割を行う。分割された仕事は、一つのマスターが各スレーブに仕事の指示を与えそれぞれの仕事の終了を待って、次の指示を設定する。各スレーブは、共有データを介して独立に仕事を進める。そして最後に各スレーブの計算結果を回収し、それを解とする方法である。

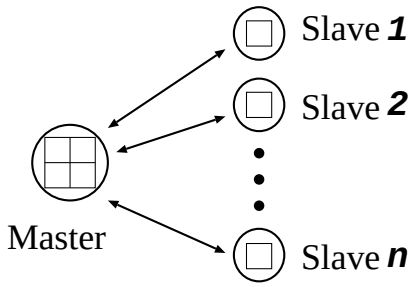


図 1: プロセッサファーム

2. 繰り返し変換

図 2各スレーブが、共有データを介して、比較的独立に仕事を進める。与えられた問題をオブジェクトに分ける。前ステップで計算結果は別のプロセッサで共有して参照され、ある与えられた終了条件を満たすまで繰り返し計算される。

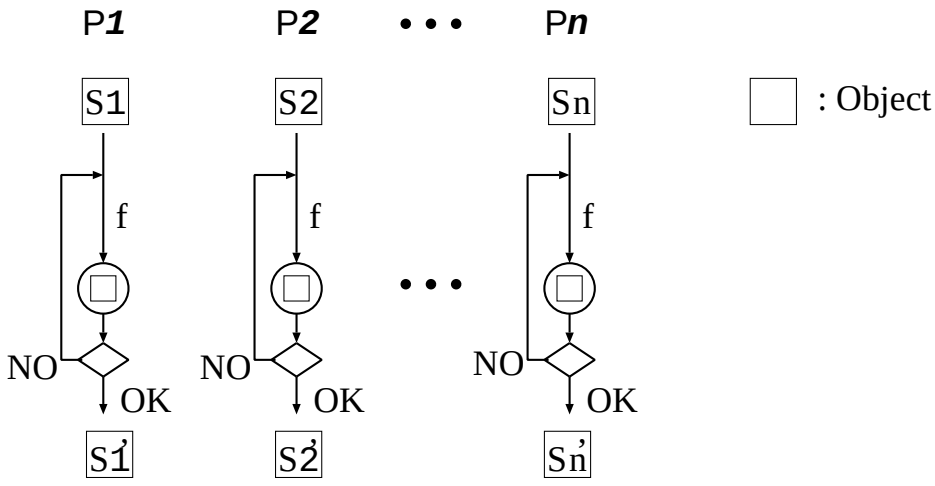


図 2: 繰り返し変換

3. プロセスネットワーク

図 3 データの流れが制御の流れに対応する。与えられた問題に対して、計算を複数のステージに分け、あるステージで計算された結果が次のステージに移り、また前のステージでデータが入力され、計算が行われ並行に実行されていることになる。そうすることで、結果データは各ステージを流れパイプライン処理される。

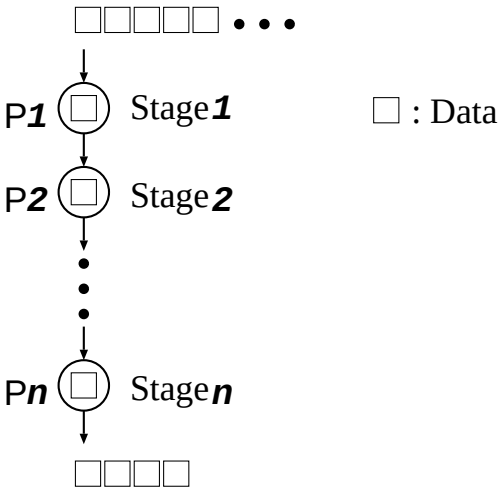


図 3: プロセスネットワーク

4. 分割統治法

図 4親が子に仕事を分割して与え、その結果を統合する。一つを二つ以上の仕事に分割されて、その結果を待って統合する形態である。木構造の通信パターンをもつ。

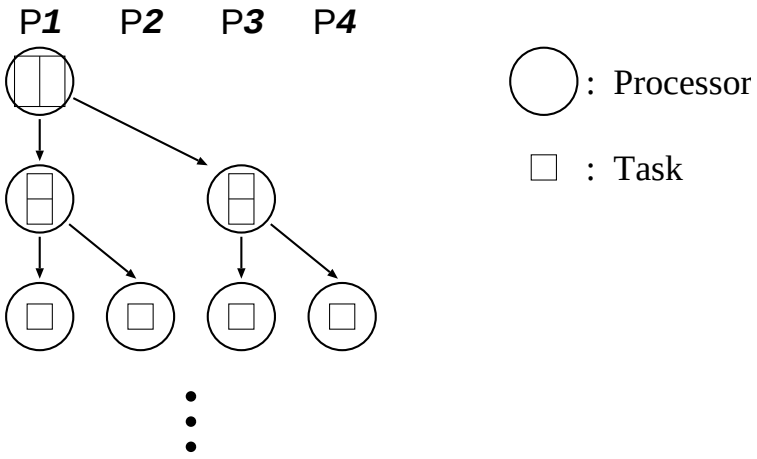


図 4: 分割統治法

2.2 並列実行構造

2.2.1 インタラクション

1. barrier
   barrier は複数の pthread をメンバーとする。ただ一つの pthread が、barrier のマスターとなる。それ以外の pthread は barrier のスレーブとなる。この barrier 変数を使用して、並列処理の同期をとることができる。マスターが barrier へチェックインするまでスレーブは実行待ちとなり、マスターは、全てのスレーブがチェックアウト (計算終了) するまで、実行を待ち合わせる。

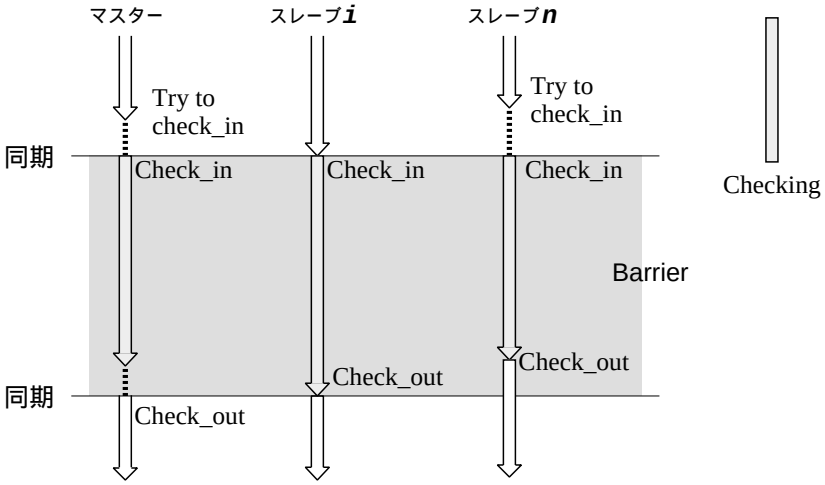


図 5: barrier 同期

2. シグナルウエイト
   スレッド 1、スレッド 2 はそれぞれフラグを待つ。もし、スレッド 1 が先に処理を終えていたら、スレッド 2 の状態をフラグ変数から読み取り、フラグが 1 (ビジー) であれば、スレッド 1 フラグを 0 (アイドル) にして処理を待つ。読み取ったフラグが 0 ならば、スレッド 1 のフラグを 1 にして実行状態にしながら同期をとる。

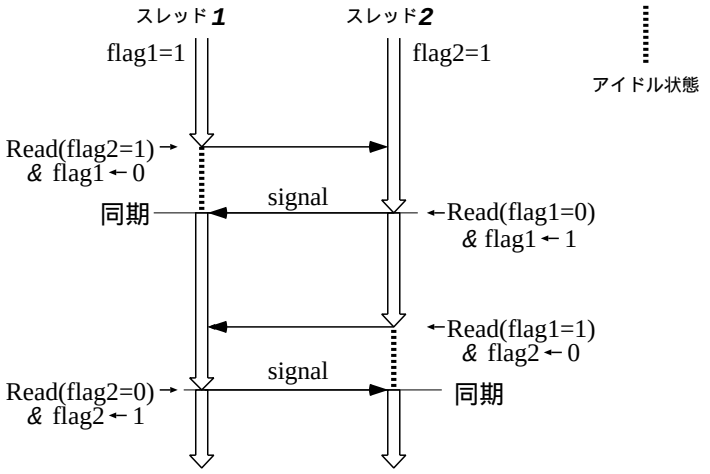


図 6: シグナルウエイト

3. mutex (相互排除)

複数の pthread が、ある一つの共通変数にアクセスし、更新すると、データの整合性が壊れないための仕組みである。ある pthread が共通変数を更新している時には、他の pthread は、この共通変数にアクセスできない。mutex は、データアクセスのためのコードフラグメントを同期化させることにより、共有データへのアクセスを連続的にするためのものである。

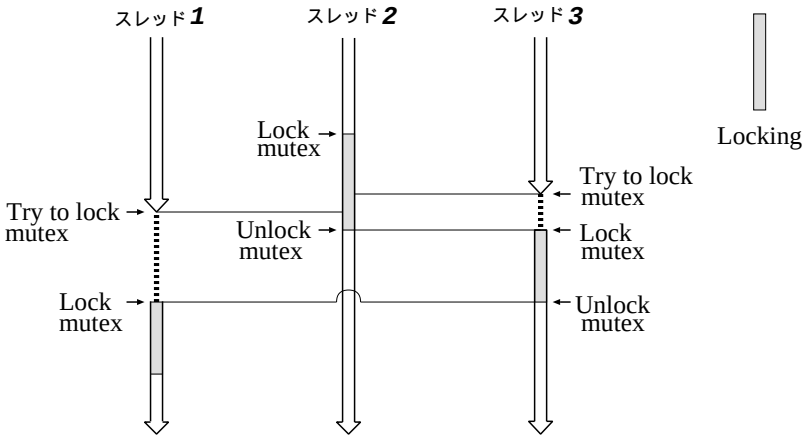


図 7: mutex による相互排除

2.3 並列プログラミング

どの分野であれ、プログラムを書いて並列マシン上で実行させる際、並列コンパイラによるか、並列計算モデルに基づいた並列型言語で書くかによって、処理における仕事の分担は変わってくる。

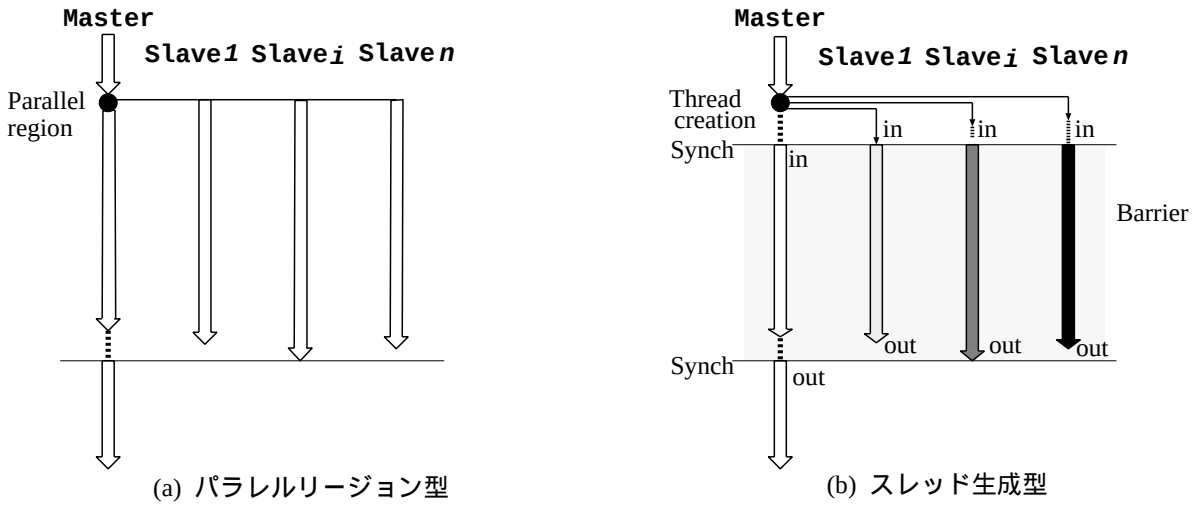
1. 並列コンパイラ
   逐次プログラムから並列に実行可能な部分を見出し、タスクに分解する。タスクの順序付けをして、可能な限りプロセッサの割り当てを行う。結果として、並列マシンのコードを生成するが、その中には、プロセッサ間の同期や通信のためのコードが含まれる。これは、これまでの逐次プログラムの習慣を残し、ユーザーに新しい負担をかけない、既存のソフトウエアの蓄積を活かすという利点を持つ。

2. 並列型言語
   並列型言語でプログラミングする場合は、問題の特徴を活かした並列アルゴリズムを工夫し、並列性を陽に示したプログラムを作成することが出来る。ユーザーがプログラム上で仕事の分割、プロセッサ割り当てや、仕事間の相互作用を指示しなければ、ならない。

2.3.1 並列構造の分類

1. プロセッサファーム

   図 8よりパラレルリージョン型とスレッド生成型とがある。パラレルリージョン型では、始めにプロセッサ台数を指定し、自動的に同期がとられるが、スレッド生成型は、スレッドの生成及び消去をプログラム内で指定することが出来る。

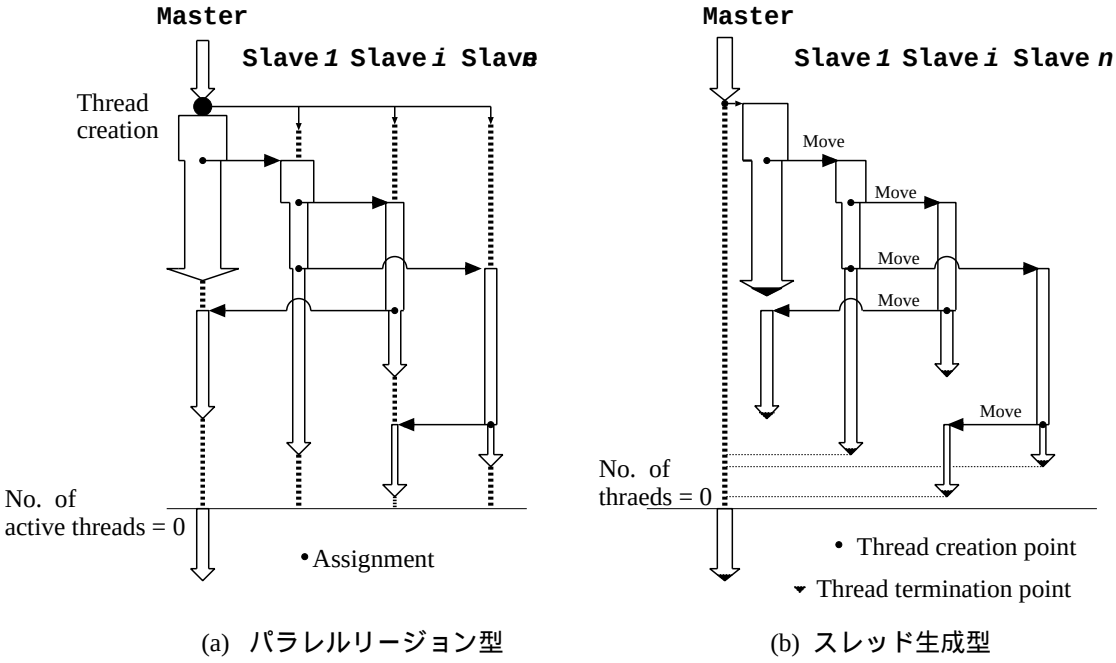


(a) パラレルリージョン型　　(b) スレッド生成型

図 8: 並列構造 (プロセッサファーム)

2. 分割統治法

図 9のようにパラレルリージョン型とスレッド生成型とがある。パラレルリージョン型では、始めにプロセッサ台数を指定し、受け取ったタスクを 2 分割し、一方を別のスレッドに割り当て、もう一方を自分に割り当てる。アイドルスレッドの検索時にはミューテックスを用いる。スレッド生成型では、一つのスレッドから始まり、分割タスクの生成と共にスレッドが生成される。



(a) パラレルリージョン型　　(b) スレッド生成型

図 9: 並列構造 (分割統治法)

3. 繰り返し変換

図 10の繰り返し変換では、スレッド生成型であり、各スレッドは割り振られたタスクを変換処理していくが、そのときバリア同期で囲まれている。マスターのチェックイン、チェックアウトにより、スレーブの実行を制御する。変換結果がある条件を満たすまで、繰り返し計算が行われる。

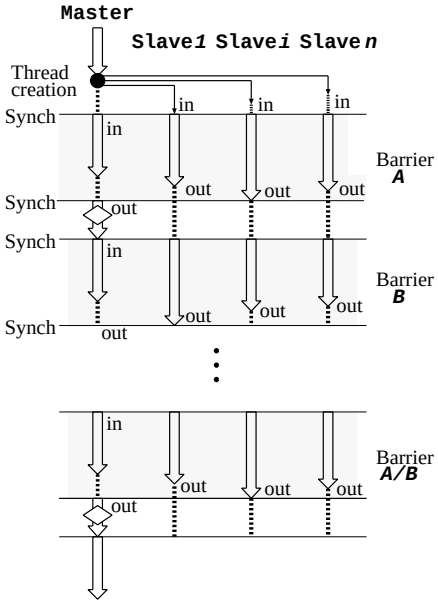


図 10: 並列構造 (繰り返し変換)

4. プロセスネットワーク
図 11のプロセスネットワークでは、FSFE と FSL の二つの処理方法がある。FSFE はマスターがスレーブ生成後、スレーブ 1、スレーブ 2、スレーブ 3 とデータが流れる。また、FSLE はマスター、スレーブ 3、スレーブ 2、スレーブ 1 とデータが流れる。ステージと次のステージとの間では、常に同期が取られる。

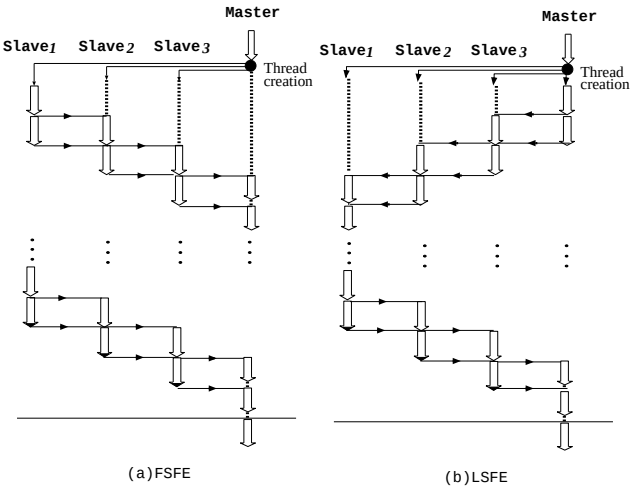


図 11: 並列構造 (プロセスネットワーク)

2.4 BACS 表現

アルゴリズムの分類 , それを、構成しているイベントなどを表現するために、BACS(Basel Algorithm Classification Scheme) 表現を用いる。

1. イベント

   独立した計算を表し、他との影響はないものとして、計算イベント (C)
   同期、相互排除を表すインタラクションイベント (I)、デーモンコールイベントがある。
   イベントとプロセスとの関係において、2 通りのイベント参加方法がある。トータルイベントとパーシャルイベントがある。トータルイベントとは、

   $$C^t$$

   と表され、全プロセスがそのイベントに参加することである。
   パーシャルイベントとは、

   $$C^p$$

   と表され、プロセッサの一部のみがそのイベントに参加することである。参加するプロセスの情報を表すものとして、次の 2 つがある。

   - 定数の定義
     プロセスの参加数表示。参加プロセスが 1 個の場合は、$C^{p(1)}$ と表せられる。
   - 指定子の定義
     複数のプロセス内でも、ある条件に添ったプロセスのグループを表すことができる。そのプロセスグループを group と定義するとすれば、その指定子をもったプロセスのみがイベントに参加でき、 $C^{p(group)}$ と表される。

2. 固定回の繰り返しを行なうループ

   繰り返しの回数は実行以前に分かっている . FOR-Loop 文に相当する .

   $$FIX(\text{イベント}, \cdots)$$

3. 条件によるループ

   繰り返しの回数は実行時の状態により決定される . WHILE-Loop 文に相当する .

   $$COND(\text{イベント}, \cdots)$$

3　事例ベース並列プログラミングシステム

3.1　システム概要

事例ベースに格納されている事例には、インデックス、スケレトン、プログラムをひとつの事例としている。システムは事例のインデックスにより最も類似した事例を検索する。ユーザー側から、その問題の応用、仕様、並列アルゴリズムのなどをインデックスの項目にそって指定する。ユーザーが作成したインデックスにそって類似事例が検索される。検索された事例のスケレトンを再利用し並列プログラムを作成することができる。スケレトン内には、スレッド使用法、同期、タスク分割、単位計算が示されており、ユーザーはその問題に応じて逐次プログラムから、初期化、変数、単位計算を埋め込み、その問題と並列アルゴリズムに応じて、タスク分割を肉付け、修正をして並列プログラムを作成する。タスク分割は最も修正を必要とする部分である。過去に類似した事例がない全く新しい問題でも、条件緩和すれば、その問題に近い事例が取り出せられる。問題にマッチしない部分は手動で修正を行わなければならない。

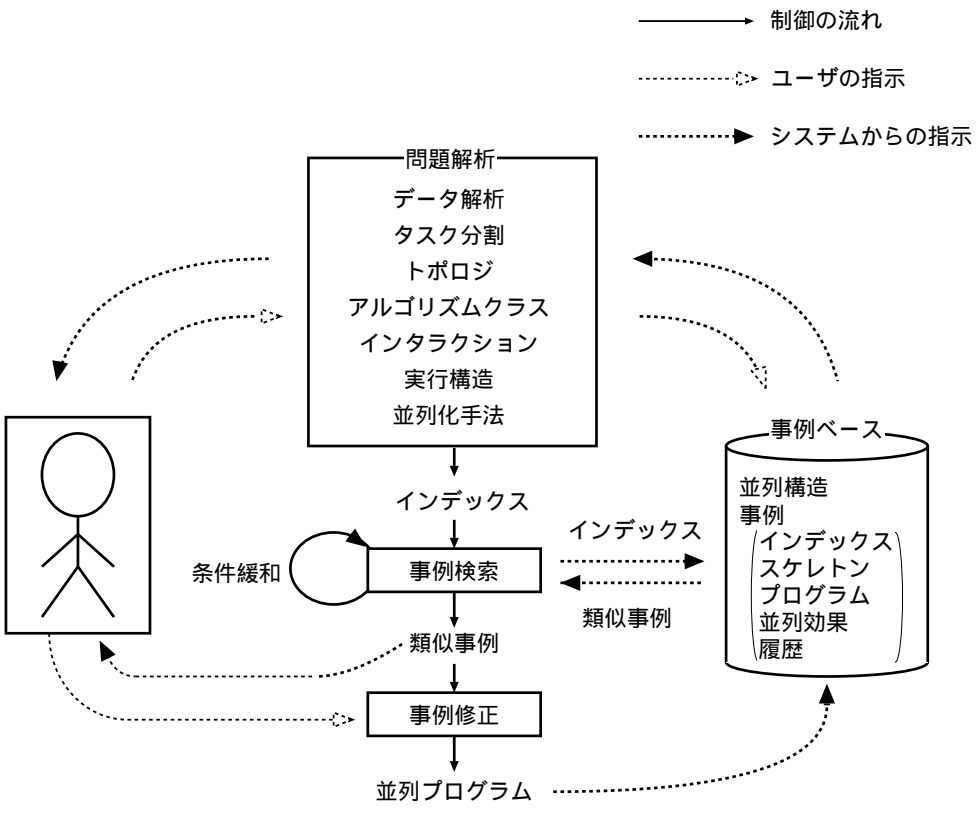


図 12: システム構成

3.2 事例ベースの構築

事例は、問題の定義、インデックス、スケレトン、プログラム、並列効果から構成されている。

1. インデックス

応用	問題の応用分野
仕様	問題の定義
終了条件	計算終了のための条件
アルゴリズム	プロセッサファーム、繰り返し変換
	プロセスネットワーク、分割統治法
データ構造	ソースデータ、結果データ
タスク分割	
インタラクション	シグナルウエイト、ミューテックス、バリア
トポロジ	トリー、ワーカー、マスター＆ワーカー、パイプ、リング
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. スケレトン
   - 並列プログラムの骨格
   - データ構造、相互排除、同期、スレッド
3. プログラム
4. 並列効果

   プロセッサ台数増加に伴う速度向上の実測

4 事例ベース並列プログラミング

4.1 BM 法

1. BM 法による文字列照合
   BM 法とは、２つのシフト方法よる照合である。ヒューリテクス１は, 照合が失敗した位置のテキストの文字 TEXT[K] を情報として使うものであり、ヒューリテクス２は KMP 法と同じように PAT の構造情報を利用するものである. ヒューリテクス１と２は、互いに独立したアイデアなので、より効果の大きいほうを選択し、文字列照合を行う方法である。TEXT、PAT はあらかじめファイルに格納しておく。TEXT １０万文字、PAT ４０個３０文字により計算を行った.

2. 並列化
   TEXT を使用するスレッド数と同数のブロックに分割する。各スレッドは自分に与えられた文字列範囲の中で検索を行う。隣接した 2 つのブロックの境界をはさんで PAT が存在する場合があるので、境界付近の一定領域は重複して 2 つのスレッドに割り振られる。PAT 情報については、全 PAT データのコピーが全スレッドに割り振られる。パラレルリージョンにより各スレッドはマスターからタスクを受取り、独立して計算を行う。そしてマスターに計算結果を返す。図 13のように各プロセッサが見つけた PAT を格納していく。

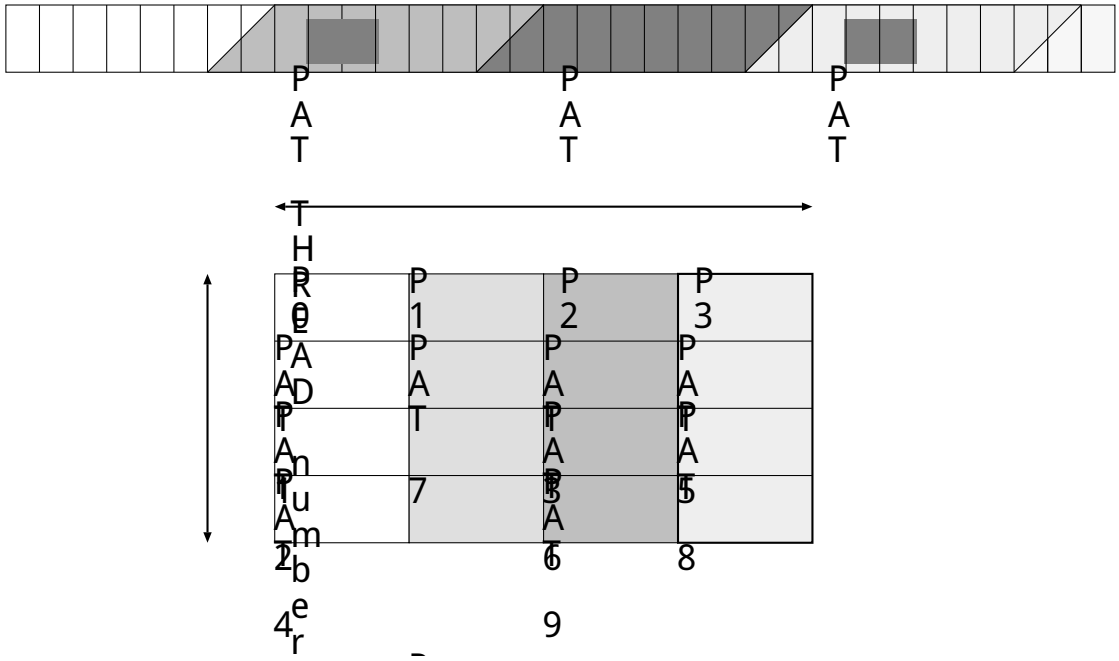


図 13: タスク分割

3. インデックス

表 1: BM 法インデックス

応用	:文字列照合
仕様	:TEXT をスレッド数で分割したサブ TEXT 内で、PAT と一致する文字列を照合、検索を行う。
データ構造	:ソースデータ 文字型 1 次元配列 TEXT 文字型 2 次元配列 PAT 結果データ 2 次元配列 RESULT
タスク分割	:ソースデータ:TEXT 、重複ブロック、ブロック PAT 、 、共有 :結果データ:RESULT 、列、サイクリック
終了条件	:なし
トポロジ	: マスタ&ワー カー
アルゴリズム	:プロセッサファーム
実行構造	:C^t
インタラクション	:なし

図 14のの結果は PAT 数が４０個、TEXT が１０万文字の場合の並列効果である。プロセッサファームにより、均等にタスクが割り振られ、インタラクションがないこともあり、速度向上は得られる。

4. 並列効果

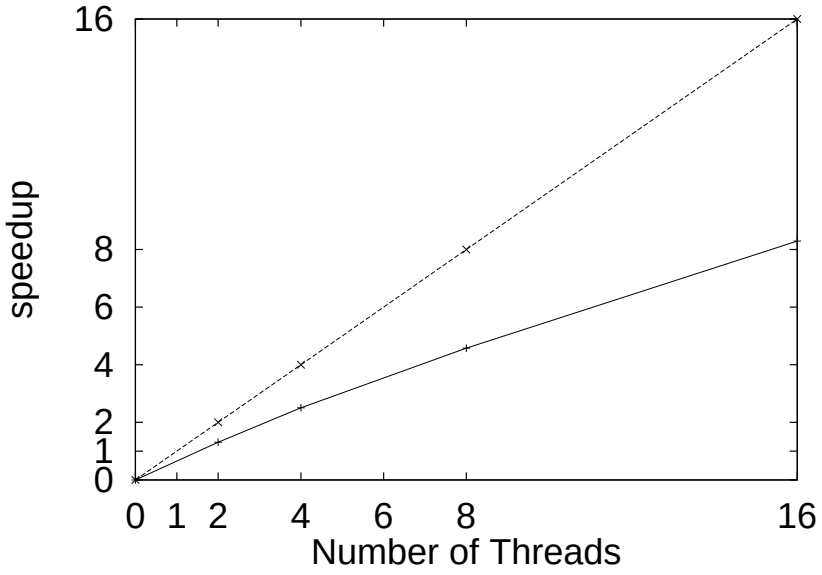


図 14: BM 法並列効果

4.2 ロンバーグ積分法

1. ロンバーグ積分法台形公式から $S_0^0, S_1^0, ..., S_k^0$ を求め $j = 1$ に対して $S_n^j = \frac{2^{2j}S_{n+1}^{j-1} - S_n^{j-1}}{2^{2j}-1}$ で S_n^j を $n = 0, 1, .., k-j$ に対して求める。次にこの計算を次々と j=2,3,.., に対して行う。S_n^jの値は表のように表される。第 1 列は台形公式から得られたもの、第 2 列はシンプソンの公式で得られたものになっている。$|S_0^k - S_0^{k-1}| < eps$ となるまで計算を行わせる。

2. 並列化

(a) 初期値として $S_0^0, S_1^0, ..., S_k^0$ を求め、それをサイクリックによって、プロセッサに割り当てる。

(b) j = 1 に対して S_n^j を $n = 0, 1, .., k - j$ に対して求める. S_0^k と S_0^{k-1} との差がある一定の値より大きければ (c) へ、小さければ (d) へ。

(c) 次々と j=2,3,.., に対して繰り返して行う.

(d) $\mid S_0^k - S_0^{k-1} \mid < eps$ となったとき S_0^kの値を解としプログラムを終了する。

各スレッドは割り振られた S_n^jに対して変換処理を行う。その際、前処理で出た全ての S_n^jは、全スレッドに対してデータ参照が行われるため、S_n^jは共有変数となっている。全スレッドが、バリアにチェックインし、同一の計算により、S_n^jが変換処理される。全スレッドが、バリアからチェックアウトすると、終了条件に持ち込み、判定を行う。条件を満たしていれば終了し、満たしてなければ、一連の動作を繰り返し行う。

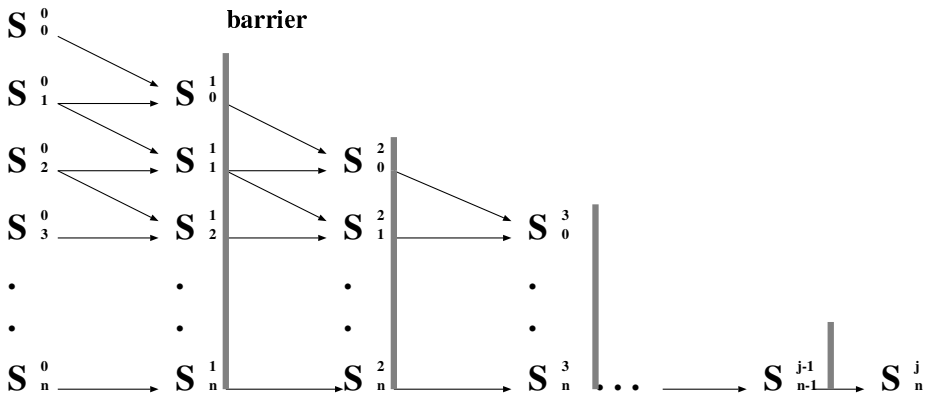


図 15: ロンバーグ積分

3. インデックス

表 2: ロンバーグ積分インデックス

応用	: 数値計算
仕様	:台形公式から $S_0^0, S_1^0, ...S_k^0$ を求め j = 1 に対して $S_n^j = \frac{2^{2j}S_{n+1}^{j-1} - S_n^{j-1}}{2^{2j}-1}$ で S_n^j を $n = 0, 1, .., k-j$ に対して求める。次にこの計算を次々と $j = 2, 3, ..,$ に対して行う。
データ構造	:ソースデータ: 積分区間を示す 2 つの x 座標値 関数 $f(x)$ の式データ :結果データ double 型 RESULT
タスク分割	: ソースデータ: S_n^j 、 1 要素、共有
終了条件	: $eps = 1.0e-8$
トポロジ	: 完全照合
アルゴリズム	: 繰り返し変換
実行構造	:$Cond\{I^t_{Barrier(A/B)}\{C^t\}, C^{p(master)}\}$
並列化手法	: パラレルリ– ジョン
インタラクション	:バリア

4. 並列効果

exp(0.5(1-x)(1-x)) を区間 [0,1000000000] で積分を行った。

初期値として求めた $S_0^0, S_1^0, ..., S_k^0$ をサイクリックにプロセッサに割り振っていくが、図 1 4 の構造から空きスレッドが生じてしまうので、4 台目以降から速度向上は減少していく。

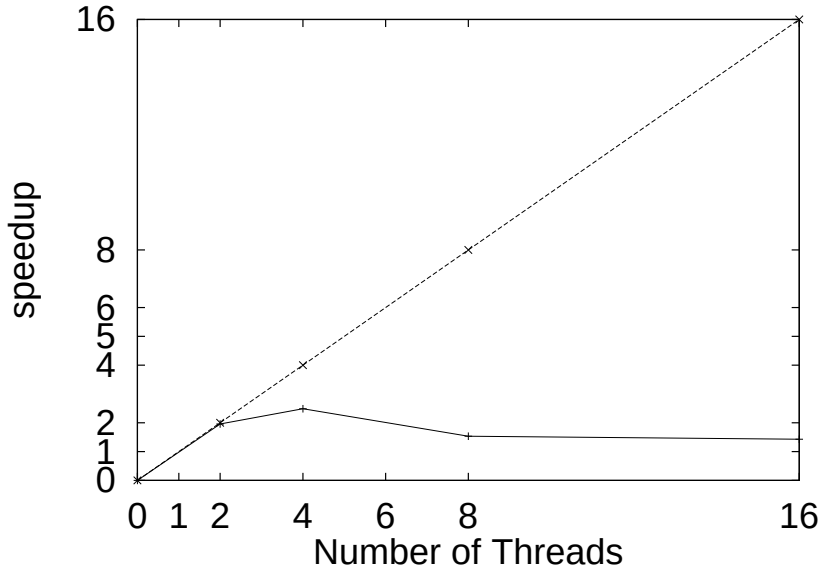


図 16: ロンバーグ積分法の並列効果

4.3 n クイーン

5　事例の再利用度の評価

事例の再利用度の評価として下図を用いて検討する。スケレトンでは、類似事例として用いたプログラムから、 スレッド、 同期、タスク分割について、それぞれ再利用、修正、新規のどれが対応してるかを、プログラムから行数で利用度を示す。逐次プログラムからは、単位計算とアルゴリズムの利用度を示す。また、プログラムを一部省略し、下の記号を使い具体的に利用された部分を示す。空白行は逐次プログラムからの利用である。

	スケレトン				逐次プログラム
	スレッド (T)	同期 (S)	タスク分割 (D)	その他	
再利用 (S)	ST	SS	SD	S	行数
修正 (M)	MT	MS	MD	M	行数
新規 (N)	NT	NS	ND	N	行数

5.1　KMP 法を類似事例とした再利用度評価

```
S :各種　include ファイル
S :main(){
S : printf("input threads="); scanf("%d",&threads);
   ソースファイル読み込み　　BM 法計算の初期設定

ST :myteam=pr_create_team(threads);
S :   as = all_seconds();  us = user_seconds();
SD : pr_pcall(myteam,work,copyargs(threads));
S : printf("CPU time = %f\n",user_seconds()-us);
S : printf("Elapssed time = %f\n",all_seconds()-as);
S : for(j=0;j<threads;j++){
S:    for(i=0;i<ad[j];i++) {
S:  printf("moji[%d][%d]=%d\n",i,j,moji[i][j]);}
      }}
S: void work(void){
 ST : no=pr_mid();
 SD: t=n/threads;
 ND: x=t*no;
 S:for(i=0;i<PAT;i++){
 S:    m=strlen(pat[i])-1;
 ND:   if(no!=threads-1)y=t*(no+1)+m-1;
 ND: else   y=t*(no+1)-1;
 ND:   k=x+m;
```

```
 ND:   while(k>0 && x<k && k<=y){
         単位計算
          if (j==0) {
S :          moji[ad[no]][no]=k+1; {結果回収}
S :          ban[ad[no]][no]=i+1;
    break;}
      else{単位計算}
   } }}
```

表 3: KMP 法再利用度

	スケレトン				逐次プログラム
	スレッド (T)	同期 (S)	タスク分割 (D)	その他	
再利用 (S)	6	1	0	1 7	6 0
修正 (M)	0	0	0	0	0
新規 (N)	1	0	6	0	0

5.1.1 評価と考察

応用が同類、ソースデータが同じで、並列アルゴリズムは 2 つともプロセッサファームを用いてる為、スレッド、同期は再利用できた。KMP 法との相違点は、KMP 法が PAT の前から照合していくのに対して、BM 法は PAT の後ろから照合していくので、タスク分割に対しては、新規作成が必要となった。また、応用分野が同類なので、結果の出力など細部にわたり、プログラム作成の際、参考となった。逐次プログラムに関しては、単位計算が全く違うので新規作成が必要となった。

5.2 DKA 法を類似事例とした再利用度評価

```
S :各種  include ファイル
   方程式のデータ入力     W :結果出力    start(); {単位計算 }
S :void main(){
ST :printf("input threads \n");scanf("%d",&threads);
ST :myteam  = pr_create_team(thread
SS :  pthread_barrier_init(&barrierA,pthread_barrierattr_default,threads);
SS :  pthread_barrier_init(&barrierB,pthread_barrierattr_default,threads);
      start();
```

```
S : as = all_seconds();us = user_seconds();
SD :  pr_pcall( myteam, worker, copyargs(threads) );
S :  printf(" CPU time = %f \n",user_seconds()-us);
S :  printf(" Elapsed time = %f \n",all_seconds()-as);
    printf("W=%14.10e\n",W);
}
S :void worker()  {
ST :  myno=pr_mid();
S:   while(1){
SS :     pthread_barrier_checkin(&barrierA,myno);
ND:          object(myno,count,i4);
SS :     pthread_barrier_checkout(&barrierA,myno);
N :    if(fabs((R[0][count+1]-R[1][count])/R[0][count+1])>eps)
         {count++;i4*=4; }
N:          else{ W=R[0][count+1]; break;}
N:void object(int myno,int count,int i4) {
N:  int t;
ND: for(t=myno;t<nmax-count-1;t=t+threads){
N:      R[t][count+1]=(i4*R[t+1][count]-R[t][count])/(i4-1);}
    }
```

表 4: DKA 再利用度

	スケレトン				逐次プログラム
	スレッド (T)	同期 (S)	タスク分割 (D)	その他	
再利用 (S)	7	8	0	9	2 0
修正 (M)	0	0	0	0	0
新規 (N)	2	0	1	1 1	0

5.2.1 評価と考察

並列アルゴリズムを、2 つとも繰り返し変換を用いているので、DKA 法からは、スレッド、同期を再利用できたが、タスク分割に関しては、応用分野が、数値積分の事例がないということと、仕様も類似したものがないため、新規作成が必要となった。単位計算は逐次プログラムから再利用した。

6　おわりに

並列プログラミングを作成する際、類似した事例を用いて、繰り返し変換、プロセッサファーム、プロセスネットワーク、分割統治法の４つの並列アルゴリズムにより、事例を作成する。並列アルゴリズム毎に類似事例を参照することによって、並列プログラム作成の負担を軽減させることが出来る。

しかし、逐次プログラムから並列プログラムへの変換の際、並列アルゴリズムとインタラクションの組合せにより実現する。

プロセッサファームは、比較的容易に逐次を並列化することが出来るが、タスク分割方法は、対象とするデータ構造の違いにより再利用することはできず、新規作成が必要となる。プロセスネットワーク、繰り返し変換では、問題の応用や仕様が違うと再利用度は低くなる。

類似事例を最大に再利用するには、問題の応用や仕様の類似点によるところが大きい。BM 法と KMP 法では、応用分野において同類だが、仕様の相違点からタスク分割のところで新規作成が必要となる。繰り返し変換と DKA 法では、問題、応用ともまったく違うのでタスク分割のみならず新規作成を要する。

今後の課題としては、事例を増やし、タスク分割の負担をいかに軽減させるかである。軽減させることによって、並列プログラミングはさらに容易となるであろう。

謝辞

本研究の機会を与えて下さり, 日頃, 貴重な御指導を頂きました山崎勝弘教授に心より深く感謝の意を表します. また, 日頃より貴重御意見を頂きました本研究室の皆様に心より感謝致します . さらに、様々な質問に答えてくださった安藤　彰一氏に深く感謝致します。

そして, KSR-１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. 株式会社に感謝致します.

参考文献

[1] 松原　仁,"推論技術の観点から見た事例に基ずく推論", 人工知能学会誌,Vol.7,No.4,pp.567-575,Jan.1992.

[2] 河野　毅, 末田　直道, 田中　利一,"CBS　システムの構築環境", 人工知能学会誌,Vol.9,No.1,pp.91-99,jan.1994.

[3] 小林　康弘,"事例ベース推論", 人工知能学会誌,Vol.8,No.1,pp.26-36,Jan.1993

[4] Burkhart H. et al., "BACS:Basel Algorithm Classification Scheme",Technikal Report 93-3,Universitat basel,1993.

[5] Rabbi,F.a.,"Exploiting Parallelism in Functional Languages: a Paradigm-Oriented Approach",Second Workshop on Asstrqact Machine Models for Highly Parallel Computers,pp.1-14,April 1993.

[6] Canon Supercomputing S.I.Inc."C ユーザーのための KSR 仕様手引書", Nov.1994.

[7] 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", 並列処理シンポジウム JSPP'97,pp.117-124,1997.

[8] 数値解析の基礎　川崎　晴久　共立出版 1993

[9] データ構造　西原　清一　オーム社