計算機アーキテクチャ　118－1
（1996. 5. 16）

並列プログラミング用事例ベースの構築

山崎 勝弘　　松田 浩一　　安藤 彰一
立命館大学理工学部情報学科
〒525－77 草津市野路町1916
{yamazaki, matsuda, ando}@hpc.cs.ritsumei.ac.jp

あらまし　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方法について述べる。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、繰り返し変換に分類される。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。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、それを自動／手動で修正して並列プログラムを生成する。

キーワード　並列プログラミング、並列アルゴリズム、事例ベース推論、スケレトン、事例検索・修正

A Case-Base for Parallel Programming

Katsuhiro Yamazaki, Koichi Matsuda and Shoichi Ando
Department of Computer Science, Ritsumeikan University
Nojicho, Kusatsu, 525-77 Japan
{yamazaki, matsuda, ando}@hpc.cs.ritsumei.ac.jp

Abstract　This paper describes how to reduce the burden of parallel programming by utilizing similar parallel programs as much as possible. This research first develops a case-base for parallel programming by making out typical parallel programs for four parallel algorithm classes. Cases consist of indices, a skeleton, a program, parallelization effects and a history. Skeletons include the most important issues of parallel programs such as task division, synchronization, mutual exclusion and parallelization methods. Indices illustrate the features of parallel programs and parallelization effects show speedups. The system retrieves the most relevant case from the case-base, and adapts it to a given problem automatically or manually so that the final parallel program is developed.

key words　Parallel Programming, Parallel Algorithm, Case-Based Reasoning, Skeleton, Case Retrieval and Adaptation

1 はじめに

気象予測、環境問題などの大規模な問題を超／高並列マシンを用いて解く高性能コンピューティングの研究がアメリカのHPCCプロジェクトを始め、世界各国で進められている。並列処理は大規模計算に必須のものであるが、並列マシンは依然として一部の専門家のみに使用され、十分に普及しているとは言いがたい。これは自動並列化コンパイラ、並列プログラミング言語が未だ研究途上であり、並列プログラミングが依然として極めて困難であるからである。

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 [1] 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。

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、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、Fortran/C 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。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レトンとして用意する。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最も重要な部分が含まれる。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けする。インデックスには応用分野、仕様、アルゴリズム、終了条件、データ構造、タスク分割、同期、並列化手法、実行構造が含まれる。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、スケレトン、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。

本システムでは、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に分類する。それらはKSR 1上で、パラレルリージョン、スレッド、バリア同期などにより実現される。また、アルゴリズムの特徴付けを行うために、バーゼル大学で開発されたBACS (Basel Algorithm Classification Scheme) [2] を用いる。

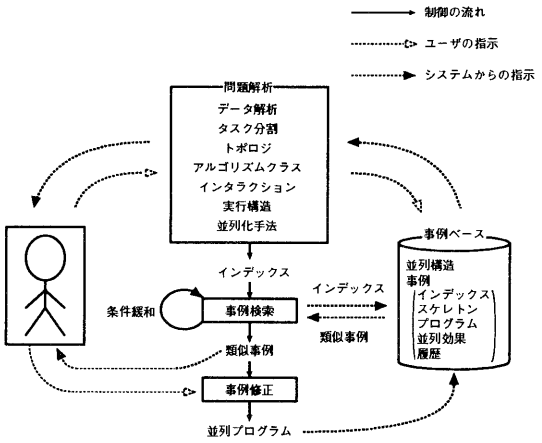


図 1: システム構成

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。システムはBACS表現、並列化手法、データ構造、インタラクションなどをインデックスとして、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、スケレトン、プログラム、並列効果がユーザに提示される。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。

スケレトン内には、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スレッド関連はスケレトン内の情報を再利用できる。従って、初期化、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。検索事例が与えられた問題と酷似している場合には、自動修正を試みる。

事例ベースは過去の代表的な並列プログラムと、代表的な並列化手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

Rabhi は並列アルゴリズムを図 2に示すように四つのパラダイムに分類しており [3]、これが最も一般的な分類である。図で丸はプロセッサ、四角はタスクを示す。

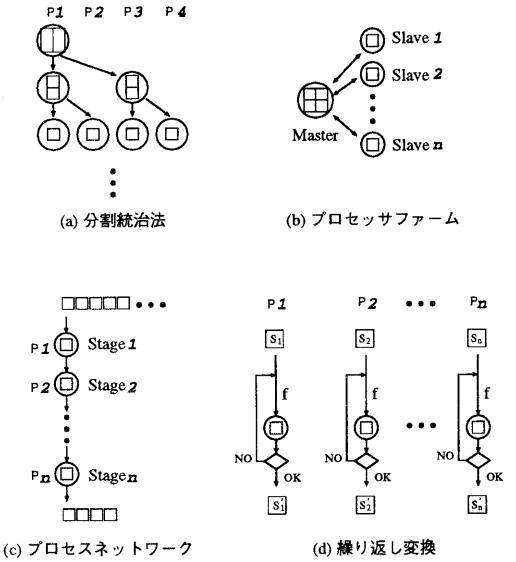


図 2: 並列アルゴリズムの一般的分類

- 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。

BACS[2] 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、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。本システムでは、インデックス内のタスク分割と実行構造に BACS 表記を用いており、類似事例の検索時にこれらを用いる。

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 [4]。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 SIMD、後者は MIMD 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。

4 事例ベースの構築

現在までに、１０個の事例が作成されている [5][6][7]。以下、図 2に示す各アルゴリズムクラス毎に事例を一つずつ示す。

4.1 クイックソート

4.1.1 問題の定義

n 個のデータを昇順にソートする。基準値を一つ選び、それより大きいグループと小さいグループに分割する。これを再帰的に適用する。

4.1.2 インデックス

応用	ソーティング
仕様	要素列 ($data[L] \sim data[R]$) を, $data[\frac{L+R}{2}]$ 未満の要素からなる部分要素列と, $data[\frac{L+R}{2}]$ 以上の要素からなる部分要素列の 2 つに分割.
アルゴリズム	分割統治法
終了条件	なし
データ構造	
ソース	整数型 1 次元配列 $data[L] \sim data[R]$
結果	整数型 1 次元配列 $data[L] \sim data[R]$
タスク分割	$data$, 要素ブロック, ブロック
トポロジ	トリー
インタラクション	ミューテックス
並列化手法	パラレルリージョン
実行構造	$Cond\{I^p_{wait}, Cond(C^p, I^p_{search})\}$

4.1.3 スケレトン

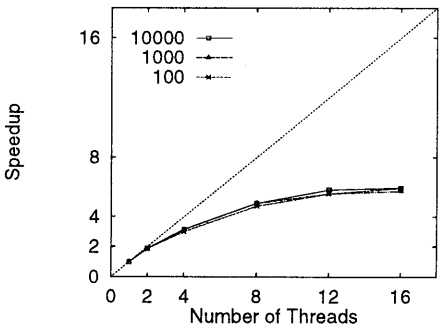
main でチーム ID を取得し、パラレルリージョンを起動する。thread_quick では他スレッドからの起動信号を待ち、起動されれば quick を呼び出す。終了後、自分をアイドルにする。quick では二つの部分列に分割した後、大きい方の再分割を行い、アイドルスレッドを探して起動する。CALCULATION では仕様で規定された計算を行う。

```
main
{ input the number of threads;
  input data; initialize a mutex;
  get a team ID ;     no. of active threads = 1;
  activate a parallel region( team ID, thread_quick );
  print the results;
}
thread_quick
{ get a thread no.;
  while(1)
  { wait an activate signal.;
    if ( no. of active threads == 0 ) break;
```

```
      quick( start of subsequence:L, end of subsequence:R );
      mutex lock;
      no. of active threads--;          inactivate myself;
      mutex unlock;
} }
quick( L, R )
{ CALCULATION;
  for a larger subsequence
  { if ( amount of data > 1% of the total amount of data )
      { mutex lock;
        if ( no. of active threads < no. of all threads )
          { no. of active threads++;
            search an idle thread;
            task assignment(start is i and end is R );
            activate the searched thread;
            mutex unlock;
          }
        else { mutex unlock;     quick(i, R); }
      }
    else quick(i,R);
  }
  for a smaller subsequence   quick(L, j);
}
```

4.1.4 並列効果

データ数を増加させても速度向上は高々6以下で、それ以上改善されない。mutex による相互排除のためであると考えられる。



4.2 マンデルブロ

4.2.1 問題の定義

複素関数 $f(Z): Z_{i+1} = Z_i^2 + C$ に対して, 初期値を $Z_0 = (0,0)$ と置き, $Z_1 = f(Z_0)$, $Z_2 = f(Z_1)$, $\cdots$, $Z_k = f(Z_{k-1})$, のように反復計算を繰り返す. 複素平面上の座標値 C の値を変化させ, $Z_k (k \to \infty)$ の値が収束か, 発散かを求める. $(-2.0 \leq$ 実部 $< 0.5)$, $(-1.25 \leq$ 虚部 $< 1.25)$ の範囲の複素平面を $X \times Y$ のメッシュに区切り, $X \times Y$ 個の点について上の反復計算を行う. その結果, $|Z_i| > 2.0$ のとき発散, $i > 200$ のとき収束するとし, 後者の結果となる座標点がマンデルブロ集合の要素となる.

4.2.2 インデックス

応用	数値計算
仕様	$Z_{i+1} = Z_i^2 + C \quad (i = 0, 1, 2, ...)$
アルゴリズム	プロセッサファーム
終了条件	なし
データ構造	
ソース	複素平面 $Z_i(x, yj)$
結果	整数型2次元配列 $data[x][y]$
タスク分割	
$Z_i(x, yj)$,	列方向 , 巡回
$data[X][Y]$,	列方向 , 巡回
トポロジ	マスター・ワーカ
インタラクション	なし
並列化手法	パラレルリージョン
BACS	C^t

4.2.3 スケレトン

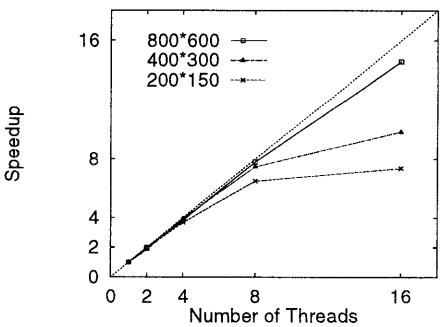
main でパラレルリージョンを起動する。work ではスレッド番号を取得し、自分が割り当てられた列を計算する。

```
main
{ input the number of threads;
  get a team ID;
  activate a parallel region( team ID, work );
  calculate the Mandelbrot set using data[X][Y];
  print the results;
}

work
{ get a thread no.;
  for ( I = min of X + thread no.; I < max of X;
  I += no. of threads )
    { for ( J = min of Y; J < max of Y; J++ )
        { CALCULATION;
          store the results into data[I][J];
        }
    }
}
```

4.2.4 並列効果

分割数が大きいとき、ほぼリニアな速度向上が得られる。



4.3 バブルソート

4.3.1 問題の定義

n 個のデータを昇順にソートする。配列の下位のデータから始めて、隣接する二つのデータを順次比較・交換することにより、最小値（バブル）を一番上へ移動させる。ソート範囲を一つ狭めて、これを繰り返す。

4.3.2 インデックス

応用	ソーティング
仕様	バブル (局所最小データ) が上がってくるのを待ち, 自分の範囲内で新たなバブルを順番に検索
アルゴリズム	プロセスネットワーク
終了条件	なし
データ構造	
ソース	整数型 1 次元配列 $data[\,]$
結果	$data[\,]$
タスク分割	
$data[\,]$	要素ブロック　, ブロック
トポロジ	パイプ
インタラクション	シグナル・ウエイト
並列化手法	スレッドライブラリ
実行構造	
top_thread	$Fix\{C, I^{p(lower\_thread)}\}$
inter_thread	$Fix\{I^{p(upper\_thread)}, C, I^{p(lower\_thread)}\}$
tail_thread	$Fix\{I^{p(upper\_thread)}, C\}$

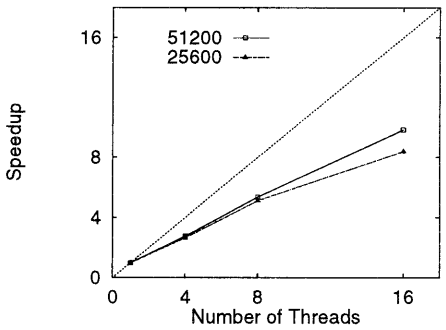
4.3.3 スケレトン

main で inter スレッド、tail スレッドを生成し、top スレッドを呼び出す。top スレッドでは上端、下端を設定した後、バブルを下から上へ流す。上端を調整して、lower スレッドと同期をとる。inter スレッドでは upper スレッドと同期をとり、境界値を比較する。upper スレッドと同期をとり、バブルを上へ流す。lower スレッドと同期をとる。

```
main
{ input data;   input the number of threads;
   initialize flag variables;
   for ( no. of threads − 2 )
   create threads( inter_thread, thread no. )
   create a thread( tail_thread, thread no. )
   top_thread( 0 );
   print the results;
}
top_thread( thread no. )
{ set the upper and lower edges;
   while ( upper edge ≤ lower edge )
   { while ( lower edge ~ upper edge )
        move smaller values to the upper;
     adjust the upper edge;
     synchronize with a lower thread;
   }
}
inter_thread( thread no. )
{ set the upper and lower edges;
   while ( 1 )
   { synchronize with an upper thread;
     compare the lowest value with the highest value
     of the upper thread;
     synchronize with an upper thread;
     while ( lower edge -1 ~ upper edge )
        move smaller values to the upper;
     adjust the upper and lower edges;
     synchronize with a lower thread;
   }
}
```

4.3.4 並列効果

データ数が 51200 個の場合、かなり良い速度向上が得られる。



4.4 高次方程式の解法（DKA 法）

4.4.1 問題の定義

実根を持つ n 次の多項式

$$f(x) = x^n + \alpha_1 x^{n-1} + \cdots + \alpha_{n-1}x + \alpha_n$$

の n 個の解 $X_1, X_2, \cdots, X_n$ を同時に求める。アルゴリズムは反復法で, X_i の k 回目の計算の近似値を $X_i^{(k)}(i = 1, 2, \cdots, n)$ とし,

$$X_i^{(k+1)} = X_i^{(k)} - \frac{f(X_i^{(k)})}{\prod_{j=1, j\neq i}^{n} (X_i^{(k)} - X_j^{(k)})}$$

により逐次近似値を求める. これを, $X_i^{(k+1)}$ と $X_i^{(k)}$ との差が 10^{-6} より小さくなるまで続ける.

4.4.2 インデックス

応用	数値計算
仕様	

$$X_i^{(k+1)} = X_i^{(k)} - \frac{f(X_i^{(k)})}{\prod_{j=1, j\neq i}^{n} (X_i^{(k)} - X_j^{(k)})}$$

アルゴリズム	繰り返し変換
終了条件	$\|X_i^{(k+1)} - X_i^{(k)}\| < 10^{-6}$ 又は ステップ数 = 5000
データ構造	
ソース	実数型 1 次元配列 $X[n]$: 解 実数型 1 次元配列 $a[n]$: 係数
結果	$X[n]$
タスク分割	
ソース	$X[n]$, , コピー $a[n]$, , コピー
結果	$X[n]$, 要素 , 巡回
トポロジ	完全結合
インタラクション	バリヤ
並列化手法	パラレルリージョン
実行構造	$Cond\{I^t_{barrier(A/B)}\{C^t\}, C^{p(master)}\}$

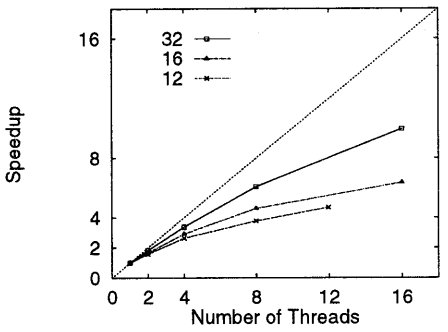
4.4.3 スケレトン

main でパラレルリージョンを起動する。worker ではスレッド番号を取得し、Xi の初期値を計算する。バリア A でチェックインし、object を呼び出して、単位計算を行う。チェックアウトしてマスターが終了条件を調べる。次に、バリア B を用いて同様のことを行う。

```
main
{ input the number of threads;      input coefficients;
   initialize barriers( A & B );
   get a team ID ;
   activate a parallel region( team ID, worker );
   print the results;
}
worker
{ get a thread no; step no. = 0;
   calculate an initial value of Xi ;
   while(1)
   {  step no.++;
      barrier check-in( &barrierA, thread no. );
         object( thread no. );
      barrier check-out( &barrierA, thread no. );
      if ( ( flag == 0 ) && ( step no. < max step no. ) )
         the master checks the termination conditions
      else  break;
      step no.++;
      barrier check-in ( &barrierB, thread no. );
         object( thread no. );
      barrier check-out( &barrierB, thread no. );
      if ( ( flag == 0 ) && ( thread no. < max step no. ) )
         the master checks the termination conditions
      else  break;
   }
}
object ( thread no. )
{ for ( i = thread no.; i < n; i =+ no. of threads )
      CALCULATION;
   judge if the result satisfies the termination condition;
}
```

4.4.4 並列効果

次数が大きくなるにつれ、良い速度向上が得られる。



5 おわりに

本論文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、新しい問題に対して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。並列アルゴリズムを四つのクラスに分類し、各クラス毎に事例を示した。今後、新たな問題に対する類似事例の検索・修正法を詳細に検討し、システムを構築していく予定である。

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。

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

[3] 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.

[4] Kendall Square Research, "KSR Parallel Programming", 1991.

[5] 山崎 勝弘, 古川 知之, "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", 信学技報 *CPSY94-85*, pp.9–16, 1994.

[6] 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.

[7] 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.