並列性忘却プログラミングとその研究推進に向けて

松崎　公紀

（受領日：2010 年 4 月 28 日）

高知工科大学情報学群
〒 782-8502 高知県香美市土佐山田町宮ノ口 185

E-mail: matsuzaki.kiminori@kochi-tech.ac.jp

要約：近年、マルチコア CPU などの並列化されたハードウェアが広く普及してきている。しかしながら、そのような環境を効率的に利用するための並列プログラミングは、多くの場合既存の逐次プログラミングの延長によって行われ、データ分配やプロセッサ間通信などより難しい作業となっている。著者らは、並列性を意識することなく並列プログラムを作成するという「並列性忘却プログラミング」の視点で研究を行っている。対象となる問題の性質や表現法に含まれる並列性をうまく利用して、並列プログラムを容易にかつ正しく作成できるようにすることを目的としている。本稿では、並列性忘却プログラミングの基本的考え方と、その具体的研究である「スケルトン並列プログラミング」について説明する。また、並列性忘却プログラミングの研究推進に向けての共同研究体制についても紹介する。

1. はじめに

近年、計算機ハードウェアは並列性によって全体の処理能力を向上させることで進化してきている。例えば、現在、一般向けを含む多くの計算機では、複数のプロセッサコアを持つマルチコア CPU が利用されており、ひとつの CPU に含まれるプロセッサコアの数も増加している。また、数十個以上のコアを含むようなメニーコア CPU や、画像処理用に特化して高い性能を持つ GPU を汎用的な目的に利用する GPGPU（General Purpose Computing on GPU）など、並列計算のためのハードウェアは複雑化している。

一方で、並列化したハードウェアを効率的に利用するソフトウェアの開発、すなわち、並列プログラミングは、既存の逐次プログラミングの延長として行われているのが現状である。例えば、並列プログラムを記述する上で現在広く利用されている MPI では、ユーザは明示的にプロセッサ間の通信や同期などを記述する必要がある。また、メニーコア CPU や GPGPU では、逐次プログラミングを拡張した個別のプログラミングモデルが採用されている。そのため、並列プログラミングは逐次プログラミングに比べて複雑で難しい作業となっている。

この問題を解決するための手法のひとつに、Cole によって提案されたスケルトン並列プログラミング（Skeletal Parallel Programming）[1,2] がある。スケルトン並列プログラミングとは、並列計算においてよく利用される抽象化された計算パターン（並列スケルトン）を組み合わせて並列プログラムを作成する手法である。データの分割やプロセッサ間の通信などの並列処理における複雑さが並列スケルトンに隠蔽されることで、ユーザは並列性についてあまり意識することなく並列プログラムを作成することができる。また、並列スケルトンが抽象化された計算パターンであることにより、それらによって作成された並列プログラムの移植性が高く、プログラム運算手法や形式的検証手法を適用することができるという利点もある。

著者らはこれまで、スケルトン並列プログラミングに関する研究を進めてきた。これまでに、さまざまな並列データ構造に対する並列スケルトンの定義や、並列スケルトンを用いたプログラムの導出手法、並列スケルトンのライブラリ実装などを提案してきた。さらに、これまでの研究で得られた知見を基礎として、「並列性忘却プログラミング（Parallelism-Oblivious Programming）」[3] という新しい考え方に向けた研究を行っている。並列

性忘却プログラミングとは、簡単に言うと、対象となる問題の性質や表現法に含まれる並列性をうまく利用することにより、プログラムを記述する際に並列性を意識しないで済む新しい並列プログラミングのスタイルである。

本稿では、並列性忘却プログラミングの考え方について議論し、またその研究推進に向けた高知工科大学を含む共同研究体制について紹介する。前者について具体的には、並列性忘却プログラミングの基本となるアイデアに加えて、そのアイデアの基礎となったスケルトン並列プログラミング、および、著者らが開発している並列スケルトンライブラリ「SkeTo」について述べる。本稿の構成は次の通りである。第 2 章では、並列性忘却プログラミングの基本的なアイデアと具体例を示す。第 3 章では、スケルトン並列プログラミングについて概観する。第 4 章では、著者らが開発している並列スケルトンライブラリ「助っ人」について述べる。第 5 章で研究推進に向けた共同研究体制について紹介し、最後に第 6 章でまとめる。

2. 並列性忘却プログラミング

並列性忘却プログラミング（Parallelism-Oblivious Programming）[3] は東京大学の武市正人教授による標語である。本章では、並列性忘却プログラミングが目指す目標について説明し、その具体的なアイデアをふたつ紹介する。

これまで、並列プログラミングは多くの場合で逐次プログラムに並列処理を追加することによって行われてきた。例えば、共有メモリ環境を対象として簡単に並列プログラムを作成できる手法のひとつである OpenMP を利用したプログラミングでは、

```
#pragma omp parallel for reduction (+:sum)
for (i = 0; i < n; i++) {
  sum += a[i] * a[i];
}
```

というように、逐次プログラム中の並列化したい部分に対して指示文（アノテーション）を付加するというスタイルをとる。また、分散メモリ環境を対象として標準的に利用されている MPI では、プロセッサごとの処理を意識し、プロセッサ間の通信や同期を明示的に記述することが必要である。

並列性忘却プログラミングは、これらの手法とは別の視点から並列プログラミングを追究しようというものである。対象となる問題の性質やその自然な表記法に含まれる並列性を積極的に利用することで、並列性を分かりやすく記述できるようにすることが目的である。これにより、並列プログラムにおける並列性を適切に構造化することができ、今後さらに発展するであろうさまざまな並列計算機環境に対応することを狙う。このようなアイデアは、Dijkstra らによって 1970 年代に活発に議論された、（逐次）プログラミングにおける構造化の重要性と類似している。

並列性忘却プログラミングの具体的なアイデアについて、以下でそのふたつを紹介する。また、次章で述べるスケルトン並列プログラミングも並列性忘却プログラミングの具体的な取り組みのひとつと捉えることができる。

ひとつ目は、内包表記の並列プログラミングへの利用である。内包表記は数学の集合操作においてよく利用される表記法で、例えば集合 A の要素の値の二乗和を表現するのに

$$\sum \left\{ a^2 \,\middle|\, a \in A \right\}$$

と表現する表記法である。ここで重要なことは、上の表現の中では二乗の計算やその合計を求める順序について特に規定していないことである。この意味で、この表記法は並列性を自然に持っていると言うこともできる。これに対して、同じ計算を例えば C 言語で記述する場合、一般的には集合を配列で表現して、

```
sqsum = 0;
for (i = 0; i < size_A; i++) {
  sqsum += (A[i] * A[i]);
}
```

と処理の順序を定めて逐次的に記述する。しかし、これはこれまでの計算機の実行モデルからくる制約である。このように処理に順序を定めた逐次プログラムを並列化するのではなく、直接上記の内包表記を並列化するほうが適切であると考える。内包表記の並列プログラミングへの利用は、Fortress [4] や NESL [5] などの言語で採用されている。例えば Fortress 言語では、リスト A の要素の二乗和を計算するのに、

```
sqsum = SUM [a <- A] a^2
```

と記述することができ、またこのプログラムは適切に並列に実行される。内包表記やその他の並列性を持った表現をうまく利用することで並列プロ

グラムを簡潔にまた汎用的に記述できるかを追究することは、並列性忘却プログラミングのひとつの研究テーマである。

ふたつ目は、対象となる問題の性質を利用するものである。ここでは例として探索問題（N クイーン配置問題やゲーム木の探索など）をその対象として考える。探索問題を解くアルゴリズムはこれまでにも多く研究がなされている。例えば、初歩的な探索手法にも、深さ優先探索、幅優先探索、A* アルゴリズムなどがあるが、これらのアルゴリズムは解の探索の順序が異なるため性能が異なる。一般に、探索問題を解く際には、これらの複数のアルゴリズムが適用できる、言い換えると、解を探索する順序に複数の方法がある場合が多い。そのような問題は、解の探索順序に自由度があり、解の探索順序の自由度の中には並列性も含まれていると考えることができる。関数プログラミングの研究分野において、類似した考え方に Improving Value [6] に関する研究がある。Improving Value と呼ばれる特殊なデータ構造と遅延評価機構を利用することで、探索問題の仕様を簡潔に記述することができるのと同時に並列探索アルゴリズムを含む複数の探索が可能となる。著者らは、この Improving Value による探索問題の記述と実行について、遅延評価機構を持たないプログラミング言語を利用しても同様の記述・実行ができることを示した [7]。

3. スケルトン並列プログラミング

スケルトン並列プログラミング（Skeletal Parallelism）は、Cole によって提唱された並列プログラミングの手法である [1,2]。スケルトン並列プログラミングでは、並列計算によく現れる計算パターンを抽象化した並列スケルトンを組み合わせることで並列プログラムを作成する。資源の分配・プロセッサ間の通信や同期などの並列計算のための詳細が並列スケルトンの中に隠蔽されるため、ユーザは並列性を深く考慮することなく並列プログラムを作成することができる。

代表的な並列スケルトンを図 1 に示す。並列スケルトンは大きく、データ並列計算のパターンであるデータ並列スケルトンとタスク並列計算のパターンであるタスク並列スケルトンに分類できる。データ並列スケルトンは、多数のデータに対して同様の処理を行うものであり、代表的なデータ並列スケルトンには map や reduce や scan などがある。タスク並列スケルトンは、パイプラインのための pipe、複数の処理を同時に行う farm など、比較的粒度の大きい処理（タスク）の組み合わせ方を表現するものである。

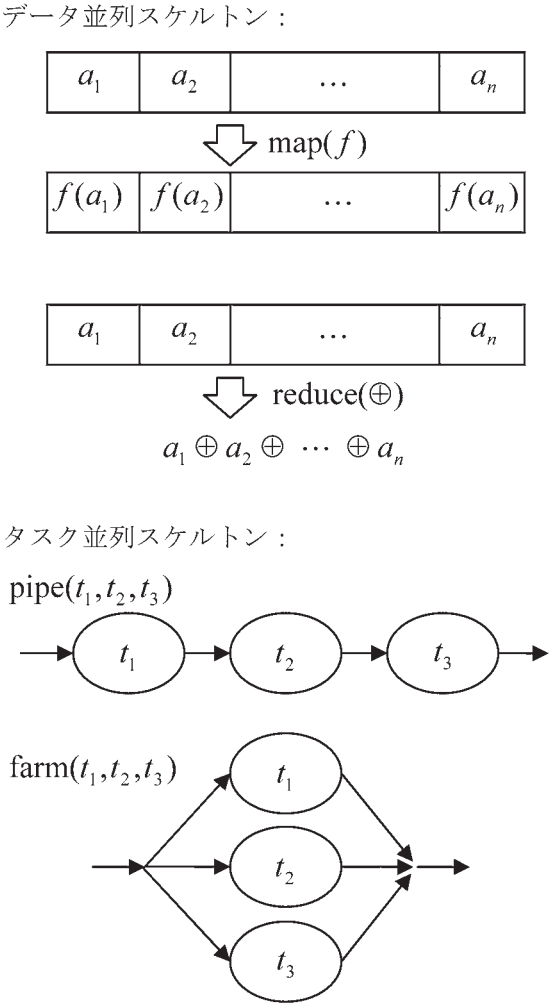

図 1　代表的な並列スケルトン

並列スケルトンプログラミングの利点のうち重要なものは以下の 5 つの性質である。

- 簡潔性：並列性が並列スケルトンの中に隠蔽されるため、ユーザは逐次プログラムのように並列プログラムを作成することができる。
- 効率性：効率的に実装された並列スケルトンのライブラリを用いることで、計算機や低レベルの並列計算ライブラリについて詳しく知らなくても効率の良い並列プログラムを作成できる。
- 移植性：並列スケルトンは多くの計算機環境で効率良く実現できる計算パターンであり、そのためスケルトン並列プログラムはそのまま多くの環境で実行可能となる。
- 性能予測性：並列スケルトンの一緒に提供

されるコストモデルを利用することで、並列スケルトンのプログラムの計算コストを見積もることができ、またコストモデルに基づく最適化も可能となる [8]。

- プログラム導出：計算パターンを抽象的に扱う理論を用いることで、逐次プログラムから系統的に並列スケルトンによるプログラムを導出することができる [9]。

スケルトン並列プログラミングに関して、これまでに多くの研究がなされ、またその中で並列スケルトンをライブラリや言語として提供する実装も多く存在している。並列スケルトンとして計算パターンを抽象化するというアイデアが関数プログラミングと親和性が高いため、Haskell による Eden [10] や ML を利用したもの [11] のように関数型言語を用いた並列スケルトンの実装がある。また、NESL [5] や SaC（Single Assignment C）[12] のように、新しい言語とそのコンパイラを実装する研究もある。一方で、特別な言語を設計することなく、一般のプログラマにとって親しみやすい手続き型プログラミング言語によるライブラリとして提供する取り組みも多く行われている。そのような並列スケルトンライブラリの例として、eSkel [13]（C）、Muesli [14]、ASSIST [15]、SkeTo [16]（C++）、Muskel [17]、Skandium [18]（Java）などがある。また、Google の MapReduce [19]、Intel の Thread Building Blocks [20] などでは並列スケルトンプログラミングのアイデアが利用されている。

4. 並列スケルトンライブラリ SkeTo

それまでのスケルトン並列プログラミングに関する知見に加えて、構成的アルゴリズム論（Constructive Algorithmics）と呼ばれる理論に基づく並列スケルトンライブラリを実現するため、著者らのグループでは並列スケルトンライブラリ「SkeTo」を開発してきた。この並列スケルトンライブラリの名前には、「助っ人」および「Skeletons in Tokyo」という意味が込められている。本章では、並列スケルトンライブラリ SkeTo の特長、プログラム例、実験結果を示す。

4.1 SkeTo の概要

SkeTo は、PC クラスタなどの分散メモリ環境を主なターゲットとして、C++ と MPI によって実装された並列スケルトンライブラリである。最も重要な特長は、構成的アルゴリズム論に基づいて並列スケルトンを設計していることであり、その結果として融合変換などのプログラム最適化機能を実現している。SkeTo ライブラリ最新版は Version 1.0（2009 年 12 月）であり、SkeTo プロジェクトのウェブページ [21] からダウンロードすることができる。

SkeTo の主な特長は次の 4 つである。

- C++ と MPI によって実装している。もともと PC クラスタなどの分散メモリ環境を主な対象としてきたことによるが、近年のマルチコア CPU による計算機でも SkeTo を利用することができる。
- MPI の実装を内部に隠蔽し、ユーザへのインターフェイスには標準の C++ のみを利用しているため、C++ にてプログラミングができるユーザは並列計算のための特別な言語やライブラリを習得する必要はない。
- データ並列計算のための並列スケルトンを提供している。特に、分散リスト（一次元配列）の他に、分散行列（二次元配列）や分散木構造を提供しており、またそれらを操作する並列スケルトンはできるだけ類似したインターフェイスを持つようにしている。
- 並列スケルトンを用いたプログラムの融合変換による最適化機能を実現している。ユーザは、分かりやすい形でプログラムを書き効率の良いプログラムを得られるという利点がある。

図 2 に SkeTo ライブラリを用いて分散リスト as の値の分散を求めるプログラムを示す。このプログラムは、分散を求める次の定義

$$var = \frac{1}{n}\sum (a_i - ave) \quad \text{where} \quad ave = \frac{1}{n}\sum a_i$$

を元に、要素毎の計算を map に、総和を求める計算を reduce にそれぞれ置き換えることで自然に得られる。このプログラム例から分かるように、map や reduce などの並列スケルトンやそれらが操作する dist_list などの並列データ構造を利用することで並列プログラムを作成する。実装において利用している MPI はユーザから隠蔽されており、それらはプログラム中には一切出現しない。また、SkeTo が提供する並列スケルトンや並列データ構造の意味を逐次的にとらえれば、図 2 のプログラムは逐次プログラムとほとんど変わらないことになる。

4.2 融合変換によるプログラム最適化

並列スケルトンライブラリ SkeTo の特長のひとつに、融合変換によるプログラムの最適化機構を実現していることがある。融合変換 [22] とは、複

```
#include <iostream>
#include <sketo/sketo.h>
#include <sketo/list_skeletons.h>
const int n = 10000000;

using namespace std;
using namespace sketo;
using namespace sketo::list_skeletons;

struct gen
  : public functions::base<int (int)> {
  int operator()(int i) const {
    return i*i*i*i*i % 100;
  }
};

int sketo::main(int, char**) {
  dist_list<int> as;
  as = generate(n, gen());
  double ave
    = reduce(plus<double>(), as) / n;

  double var
    = reduce(plus<double>(),
        map(functions::square<double>(),
          map(bind2nd(minus<double>(),
ave),
            as))) / n;

  sketo::cout << var << endl;
}
```

図 2. 分散を計算する SkeTo プログラム

数の関数の呼び出しをひとつにまとめることで中間データを削減するプログラム変換である。本節では、SkeTo における融合変換による最適化について、その概要を示す。なお、本節の内容の詳細は文献 [23] にある。

図 2 にあるように、スケルトン並列プログラミングでは並列スケルトンを多く組み合わせることでプログラムを作成する。適切に計算を並列スケルトンに分割することで、そのプログラムは理解・解析しやすくなる。しかし一方で、計算が分割されることにより、その間でやりとりされる中間データがスケルトン並列プログラミングのオーバーヘッドとなってしまう。例えば、配列 a の値の平均値 ave が分かっているとき、分散を求める逐次プログラムは次のひとつのループによって実現することができる。

```
sqsum = 0;
for (i = 0; i < n; i++) {
  sqsum += (a[i] - ave) * (a[i] - ave);
}
var = sqsum / n;
```

しかしながら、図 2 ではふたつの map スケルトンとひとつの reduce スケルトンを用いている。そのプログラムの逐次的な意味は、次のようなプログラムに対応する。

```
for (i = 0; i < n; i++) {
   b[i] = a[i] - ave;
}
for (i = 0; i < n; i++) {
   c[i] = b[i] * b[i];
}
sqsum = 0;
for (i = 0; i < n; i++) {
   sqsum += c[i];
}
var = sqsum / n;
```

このプログラム中の配列 b や c はひとつのループで実現するときには不要な中間データである。なお、配列 b や c をもともとの配列 a で代用すればメモリ使用量の点ではオーバーヘッドがなくなるが、依然書き込みが必要であるためオーバーヘッドがなくなるわけではない。

このように、中間データを除去することがスケルトン並列プログラムの効率化において重要となる。最新版の SkeTo では C++ のテンプレートによるメタプログラミング技法のひとつである「式テンプレート（Expression Templates）」を用いることで、融合変換による最適化機構を実現している [23] 。式テンプレートを利用した融合変換は次のようにして実現されている。なお、この融合変換による最適化はコンパイル時に行われる。

(1) 並列スケルトンが呼び出された際に、その並列スケルトンが行うローカルな計算を表現するテンプレートオブジェクトを生成する。例えば、map スケルトンが呼び出された際には、mapobj クラスのオブジェクトを生成することのみ行う。
(2) 並列スケルトンの計算結果が分散リストに

代入されるとき、もしくは、並列スケルトンの内部でプロセッサ間通信が行われるときに、テンプレートオブジェクトによって表現された計算を実際に行う。このとき、可能な限り計算をまとめて行うようにする。

分散の計算の例を用いて説明する。まず、map の実際の計算は行われずに mapobj クラスのオブジェクトが生成される。特に、ふたつの map が呼ばれるため、reduce にはネストした mapobj オブジェクト、

```
mapobj<g, mapobj<f, dist_list<double> > >
```

(ただし、f は ave を減算する計算、g は二乗の計算をそれぞれ表す)が渡されることになる。次に、reduce の中で通信が行われる前に、オブジェクトとして渡されたふたつの map の計算とローカルに総和を求める計算をひとつのループによって実現する。これにより、コンパイルによって生成されるコードの主要部分は、ひとつのループによる計算と同じものとなる。

また、SkeTo ライブラリでは式テンプレートを利用することで、結果を入力に上書きして分散配列を再利用する最適化も実現している。これらの融合変換による最適化の効果については、次節で示す。

4.3 実験による性能評価

並列スケルトンライブラリ SkeTo を用いたプログラムの実行効率と融合変換による最適化の効果について報告する。

まず、簡単な問題の例として分散を計算するプログラムについて実験した。実験に用いたプログラムは次の 4 つである。

- OpenMP：単純な逐次プログラムを OpenMP を用いて共有メモリ向けに並列化したプログラム。
- SkeTo：SkeTo ライブラリを利用した図 2 のプログラムを、融合変換による最適化ありでコンパイルしたもの。
- SkeTo NO：SkeTo と同じプログラムであるが、融合変換による最適化を行わなかったもの。
- STL：C++ の標準テンプレートライブラリ STL のマルチコア向け実装 [24] を利用して並列化したプログラム。

データサイズはそれぞれ 2 億要素である。実験には、CPU に Intel Xeon E5430（2.66GHz、クアッドコア）を 2 つと 8GB のメモリを持つ計算機を用いた。コンパイラには GCC4.4.0、MPI の実装には mpich 1.2.7p1 を用いた。

表 1. 分散の計算の実行時間(単位：秒)

コア数	1	2	4	8
OpenMP	1.10	1.11	0.61	0.51
SkeTo	2.38	1.25	0.68	0.50
SkeTo NO	3.93	2.25	1.43	1.26
STL	2.60	1.64	1.24	1.19

実験の結果を表 1 に示す。SkeTo ライブラリを用いたプログラムは最適化を行わない場合においては、OpenMP によるプログラムや C++ の標準ライブラリを用いたプログラムに比べて性能が劣る。しかしながら、融合変換を行うことでその問題が解決され、結果として OpenMP を用いた効率の良いプログラムとほぼ同等の性能が得られる。なお、OpenMP は共有メモリ環境でのみ利用可能であるが、SkeTo を用いて作成したプログラムは分散メモリ環境へそのまま移行できる。

次に、より実用的な例として、BiCGStab 法(Bi-Conjugate Gradient Stabilized method; 双共役勾配安定化法）[25] による線形方程式ソルバの SkeTo ライブラリによる実装について報告する。この例では、ある研究機関において実際に利用されている Fortran によるプログラムをもとに、それを C++ によって書き換えたプログラムを対象として実験を行った。対象となる計算は、3 次元における近傍の畳み込みを行う演算（3 次元ステンシル計算）と内積の計算とを反復して行うものである。なお、本実験の詳細は METR テクニカルレポート(東京大学大学院情報理工学系研究科数理情報学専攻)にて発行される予定である。

この問題に対して、次に示す 4 つのプログラムを作成して実験を行った。なお、SkeTo を用いたプログラムについては、いずれも融合変換による最適化を有効にしてコンパイルしている。しかし、SkeTo-3D のプログラムにおいて予想される速度が達成されなかったため、人手による最適化をさらに行った Opt-3D を実装した。

- Seq：もともとの逐次プログラムを C++ によって書き直したプログラム。
- SkeTo-1D：データを 1 次元に並べて分散リストに格納し、データのシフトを行うスケルトンを利用して近傍計算を実現したスケルトン並列プログラム。

表 2. BiCGStab の実行時間（単位：秒）

コア数	1	2	4	8	16
Seq	21.08				
SkeTo-1D	48.09	25.05	14.42	12.08	16.05
SkeTo-3D	36.54	19.16	11.21	9.20	15.88
Opt-3D	32.36	17.43	10.74	8.06	13.18
Opt-3D-bk	32.39	21.43	18.88	10.16	6.07

- SkeTo-3D：3 次元のうち 1 次元方向のみ分散リストに格納し残りを構造体で扱うようにしたスケルトン並列プログラム。ネストした融合変換について人手にて行ったもの。
- Opt-3D：SkeTo-3D のプログラムに対して、現在の SkeTo では提供されていない動的確保オブジェクトの機能を拡張したスケルトン並列プログラム。

データサイズは 140 × 140 × 140 とし、BiCGStab 法の主要ループを 100 回反復した。実験には、CPU に Core2Quad Q6600（2.4GHz、クアッドコア）を 1 つと 4GB のメモリを持つ計算機 4 台がギガビットイーサネットによって接続された PC クラスタを用いた。使用するコアの割り当て方についてはラウンドロビン方式を用いた。コンパイラには、Intel Compiler 9.1、MPI の実装には mpich 1.2.7 を用いた。

実験の結果を表 2 に示す。Opt-3D については、使用するコアをできるだけひとつの計算機から連続して割り当てた場合を Opt-3D-bk として表示している。いずれのスケルトン並列プログラムも 8 コアまでの範囲において速度向上が見られる。8 コア、16 コアと使用するコアを増やした場合に速度が落ちているのは、ラウンドロビンによる割り当てのためネットワークへの負荷が増大したことが原因である。一番速度が速かったものは、プログラムに Opt-3D を使用し、16 コアをひとつの CPU のコアが連続するように割り当てた場合で、逐次プログラムの 3.5 倍の速度を達成した。しかし、1 コアでの実行において、もともとの逐次プログラム（Seq）と比較して約 1.5 倍の速度となっており、このオーバーヘッドの原因を見つけ改善することは今後の課題である。

5. 研究推進に向けた共同研究体制

著者らは、並列性忘却プログラミングおよびスケルトン並列プログラミングに関して、以下に示す 3 大学・1 研究所・1 企業から構成されるグループ（2010 年 4 月現在、教員を含むスタッフ 8 名、大学院学生 5 名）で研究を行っている。

- 東京大学武市研究室：研究の統括、および、Fortress を中心とした並列性忘却プログラミングと並列スケルトンライブラリ SkeTo の開発
- 電気通信大学岩崎研究室：並列スケルトンライブラリにおける要素技術とその応用
- 高知工科大学松崎研究室：並列スケルトンライブラリ SkeTo の開発とその応用
- 国立情報学研究所胡研究室：構成的アルゴリズム論に基づく並列スケルトンの定式化や最適化
- ㈱ HPC システムズ：実問題を扱う計算科学分野における並列性忘却プログラミングの応用

組織をまたいだ研究グループであることもあり、互いの研究進捗や研究方針の議論・確認のためおよそ一ヶ月に一度程度の研究ミーティングを持つようにしている。また、研究ミーティング以外にも、必要に応じてビデオ会議システムや Skype を使っての個別の議論を行うようにしている。また、SkeTo ライブラリなどのソフトウェア開発においては、バージョン管理ソフトウェア Subversion の共有リポジトリを用いて、これらのグループ間で共同開発することができる体制をとっている。

6. おわりに

本稿では、並列性忘却プログラミングに関する基本的なアイデアを紹介し、その具体的な研究としてスケルトン並列プログラミングおよび著者らが開発している並列スケルトンライブラリである SkeTo について示した。また、並列性忘却プログラミングの研究推進に向けた共同研究体制についても紹介した。

並列性忘却プログラミングの研究はまだ初期段階であり、今後より多くの研究を行うことが必要である。現在、並列計算が必要となる問題を多く集め、それらに対して並列性忘却プログラミングやスケルトン並列プログラミングの手法を適用できるかどうかの検証を行っている。

文献

[1] M. Cole,“Algorithmic Skeletons: Structural Management of Parallel Computation,”*Research Monographs in Parallel and Distributed Computing,* MIT Press, 1989.

[2] Wikipedia,“Algorithmic skeleton,”2010（URL = http://en.wikipeida.org/wiki/Algorithmic_skeleton）.

[3] 武市正人 ,“並列プログラム開発を抜本的に改革,”東京大学大学院情報理工学系研究科フォーカス , 2010（URL = http://www.i.u-tokyo.ac.jp/ news/focus/100215_1.shtml）.

[4] E. Allen, D. Chase, J. Hallett, V. Luchangco, J.-W. Maessen, S. Ryu, G.L. Steele Jr., and S. Tobin-Hochstadt,“The Fortress language specification version 1.0,”2008（URL = http://research.sun.com/projects/plrg/fortress.pdf）.

[5] G.E. Blelloch, J.C. Hardwick, J. Sipelstein, M. Zagha, and S. Chatterjee,“Implementation of a portable nested data-parallel language,”*Journal of Parallel and Distributed Computing,* Vol. 21, No. 4, pp. 4–14, 1994.

[6] F.W. Burton,“Encapsulating non-determinacy in an abstract data type with determinate semantics,”*Journal of Functional Programming,* Vol. 1, No. 1, pp. 3–20, 1991.

[7] 江本健斗, 寺田洋介, 松崎公紀, 胡振江, 武市正人,“最適値の並列探索のための improving value の Fortress 実装,”第 12 回プログラミングおよびプログラミング言語ワークショップ（PPL2010）論文集 , pp. 85–99, 2010.

[8] D.B. Skillicorn, M. Danelutto, S. Pelagatti, and A. Zavanella,“Optimizing data-parallel programs using the BSP cost model,”*Euro-Par '98 Parallel Processing, 4th International Euro-Par Conference, Proceedings,* LNCS Vol. 1470, pp. 698–703, 1998.

[9] Z. Hu, M. Takeichi, and W.-N. Chin,“Parallelization in calculational forms,”*POPL '98, Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages,* pp. 316–328, 1998.

[10] R. Loogen, Y. Ortega-Mallén, and R. Peña-Marí,“Parallel functional programming in Eden,”*Journal of Functional Programming,* Vol. 15, No. 3, pp. 431–475, 2005.

[11] N. Scaife, S. Horiguchi, G. Michaelson, and P. Bristow,“A parallel SML compiler based on algorithmic skeletons,”*Journal of Functional Programming,* Vol. 15, No. 4, pp. 615–650, 2005.

[12] S.-B. Scholz,“Single assignment C —— efficient support for high-level array operations in a functional setting,”*Journal of Functional Programming,* Vol. 13, No. 6, pp. 1005–1059, 2003.

[13] A. Benoit, M. Cole, S. Gilmore, and J. Hillston,“Flexible skeletal programming with eSkel,”*Euro-Par 2005, Parallel Processing, 11th International Euro-Par Conference, Proceedings,* LNCS Vol. 3648, pp. 761–770, 2005.

[14] H. Kuchen,“A skeleton library,”*Euro-Par 2002, Parallel Processing, 8th International Euro-Par Conference, Proceedings,* LNCS Vol. 2400, pp. 620–629, 2002.

[15] M. Aldinucci, S. Campa, P. Ciullo, M. Coppola, S. Magini, P. Pesciullesi, L. Potiti, R. Ravazzolo, M. Torquati, M. Vanneschi, and C. Zoccolo,“The implementation of ASSIST, an environment for parallel and distributed programming,”*Euro-Par 2003, Parallel Processing, 9th International Euro-Par Conference, Proceedings,* LNCS Vol. 2790, pp. 712–721, 2003.

[16] K. Matsuzaki, H. Iwasaki, K. Emoto, and Z. Hu,“A library of constructive skeletons for sequential style of parallel programming,”*InfoScale '06: Proceedings of the 1st international conference on Scalable information systems,* 2006.

[17] M. Aldinucci, M. Danelutto, and P. Dazzi,“Muskel: an expandable skeleton environment,”*Scalable Computing: Practice and Experience,* Vol. 8, No. 4, pp. 325–341, 2007.

[18] M. Leyton and J.M. Piquer,“Skandium: Multi-core programming with algorithmic skeletons,”*Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, PDP 2010,* pp. 289–296, 2010.

[19] J. Dean and S. Ghemawat,“MapReduce: Simplified data processing on large clusters,”*6th Symposium on Operating System Design and Implementation (OSDI2004),* pp. 137–150, 2004.

[20] J. Reinders,“Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism,”O'Reilly Media, Inc., 2007.

[21] SkeTo Project, Sketo project homepage, 2009

（URL = http://www.ipl.t.u-tokyo.ac.jp/sketo/）.

[22] A.J. Gill, J. Launchbury, and S.L. Peyton Jones, "A short cut to deforestation," *FPCA '93 Conference on Functional Programming Languages and Computer Architecture,* pp. 223–232, 1993.

[23] K. Matsuzaki and K. Emoto, "Implementing fusion-equipped parallel skeletons by expression templates," *Draft Proceedings of the 21st International Symposium on Implementation and Application of Functional Languages,* pp. 100–115, 2009.

[24] J. Singler, P. Sanders, and F. Putze, "The multi-core standard template library," *Euro-Par 2007, Parallel Processing, 13th International Euro-Par Conference, Proceedings,* LNCS Vol. 4641, pp. 682–694, 2007.

[25] H. A. van der Vorst, "BI-CGSTAB: a fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing,* Vol. 13, No. 2, pp. 631–644, 1992.

Parallelism-Oblivious Programming and Towards Promoting the Research

Kiminori Matsuzaki
（Received : April 28th, 2010）
School of Information, Kochi University of Technology
185 Miyanokuchi, Tosayamada, Kami city, Kochi 782-8502

E-mail: matsuzaki.kiminori@kochi-tech.ac.jp

Abstract: Recent computer hardware such as multicore CPUs gains its performance by parallelism. However, developing software for an effective use of those environments — parallel programming — is still a hard task, because it is done as an extension of sequential programming with data distribution and inter-processor communication. The author is working in a group that studies a new methodology of parallel programming, namely parallelism-oblivious programming, to support parallel programming without being aware of detailed parallelism. With this methodology, we will be able to develop correct parallel programs more easily by exploiting the parallelism that problems and notations essentially have. This paper discusses the basic ideas of the parallelism-oblivious programming, and shows our results on skeletal parallel programming as a concrete study on the parallelism-oblivious programming. This paper also reports how we collaborate in the group to promote the research of the parallelism-oblivious programming.