スケルトン並列プログラミング

胡 振江 [1]　岩崎 英哉 [2]

[1] 東京大学大学院情報理工学系研究科
hu@mist.i.u-tokyo.ac.jp
[2] 電気通信大学情報工学科
iwasaki@cs.uec.ac.jp

1　はじめに

近年，PC ハードウェアの高性能化と低価格化が進み，PC クラスタに代表される並列計算機が身近なものとなった．その結果，様々な並列計算機を利用して並列計算を行おうとする人が増え続けているが，その中には並列計算機に対する知識をあまり持たない人も少なくない．そのような人にとって効率的な並列プログラムを作成するのは，プロセッサ間通信，同期，資源の分配など考慮すべき点が多く，敷居が高いと言われている．さらに，ある並列計算環境において効率の良いように最適化して書かれた並列プログラムは，別の環境では実行できなかったり効率が著しく落ちることもあり，移植性の点で問題がある．

以上のような問題点を解決するための有効な方法として期待されているのが，スケルトン並列プログラミング (skeletal parallel programming) である．スケルトン並列プログラミングは，1980 年代後半に Murray Cole [3] によって提案された．スケルトン並列プログラミングでは,「並列スケルトン」と呼ばれる並列処理の頻出する計算パターンを抽象化したライブラリ関数をあらかじめ実装しておき，並列スケルトンの組み合わせで並列プログラムを作成する．並列スケルトンを基本ブロックと考えれば，まるで積木ゲーム (図 1) のようにして並列プログラムを作成することができる．

通常の並列プログラミングの手法と比べると，スケルトン並列プログラミングには次のような特長がある．

機能と実装の分離　並列スケルトンによって，並列計算における高レベルの機能と低レベルの実装が分離される．ユーザからは低レベルの実装は隠蔽されるため，プロセッサ間通信，同期，資源の分配といった実装の詳細を理解しなくても，容易に並列プログラムを作成することができる．さらに，作成されたスケルトン並列プログラムは特定の並列環境に依存しない．

「逐次的」なプログラミングスタイル　各スケルトンは機能的には逐次的な意味を持つので，並列計算の知識を持たない人でも，並列スケルトンのライブラリ関数を利用するだけで，逐次プログラムを書く感覚で並列プログラムを記述できる．このようにして作成した並列プログラムは，デッドロックを発生することがない．

コストモデル　スケルトンは基本的な並列処理パターンを抽象化したものなので，一般的で複雑なプログラムより，並列計算のコストを見積りやすい．コストの見積り (コストモデル) は効率のよいスケルトン並列プログラムの開発や自動的最適化などに大きな役割を果たすと考えられる．

系統的な並列プログラムの開発　スケルトンの構造に隠れた代数的性質を追究することにより，ス

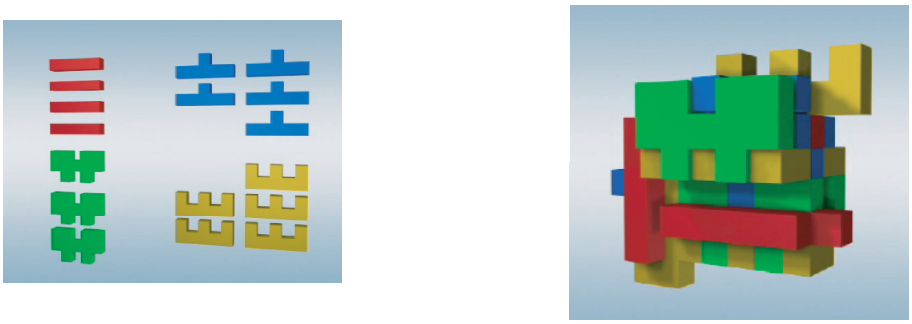

図 1: 並列プログラムを積み木のように構築する

ケルトン間の変換やスケルトンの合成法等を明らかにし，仕様から効率のよい並列プログラムを系統的に開発する方法論が期待できる．

本稿はスケルトン並列プログラミングの基本的アイデアを分かりやすく紹介する．まず，スケルトン並列プログラミングの歴史と背景を概観する．次いで，代表的な基本並列スケルトンを紹介した後，並列スケルトンの実装手法に触れ，スケルトンを用いた並列プログラムの構築手法を例示する．最後に，今後の展開と課題を示す．

2 研究の背景・歴史

スケルトンの概念は，Cole [3] より提案された．彼は，多くの並列計算問題に頻繁に出現する共通の計算パターンを並列スケルトンとして抽象化し，このような並列スケルトンの効率的な実装，最適化手法，そして実用問題への応用を中心に研究を行った．しかしながら，一つの問題を基本的にスケルトン中から一つ選んで解くという仮定があったため，スケルトンの定式化と合成法が明確ではなく，スケルトン並列プログラムの系統的な開発法が課題となった．

スケルトン並列プログラミングをより形式的に議論するためには，スケルトンを定式化するための理論が必要になる．Skillicorn [11] は，効率的な逐次プログラムを合成するための理論として知られている構成的アルゴリズム論 BMF (Bird-Meertens Formalism) [2] を並列計算の枠組で新しく解釈し，並列プログラミングモデルとして利用できることを示した．このモデルの中で最も重要な概念の一つはリスト準同型 (List Homomorphism) である．リスト準同型は，後述するデータ並列スケルトンと緊密な関係を持つ．データ並列スケルトンはリスト準同型で表現できる．また，逆にリスト準同型はデータ並列スケルトンの組み合わせで表現できる．これにより，リスト準同型の系統的な導出手法をスケルトン並列プログラムの系統的な開発に応用できる [5, 6].

スケルトン並列プログラミングに関する方法論の研究が進展する一方，スケルトン並列プログラムを効率的に実行可能な環境も多く開発されてきた [8, 7, 9, 1]. Kuchen [7] は，データ並列タスケルトンとタスク並列スケルトン (後述) による並列プログラミングを支援するための C++のスケルトンライブラリを作成し，多くの実用例を用いて実験を行いその有効性を示した．また，実用的で総合開発環境として有名なのは ASSIST [1] である．

最近ではスケルトン並列プログラミングの研究がますます注目され，2004 年の並列・分散に関する国際会議 EuroPar 2004 でも招待講演のテーマの 1 つとして紹介されている．現在，世界で 30 チーム以上[1] の研究者が，スケルトン並列プログラミングに関して理論から応用まで幅広く積極的に研究に取り組んでいる [10].

[1] http://homepages.inf.ed.ac.uk/mic/Skeletons/

3　基本スケルトン

スケルトン並列プログラミングで最も重要な課題は，何を並列スケルトンとして用意すればよいかということである．数多く用意すればどれを選べばよいか分からなくなる一方，少なければ複雑な問題に対処するのが難しくなる．アドホックな方法で基本ブロックを定めた例はあるが，いろいろなプログラムに過不足なく使えるとはいえない．本節で紹介する並列スケルトンは少数ではあるが，それらの組み合わせでかなり高い表現力を持つ．

並列スケルトンは，データ並列スケルトン (data parallel skeleton) とタスク並列スケルトン (task parallel skeleton) に分類される．データ並列スケルトンは同じ操作をデータの異なる部分に独立に適用するのに対し，タスク並列スケルトンは一つの操作に含まれるいくつかのタスクを異なるプロセッサで独立に実行する．

3.1　データ並列スケルトン

データ並列スケルトンはデータの異なる部分に，独立に同じ操作を行うような計算パターンである．本稿では簡単のため，木や二次元データを考えず，同種の要素からなるリスト $[x_1, x_2, \ldots, x_n]$ を用いてデータ並列スケルトンを説明する．重要なデータ並列スケルトンは，map，reduce，scan，zip の 4 つである．

map はリストの各要素に関数 k を適用するスケルトンである．

$$\mathsf{map}\ k\ [x_1, x_2, \ldots, x_n] = [k\ x_1, k\ x_2, \ldots, k\ x_n]$$

reduce は，リストの各要素を結合的な二項演算子 $\oplus$ で結合し，一つの値を返すスケルトンである．

$$\mathsf{reduce}\ (\oplus)\ [x_1, x_2, \ldots, x_n] = x_1 \oplus x_2 \oplus \cdots \oplus x_n$$

scan は reduce の全ての途中経過をリストとして返すスケルトンである．$\oplus$ を結合的な二項演算子，$\oplus$ の単位元を e とすると，次のようになる．

$$\begin{aligned}&\mathsf{scan}\ (\oplus)\ [x_1, x_2, \ldots, x_n]\\ &= [e, e \oplus x_1, e \oplus x_1 \oplus x_2, \ldots, e \oplus x_1 \oplus \cdots \oplus x_n]\end{aligned}$$

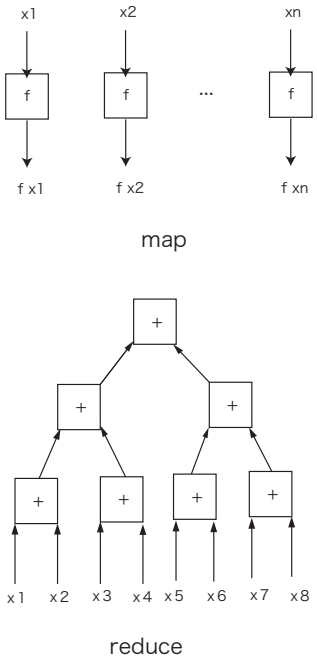

図 2: データ並列スケルトン

zip は 2 つのリストの要素を組にして，1 つのリストにするスケルトンである．

$$\begin{aligned}&\mathsf{zip}\ [x_1, x_2, \ldots, x_n]\ [y_1, y_2, \ldots, y_n]\\ &= [(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)]\end{aligned}$$

図 2 に，map と reduce の直感的な並列計算構造を示す．他のデータ並列スケルトン zip と scan も，それぞれ map，reduce に似た方法により，同じオーダーの計算コストで計算できる [11].

並列スケルトンで書かれたプログラムの並列計算コストは，容易に見積ることができる．たとえば，プロセッサが十分にあると仮定すると，map $k\ [x_1, x_2 \ldots . x_n]$ の並列計算時間は k の並列計算時間と同じであり，reduce $(+)\ [x_1, x_2 \ldots . x_n]$ の並列計算時間は $\log(n)$ となる．以上より，二つのスケルトンの組み合わせ reduce $(+)$ (map $f\ [x_1, x_2 \ldots . x_n]$) の並列計算時間は $\log(n)$ ということが容易に導ける．

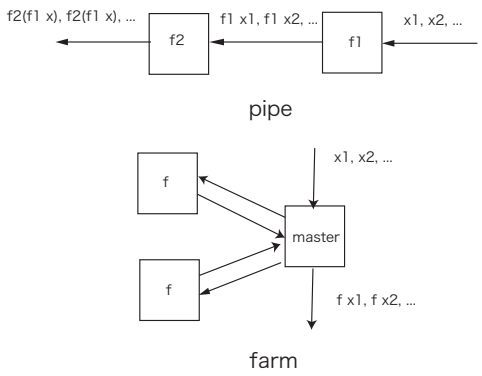

図 3: タスク並列スケルトン

3.2 タスク並列スケルトン

タスク並列スケルトンはデータのストリームを引数に取り，データストリームを返す並列計算を行うようなスケルトンである．本稿では，$\langle x_1, x_2, \ldots, x_n \rangle$ で，n 個の要素からなるストリームを表す．重要なタスク並列スケルトンは図 3 に示す pipe と farm である．

pipe f_1 f_2 では，二つの計算 (タスク) f_1 と f_2 をパイプライン的に結合して並列計算を行う．

$$\begin{aligned}&\mathsf{pipe}\ f_1\ f_2\ \langle x_1, x_2, \ldots, x_n \rangle \\ &\quad = \langle f_1(f_2\ x_1), f_1(f_2\ x), \ldots, f_1(f_2\ x) \rangle\end{aligned}$$

farm m f では，f というタスクを m 個生成し，m 個のタスクがそれぞれ独立に計算することによって入力のストリームの各要素に f を適用する．farm の中のマスタプロセッサは，効率的にデータを分散し，また各タスクで計算した結果をまとめて順番の正しいストリームの結果を出力する仕事を担当する．

$$\begin{aligned}&\mathsf{farm}\ m\ f\ \langle x_1, x_2, \ldots, x_n \rangle \\ &\quad = \langle f\ x_1, f\ x_2, \ldots, f\ x_n \rangle\end{aligned}$$

4 スケルトン並列プログラム

4.1 簡単な例題

並列スケルトンを用いただけでは，簡単な問題しか解けないように一見思えるかもしれないが，実際は，並列スケルトンの組み合わせで，科学計算，画像処理，情報検索，データマイニング等，様々な応用プログラムを記述することができる．一つの例として，最近，Google のグループがデータ並列スケルトン map と reduce を用いて，実際の Google の検索エンジンの中の次の計算を効率的に実現したことが発表 [4] され，スケルトン並列プログラミングの実用性が脚光を浴びた．

- 分散的 grep (distributed grep)
- 分散的整列 (distributed sort)
- ウェブのリンクグラフの逆転 (web link-graph reversal)
- ウェブの訪問ログの統計 (web access log stats)
- 逆インデクスの生成 (inverted index construction)
- 文書クラスタリング (document clustering)
- 機械学習 (machine learning)

ここで，スケルトン並列プログラミングの簡単な例として，リスト $xs = [x_1, x_2, \ldots, x_n]$ が与えられたとき，その分散 var を求めるプログラムを考える．分散は次の式で求められる．

$$\begin{aligned}var &= \sum_{i=1}^{n} (x_i - ave)^2 / n \\ ave &= \sum_{i=1}^{n} x_i / n\end{aligned}$$

このような簡単な問題でも，標準的なメッセージパッシングのライブラリ MPI などによって並列計算できるようにプログラミングすることは容易では

ない．ところが，これを並列スケルトンを用いて記述すると，次のようになる．

$$
\begin{aligned}
ave &= \mathsf{reduce}\ (+)\ xs/n \\
xs' &= \mathsf{map}\ subsq\ xs \\
&\quad \text{where } subsq\ x = (x - ave)^2 \\
var &= \mathsf{reduce}\ (+)\ xs'/n
\end{aligned}
$$

この記述であればきわめて容易であり，並列計算に関する低レベルの実装などを一切考慮する必要はない．

4.2 効率

単純な機能を持つ並列スケルトンを多数組み合せて記述したプログラムは，実行効率が良くないのではないかと思えるかもしれない．この点に関して，Kuchen による興味深い実験結果がある．

Kuchen は，自身で実装した C++ による並列スケルトンライブラリを用いて，行列乗算，最小パス問題，ガウス消去法，FFT といった典型的な並列計算を記述可能であることを示した上で，並列スケルトンを利用せず MPI のみを用いて書いた標準的な並列プログラムとの性能比較 (表 1) を行った [7]．その結果，スケルトン並列プログラムの計算時間は，プロセッサ間通信，同期などを考慮して書いた効率的な並列プログラムの，2 倍以内に抑えられることが分かった．

並列スケルトンを用いてプログラムを作成するか，あるいは，並列スケルトンを用いずに通信や同期を考慮した最適なプログラムを作成するかは，プログラム作成の手間とプログラム実行効率のトレードオフの関係にある．Kuchen による実験結果は，スケルトン並列プログラムを用いれば，プログラム開発が容易であるだけでなく，実用的な効率もある程度保証できる (極端に遅いわけではない) ということを示している．

5 並列スケルトンライブラリ

並列スケルトンは，その単純さの故に，ほとんどの並列計算環境で効率的に実現できる [7]. 本節では，C++ と MPICH を用いて実装した C++のデータ並列スケルトンライブラリ [12] を通して，その使用法を簡単に紹介する．このライブラリはリスト以外にも，二次元データ，木などのデータ構造を扱えるが，ここではリストの並列スケルトンを用いて説明する．

並列計算では，リストなどのデータを各計算機に均等に分散して，データに対する操作を同時に行うことで，並列処理を実現する必要がある．並列スケルトンのライブラリでは，リストを `dist_array` クラスとして実現し，データの分散や収集といった分散並列特有の操作を，リストの構成子の中に閉じ込め，ユーザから隠蔽している．

たとえば，整数の配列 `array` の要素を各計算機に分散させてリストを生成する場合，次のように記述する．

```
dist_array<int> *as =
new dist_array<int>(array, array_size);
```

このようにして生成されたリストに対して，並列スケルトンを呼び出して並列プログラムを記述する．たとえば，前節で示した分散を求めるアルゴリズムは，次のように記述する．

```
sum = as->reduce(add);
ave = sum / n;
as->map_ow(sub_ave);
as->map_ow(square);
sq_sum = as->reduce(add);
var = sq_sum / n;
```

6 今後の展開

スケルトン並列プログラミングは，今後ますますその重要性が増すと考えられる．これまでは学術的な研究が多く，実用的なシステムの開発や説得力の

表 1: スケルトン並列プログラムと普通並列プログラムとの性能比較 [7]

応用例	データサイズ	プロセッサ数 = 4			プロセッサ数 = 16		
		スケルトンプログラム (秒)	MPI プログラム (秒)	割合	スケルトンプログラム (秒)	MPI プログラム (秒)	割合
行列乗算	256	0.459	0.413	1.11	0.131	0.124	1.06
	512	4.149	3.488	1.19	1.057	0.807	1.31
	1024	35.203	29.772	1.18	8.624	6.962	1.24
最小パス問題	1024	393.769	197.979	1.99	93.825	44.761	2.10
ガウス消去法	1024	13.816	9.574	1.44	7.401	4.045	1.83
FFT	2^{18}	2.127	1.295	1.64	0.636	0.403	1.58

高い応用例はまだ決して多いとはいえない．今後は，次のような問題を解決することが重要と考えられる．

まず第一に，複雑な問題を解く際に，どの並列スケルトンを選択すれば良いか，また，どのように並列スケルトンを組み合わせれば性能の良い並列プログラムが作れるのかに対する，系統的な設計法が整理されてるとはいえない．

第二に，スケルトンの使用により生じるデータ通信オーバーヘッドなどを取り除くためには，どのようにスケルトン並列プログラムを最適化すればよいかを検討する必要がある．これまで最適化手法に関する研究は少なく，数万行規模のスケルトンプログラムを対象に自動的に最適化できるシステムはまだ存在しない．

最後に，これまでの研究の多くは一次元のデータ (リスト) を扱うスケルトンを対象としており，計算機科学及びその様々な応用分野において基本的な二次元配列や木を扱う並列スケルトンの研究は少なく，僅かに存在する実装も定式化が不十分であり，適用範囲がまだ狭い．

以上で述べた問題点を解決することによって，スケルトン並列プログラミングは，一般のユーザにも受け入れられる標準的な並列プログラム開発手法となることが期待される．

参考文献

[1] Marco Aldinucci, Sonia Campa, Pierpaolo Ciullo, Massimo Coppola, Silvia Magini, Paolo Pesciullesi, Laura Potiti, Roberto Ravazzolo, Massimo Torquati, Marco Vanneschi, and Corrado Zoccolo. The implementation of assist, an environment for parallel and distributed programming. In *EuroPar'02, LNCS*, pages 712–721, 2003.

[2] R. Bird. An introduction to the theory of lists. In M. Broy, editor, *Logic of Programming and Calculi of Discrete Design*, pages 5–42. Springer-Verlag, 1987.

[3] M. Cole. *Algorithmic skeletons : a structured approach to the management of parallel computation.* Research Monographs in Parallel and Distributed Computing, Pitman, London, 1989.

[4] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *6th Symposisum on Operating Systems Design and Implementation (OSDI'04), San Francisco*, pages 137–150, December 2004.

[5] S. Gorlatch. Systematic efficient parallelization of scan and other list homomorphisms. In *Annual European Conference on Parallel Processing, LNCS 1124*, pages 401–408, LIP, ENS Lyon, France, August 1996. Springer-Verlag.

[6] Z. Hu, H. Iwasaki, and M. Takeichi. Formal derivation of efficient parallel programs by construction of list homomorphisms. *ACM Transactions on Programming Languages and Systems*, 19(3):444–461, 1997.

[7] Herbert Kuchen. A skeleton library. In *EuroPar'02, LNCS*. Springer-Verlag, August 2002.

[8] R. Loogen, Y. Ortega-Mallén, R. Peña, S. Priebe, and F. Rubio. Parallelism Abstractions in Eden. In Fethi A. Rabhi and Sergei Gorlatch, editors, *Patterns and Skeletons for Parallel and Distributed Computing*, volume 2011 of *LNCS*. Springer, 2002.

[9] Kiminori Matsuzaki, Kazuhiko Kakehi, Hideya Iwasaki, Zhenjiang Hu, and Yoshiki Akashi. A fusion-embedded skeleton library. In *International Conference on Parallel and Distributed Computing (EuroPar 2004)*, pages 644–653. Springer, LNCS 3149, August 2004.

[10] F.A. Rabhi and S. Gorlatch (eds). *Patterns and Skeletons for Parallel and Distributed Computing*. Springer Verlag, 2002.

[11] D.B. Skillicorn. *Foundations of Parallel Programming*. Cambridge University Press, 1994.

[12] 明石 良樹, 松崎 公紀, 岩崎 英哉, 筧 一彦, and 胡 振江. 最適化機構を持つ c++ 並列スケルトンライブラリ. **コンピュータソフトウエア**, 22, 2005. 掲載予定.