解説

スケルトン並列プログラミング

胡　振江
東京大学大学院情報理工学系研究科
hu@mist.i.u-tokyo.ac.jp

岩崎 英哉
電気通信大学情報工学科
iwasaki@cs.uec.ac.jp

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，さまざま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．しかし，その中には並列プログラミング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，敷居が高いといわれている．さらに，ある並列計算環境において効率が良くなる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 [9]．スケルトン並列プログラミングは，1980 年代後半に 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における頻出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，並列スケルトンを組み合わせること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　各並列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　スケルトンは簡単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) は効率の良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　スケルトンの構造の背後にある代数的性質を追究することにより，ス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．

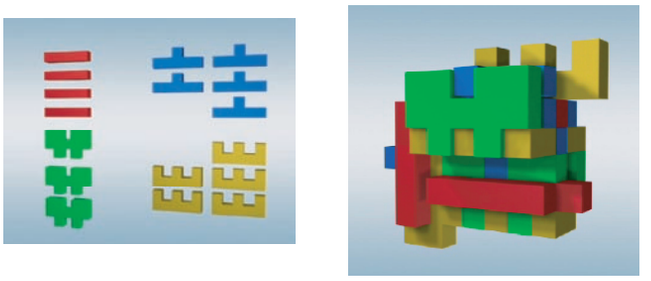

図 -1　並列プログラムを積み木のように構築する

本稿はスケルトン並列プログラミングの基本的アイディ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．

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，基本的に 1 つの問題に対してスケルトン中から 1 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[10] は，効率的な逐次プログラムを合成するための理論である Bird-Meertens Formalism (BMF)[2] を並列計算の枠組みで新しく解釈し，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．BMF はその後，再帰的なデータ型やそのデータの生成／消費関数を数理的な対象としてとらえ，これらの関数の合成に関するプログラム変換をアルゴリズム的に与える “構成的アルゴリズム論 (constructive algorithmics)” [12] へと発展していった．

BMF あるいは構成的アルゴリズム論のモデルの中で最も重要な概念の 1 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，また逆に，リスト準同型はデータ並列スケルトンの組合せで表現できる．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ミングの実行環境も多く開発されてきた．中でも有名なのは，P3L，Skil，eSkel などである [9)]．また，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++ のライブラリ Muskel 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．さらに，実用的な総合開発環境として有名なものに ASSIST[1] がある．

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 チーム近く☆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 [9)]．

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，どのような処理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的な並列スケルトン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．本章で紹介する並列スケルトンは少数ではあるが，それらの組合せは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

データ並列スケルトンはデータの異なる部分に，同時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 2 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 f を適用するスケルトンである．

$$\mathbf{map}\ f\ [x_1, x_2, \ldots, x_n] = [f\,x_1, f\,x_2, \ldots, f\,x_n]$$

reduce は，リストの各要素を結合的な二項演算子 $\oplus$ で結合し，1 つの値を返すスケルトンである．

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

scan は **reduce** のすべ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とすると，次のようになる．

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

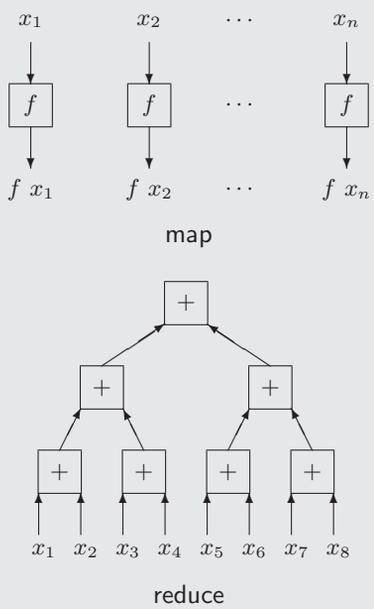

図-2　データ並列スケルトン

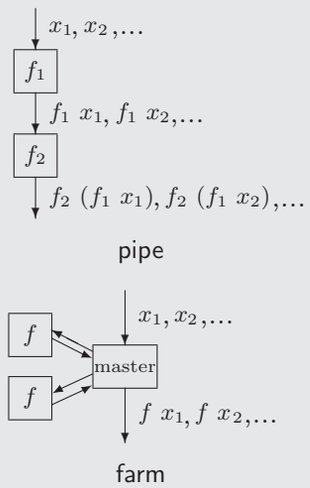

図-3　タスク並列スケルトン

scan(⊕) $[x_1, x_2, \ldots, x_n] = [x_1, x_1 \oplus x_2, \ldots, x_1 \oplus \ldots \oplus x_n]$

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．

zip $[x_1, x_2, \ldots, x_n]\ [y_1, y_2, \ldots, y_n]$
$= [(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)]$

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 [10]．

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，容易に見積もることができる．たとえば，プロセッサが十分にあると仮定すると，**map** $k_f\ [x_1, x_2, \ldots, x_n]$ の並列計算時間は k_f 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，2 つのスケルトンの組合せ **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．

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，2 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．

pipe $f_1\ f_2\ \langle x_1, x_2, \ldots, x_n \rangle = \langle f_2(f_1\ x_1), f_2(f_1\ x), \ldots, f_2(f_1\ x) \rangle$

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．

farm $m\ f\ \langle x_1, x_2, \ldots, x_n \rangle = \langle f\ x_1,\ f\ x_2, \ldots, f\ x_n \rangle$

スケルトン並列プログラム

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合せで，科学計算，画像処理，情報検索，データマイニング等，さまざまな応用プログラムを記述することができる．1 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以下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 grep
- 分散ソート

- Web のリンクグラフの逆転
- Web の訪問ログの統計
- 逆インデクスの生成
- 文書クラスタリング
- 機械学習

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．

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

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 などによって並列プログラムを作成することは面倒である．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．

$$ave = \textbf{reduce}\ (+)\ xs\ /\ n$$
$$xs' = \textbf{map}\ subsq\ xs$$
$$\text{where}\ subsq\ x = (x - ave)^2$$
$$var = \textbf{reduce}\ (+)\ xs'\ /\ n$$

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．

効率

単純な機能を持つ並列スケルトンを多数組み合わ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 [7]，および，プログラム変換による最適化の両面から説明する．

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．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，およそ約 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．

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

表 -1　スケルトン並列プログラムと通常の並列プログラムとの性能比較（文献 7）より抜粋）

プログラム最適化の側面からは，先にも述べたスケルトンを定式化する枠組みとしての BMF あるいは構成的アルゴリズム論がある．これらの分野では，2 つの関数の連続した呼出し（関数合成）を 1 つにまとめあげる融合変換に関する研究が活発に行われている．これらの研究成果は，2 つの並列スケルトンの連続した呼出しを 1 つの並列スケルトン呼出しに変換して効率を向上させるような，スケルトン並列プログラムの最適化に取り入れられると期待される [8]．現段階では，このような最適化を十分に行うシステムはないが，最適化機構を部分的に取り入れたシステムにおける簡単な実験 [11] では，最適化後のプログラムは最適化前と比較して実行速度が 7.8 ～ 29.7% 向上したことが報告されている．このことは，スケルトン並列プログラムの実用的な効率のさらなる向上の可能性を示唆している．

並列スケルトンライブラリ

並列スケルトンは，その単純さゆえに，ほとんどの並列計算環境で効率的に実現できる [7]．本章では，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 [11] 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，2 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に対する操作を同時に行うために，データを各計算機に適切に分散する必

要がある．並列スケルトンのライブラリでは，リストを dist_list 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計算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，大きさ n の double の配列 array の要素を各計算機に分散させたリストを生成する場合，次のように記述する．このライブラリでは，リストを配列として表現している．

```
dist_list<double> *as =
  new dist_list<double>(array, n);
```

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，前章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．

```
ave = list_skeletons::reduce
            (add, 0.0, as) / n;
list_skeletons::map_ow(sub(ave), as);
list_skeletons::map_ow(sqr, as);
var = list_skeletons::reduce
            (add, 0.0, as) / n;
```

ここで map_ow は，**map** の結果を入力のリスト（配列）に上書きするものである．このようにして，前章で示した記述とほぼ同様の形で，C++ の並列プログラムを書くことができる．

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 1 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 2 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．先にも述べたように，構成的アルゴリズム論の研究成果を最適化に応用できるが，これまでは，このような最適化機構を実際にシステムに組み込む研究は少なく，実際，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を行うことが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 1 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，計算機科学およびそのさまざまな応用分野において基本的なデータ構造である 2 次元配列や木を扱う並列スケルトンの研究は少なく，わず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) Aldinucci, M. et al.: The Implementation of ASSIST, an Environment for Parallel and Distributed Programming, Proc. Annual European Conference on Parallel Processing (Euro-Par2002), Lecture Notes in Computer Science 2400, pp.712-721 (2002).
2) Bird, R.: An Introduction to the Theory of Lists, Logic of Programming and Calculi of Discrete Design, pp.5-42, Springer-Verlag (1987).
3) Cole, M.: Algorithmic Skeletons: a Structured Approach to the Management of Parallel Computation, Research Monographs in Parallel and Distributed Computing, Pitman, London (1989).
4) Dean J. and Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, Proc. 6th Symposisum on Operating Systems Design and Implementation (OSDI'04), pp.137-150 (2004).
5) Gorlatch, S.: Systematic Efficient Parallelization of Scan and Other List Homomorphisms, Proc. Annual European Conference on Parallel Processing (Euro-Par1996), Lecture Notes in Computer Science 1124, pp.401-408 (1996).
6) Hu, Z., Iwasaki, H. and Takeichi, M.: Formal Derivation of Efficient Parallel Programs by Construction of List Homomorphisms, ACM Transactions on Programming Languages and Systems, 19(3): pp.444-461 (1997).
7) Kuchen, H.: A Skeleton Library, Proc. Annual European Conference on Parallel Processing (Euro-Par2002), Lecture Notes in Computer Science 2400, pp.712-721 (2002).
8) Matsuzaki, K., Kakehi, K., Iwasaki, H., Hu, Z. and Akashi, Y.: A Fusion-embedded Skeleton Library, Proc. Annual European Conference on Parallel Processing (Euro-Par2004), Lecture Notes in Computer Science 3149, pp.644-653 (2004).
9) Rabhi, F. A. and Gorlatch, S. (eds): Patterns and Skeletons for Parallel and Distributed Computing, Springer-Verlag (2002).
10) Skillicorn, D. B.: Foundations of Parallel Programming, Cambridge University Press (1994).
11) 明石良樹，松崎公紀，岩崎英哉，筧　一彦，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ェア，Vol.22, No.3 (2005).
12) 岩崎英哉：構成的アルゴリズム論，コンピュータソフトウェア，Vol.15, No.6, pp.57-70 (1998).

（平成 17 年 9 月 8 日受付）