［フォーラム］

プログラム運算による並列プログラミング

胡 振江

1 はじめに

並列プログラムを開発する際には，並列性を持ついくつかの並列計算パタンを基本ブロックとして，プログラマが並列プログラムを構築する手法 [4] (並列パタン手法 (skeleton approach)) が広く使われている．並列計算パタンによる抽象化によって，プログラマは低レベルの並列計算を考慮せず並列プログラムを開発することができる．しかし，問題が 1 つ残っている．効率のよい並列プログラムを構築する際に，どのような並列パタンを選び，またそれをどのように組み合わせればよいかをプログラマが考えなければならない．

これを背景に論文 [6]では，自然な再帰的定義から並列パタンを用いた効率のよい並列プログラムに変換する逆融合運算定理 (diffusion calculation theorem) を提案した．本小論文では，この定理を拡張するとともに，その応用例を通して，運算による並列プログラミングのアイデアを簡単に述べる．

2 プログラム運算・逆融合運算定理

数学分野では加算，乗算，微分，積分などの数式の運算 (calculation) が行われる．運算には，代数的性質に基づいて式を形式的に操作し，式の “意味” を解釈しないという特徴がある．例えば，123×456 のような運算を行う時には，代数的性質に基づく変換操作を行うことにより，1, 2 などの記号の意味を考えることなく，結果を得ることができる．

計算機プログラムには，問題を解くプロセスが記述されている．しかし，単にそれだけではなく，代数的性質も備えているので，運算によるプログラムの変換或いは操作をすることが可能である．

このプログラムの運算を支援するものとして，Bird Meertens Formlism (BMF と略記する) [1] [2] がよく知られている．BMF では，プログラムを変換するための運算規則 (calculational rules) が組織的に構築されており，分かりやすく書かれたプログラムにこれらの運算規則を適用していくことにより，効率のよいプログラムが構成的に構築される．この運算技法をアルゴリズムの導出やコンパイラにおけるプログラムの最適化に適用する研究は数多くなされており，よい成果がでている．

しかし，並列プログラミングのための運算規則 [7] [5] [6]はまだ少ない．我々が提案した逆融合運算定理 [6]はその中の 1 つであり，自然な再帰的定義から並列パタンを用いた並列プログラムに変換する運算規則である．この逆融合運算定理で用いる並列パタンは次のようなものである．

- map 操作 $*$.
$$f * [x_1, x_2, \ldots, x_n] = [f\, x_1, f\, x_2, \ldots, f\, x_n].$$
- reduce 操作 $/$.
$$\oplus/\, [x_1, x_2, \ldots, x_n] = x_1 \oplus x_2 \oplus \cdots \oplus x_n.$$
- scan 操作 $/\!\!/$.
$$\oplus/\!\!/\, [x_1, x_2, \ldots, x_n] = [\iota_\oplus, x_1, x_1 \oplus x_2, \ldots, x_1 \oplus x_2 \oplus \cdots \oplus x_n].$$
- zip 操作 zip.
$$zip\ [x_1, x_2, \ldots, x_n]\ [y_1, y_2, \ldots, y_n] = [(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)].$$

ただし，$\oplus$ は単位元 $\iota_\oplus$ を持つ結合的な二項演算子でなければならない．もし，f と $\oplus$ が単位時間で計算

Parallel Programming via Program Calculation.

Zhenjiang Hu, 東京大学大学院工学系研究科情報工学専攻, Department of Information Engineering, University of Tokyo.

コンピュータソフトウェア, Vol.17, No.1 (2000), pp.21–23.
[フォーラム] 1999 年 11 月 18 日受付.

できるなら，$k*$, $\oplus/$, $\oplus\#/$ と zip は，それぞれ $O(1)$, $O(\log n)$, $O(\log n)$, $O(1)$ の並列時間で計算できる．ここで，n は入力リストの長さを表す．

リスト上の再帰的関数 h は，一般的には次のように定義することができる．

$$\begin{aligned} h\ []\ c &= g_1\ c \\ h\ (a:x)\ c &= k(a,c)\ \oplus\ h\ x\ (c \otimes g_2\ a) \end{aligned}$$

h は2引数関数である．第1引数はリストであり，第2引数は計算途中の情報を蓄積するためのものである．入力リストが空の場合，関数 g_1 を c に適用し，その結果を返す．入力リストが空でない場合，そのリストの先頭要素を a，残りを x とすると，$h\ (a:x)\ c$ の結果は，非再帰部分 $k(a,c)$ と再帰部分 $h\ x\ (c \otimes g_2\ a)$ の値を用いて $\oplus$ によって計算を行う．蓄積引数は $\otimes$ と g_2 によって更新される．

逆融合運算定理とは，上に述べた h の定義の中の $\oplus$ と $\otimes$ が単位元を持つ結合的な二項演算子である時，h を次のように変換する運算規則である．

$$\begin{aligned} h\ x\ c \quad = \quad & \mathbf{let}\ cs' +\!\!+ [c'] = (c\otimes) * (\otimes\#(g_2 * x)) \\ & acs = zip\ x\ cs' \\ & \mathbf{in}\ (\oplus/\ (k * acs)) \oplus (g_1\ c') \end{aligned}$$

この変換結果は読みにくいが，その意味を理解する必要はなく，運算をする時には，この変換が持つ性質が分かっていればよい．逆融合運算定理による変換は，次の2つの性質を持つ．

- 有効な並列化プロセスである．つまり，仮に，もとのプログラム h が $O(n)$ の逐次アルゴリズムだとすると，変換された結果は $O(\log n)$ の並列プログラムになる．
- 変換された並列プログラムは work efficient の性質を持つ．つまり，一台のプロセッサ上でも逐次プログラムと同じオーダで計算が行われる．

3 逆融合運算定理の拡張

上で述べた逆融合運算定理の操作対象となる再帰関数には，蓄積引数は1つしかない．また，蓄積引数の更新部分を結合的な演算子 $\otimes$ を用いて記述することができなければ逆融合運算定理を適用することができない．一般的には，蓄積引数は複雑になる場合があるので，いくつかの蓄積引数を用いて関数を定義することがある．そのため，複数の蓄積引数を持つ再帰的関数に対する，逆融合運算規則を与える．簡単のため，2つの蓄積引数を持つ再帰定義

$$\begin{aligned} & h\ []\ c_1\ c_2 = g_1(c_1, c_2) \\ & h\ (a:x)\ c_1\ c_2 = \\ & \qquad k(a,(c_1,c_2))\ \oplus \\ & \qquad h\ x\ (c_1 \otimes_1 g_2(a,c_2))\ (c_2 \otimes_2 g_3\ a) \end{aligned}$$

を考える．ここで注意したいのは，c_1 の計算が c_2 に依存することである．

逆融合運算定理と同様，もし $\oplus$ と $\otimes_i$ が単位元を持つ結合的二項演算子であれば，h を次のような並列パタンを用いた効率のよい並列プログラムに変換することができる．

$$\begin{aligned} & h\ x\ c_1\ c_2 = \\ & \quad \mathbf{let}\ cs'_2 +\!\!+ [c'_2] = \\ & \qquad (c_2\otimes_2) * (\otimes_2\#(g_3 * x)) \\ & \qquad cs'_1 +\!\!+ [c'_1] = \\ & \qquad (c_1\otimes_1) * (\otimes_1\#(g_2 * (zip\ x\ cs'_2))) \\ & \qquad ccs = zip\ cs'_1\ cs'_2 \\ & \qquad accs = zip\ x\ ccs \\ & \quad \mathbf{in}\ (\oplus/\ (k * accs)) \oplus (g_1(c'_1, c'_2))) \end{aligned}$$

4 並列プログラミング

ここで，応用例として line-of-sight という問題を取り上げる．この問題を解く scan を用いた賢い並列アルゴリズムは [3]で与えられたが，どのようにしてそれが導かれたかについては説明されていない．

line-of-sight とは，格子点における高度が与えられている地形の地図と，ある格子点上の観察場所 X が与えられた時，各格子点上の場所が，観察場所 X から見えるかどうかを判定する問題である．具体的に言うと，観察場所 X と場所 Y を端点とする線分上の場所の高度のリストを x とすると，求めるのは，x を引数にとり，それぞれの場所が観察場所から見えるかどうかを返す関数 los である．例えば，引数として

$$[1,5,2,7,19,2]$$

が与えられると，los を適用した結果は

$$[True, True, False, False, True, False].$$

となる．

ある場所 Y は，Y と観察場所 X を端点とする線分

上の任意の場所 A について，X から Y を見る角度が，X から A を見る角度より大きい時，X から見える．そのため，ある Y が X から見えるかどうかを判定するには，その Y の高度のほかに，その Y と X の間の場所 X からの観察角度の最大値，および X との Y の間の距離の情報が必要である．それらの情報を，2 つの蓄積引数を導入して蓄積する．

$$los\ x = los'\ x\ 0\ 1$$

これは，観察角度の最大値の初期値 0 と，観察点からの距離の初期値 1 が蓄積引数に与えられていることを意味している．関数 los' の仕様は以下のように簡潔に書き表すことができる．

$$\begin{aligned}
&los'\ [\,]\ \theta\ dist = [\,]\\
&los'\ (a:x)\ \theta\ dist =\\
&\quad \textbf{let}\ \theta' = angle\ a\ dist\ \textbf{in}\\
&\quad \textbf{if}\ \theta' > \theta\ \textbf{then}\ True : los'\ x\ \theta'\ (dist+1)\\
&\qquad\qquad \textbf{else}\ False : los'\ x\ \theta\ (dist+1)
\end{aligned}$$

ここで，$angle$ は高度と距離を引数にとり，その角度を計算する関数である．

ここまでは並列に関しては何も考慮せず line-of-sight 問題を解くプログラムを記述した．以下で，逆融合運算によって，los' の並列プログラムを導出する．逆融合運算定理を適用するために，los' と h の対応づけをしなければならない．そのため，los' の再帰的な部分を 1 つにまとめ，再帰的部分と非再帰部分に分離する．if の代数的性質を用いることにより，次のような結果が得られる．

$$\begin{aligned}
&los'\ [\,]\ \theta\ dist = g_1(\theta, dist)\\
&los'\ (a:x)\ \theta\ dist =\\
&\qquad k(a,(\theta,dist))\ +\!\!+\\
&\qquad los'\ x\ (\theta \otimes_1 g_2(a,dist))\ (dist + g_3\ a)
\end{aligned}$$

ただし，

$$\begin{aligned}
&g_1(\theta, dist) = [\,]\\
&\theta \otimes_1 \theta' = \textbf{if}\ \theta' > \theta\ \textbf{then}\ \theta'\ \textbf{else}\ \theta\\
&k(a,(\theta,dist)) =\\
&\qquad \textbf{if}\ angle\ a\ dist > \theta\ \textbf{then}\ [True]\\
&\qquad\qquad\qquad \textbf{else}\ [False]\\
&g_2(a,dist) = angle\ a\ dist\\
&g_3\ a = 1.
\end{aligned}$$

これに逆融合運算定理を適用することによって，[3]の論文で記述されたものと類似の，効率のよい並列アルゴリズムが得られる．

5 おわりに

本論文は，逆融合運算定理を拡張し，その応用例を説明するとともに，運算による並列プログラミングのアイデアを述べた．詳細を知りたい読者は [5] [6]を参照されたい．

参考文献

[1] Bird, R. : An Introduction to the Theory of Lists, *Logic of Programming and Calculi of Discrete Design* (Broy, M. (ed.)), Springer-Verlag, 1987, pp. 5–42.

[2] Bird, R. : Constructive Functional Programming, *STOP Summer School on Constructive Algorithmics, Abeland*, 9, 1989.

[3] Blelloch, G. E. : Scans as Primitive Operations, *IEEE Trans. on Computers*, Vol. 38, No. 11 (1989), pp. 1526–1538.

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

[5] Hu, Z., Takeichi, M. and Chin, W. : Parallelization in Calculational Forms, *25th ACM Symposium on Principles of Programming Languages*, San Diego, California, USA, January 1998, pp. 316–328.

[6] Hu, Z., Takeichi, M. and Iwasaki, H. : Diffusion: Calculating Efficient Parallel Programs, *1999 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation*, San Antonio, Texas, BRICS Notes Series NS-99-1, January 1999, pp. 85–94.

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