並列スケルトンによる差分法計算の記述とその効率化

数理第七研究室　野村 芳明
指導教員　胡 振江 助教授

2007/02/06

1　はじめに

スケルトン並列プログラミングは並列プログラムの構築を容易にする一方で，並列スケルトンに合わせた処理手順を強要するため非効率性を招くことが多い．個々の並列スケルトンは効率的に実装されていても，それらを組み合わせた計算にはより効率的な実装が存在することがある．そこで，並列スケルトンによる計算を融合し，より効率的な実装を与える研究がなされている．[1, 2]

本研究では，ある値の計算にその近傍値のみを使うような局所的な依存関係を持った計算を対象として，並列スケルトンプログラムの融合変換による効率化手法を提案した．この計算は微分方程式の数値解法である差分法で中心となる計算である．

2　並列スケルトン

本稿で用いる並列スケルトンの定義を示す．

$$\mathsf{map}\ f\ [a_1,\ldots,a_n] = [f\ a_1,\ldots,f\ a_n]$$
$$\mathsf{zip}\ [a_1,\ldots,a_n]\ [b_1,\ldots,b_n] = [(a_1,b_1),\ldots,(a_n,b_n)]$$
$$\mathsf{shift}_l\ e\ [a_1,\ldots,a_n] = [a_2,\ldots,a_n,e]$$
$$\mathsf{shift}_r\ e\ [a_1,\ldots,a_n] = [e,a_1,\ldots,a_{n-1}]$$

3　並列スケルトンによる差分法計算の記述

本手法が対象とする計算は，漸化式

$$u_i^n = \begin{cases} \mathit{initial}\ i & n = 0 \\ \mathit{boundary}\ n\ i & i \notin [1,\ldots,N] \\ f\ (u_{i-l}^{n-1},\ldots,u_{i+r}^{n-1}) & \text{otherwise} \end{cases}$$

が与えられたとき

$$u_1^T, u_2^T, \ldots, u_N^T$$

を求める計算である．この計算は陽解法の差分法スキームを一般化したものである．

時刻 n の状態から時刻 $n+1$ の状態を求める計算は並列スケルトンを用いて

$$\begin{aligned}
&\mathit{uls}_1 = \mathsf{shift}_r\ (\mathit{boundary}\ n\ 0)\ \mathit{us}^n; \\
&\quad\vdots \\
&\mathit{uls}_l = \mathsf{shift}_r\ (\mathit{boundary}\ n\ (-l+1))\ \mathit{uls}_{l-1}; \\
&\\
&\mathit{urs}_1 = \mathsf{shift}_l\ (\mathit{boundary}\ n\ (N+1))\ \mathit{us}^n; \\
&\quad\vdots \\
&\mathit{urs}_r = \mathsf{shift}_l\ (\mathit{boundary}\ n\ (N+r))\ \mathit{urs}_{r-1}; \\
&\\
&\mathit{zs} = \mathsf{zip}_{l+1+r}\ \mathit{uls}_l\ \cdots\ \mathit{uls}_1\ \mathit{us}^n\ \mathit{urs}_1\ \cdots\ \mathit{urs}_r; \\
&\quad \textbf{where} \\
&\qquad \mathsf{zip}_2 = \mathsf{zip} \\
&\qquad \mathsf{zip}_m\ \mathit{as}\ \mathit{bs} = \mathsf{zip}_{m-1}\ (\mathsf{zip}\ \mathit{as}\ \mathit{bs}) \\
&\\
&\mathit{us}^{n+1} = \mathsf{map}\ f'\ \mathit{zs} \\
&\quad \textbf{where} \\
&\qquad f'\ (\cdots((u_{-l},u_{-l+1}),u_{-l+2}),\ldots,u_r) \\
&\qquad\quad = f\ (u_{-l},u_{-l+1},\ldots,u_r)
\end{aligned}$$

と記述できる．計算の概要を図 1 に示す．

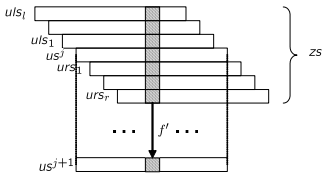

図 1　並列スケルトンによる差分法計算

このプログラムは大量の中間データを介した処理を行っているため，このままでは非常に効率が悪い．この問題は本効率化手法によって解決される．

4　融合変換による差分法プログラムの効率化

並列スケルトンの融合は，より一般的な計算を扱う拡張並列スケルトンを導入し，なされる計算を拡張並列スケルトンで表現することで行う．本手法では，拡張並列スケル

(ID-RULE)

$$as \rightarrow \mathsf{s\text{-}map}\ bound\ (strip, 0, 0)\ as$$

(MAP-RULE)

$$\mathsf{map}\ g\ (\mathsf{s\text{-}map}\ bound\ (f, j, k)\ as) \rightarrow \mathsf{s\text{-}map}\ bound\ (g \circ f, j, k)\ as$$

(SHIFTL-RULE)

$$\frac{e = f\ [b_{N+1+j}, \ldots, b_{N+1+k}],\ N = length\ as,\quad b_i = \mathbf{if}\ 1 \leq i \leq N\ \mathbf{then}\ a_i\ \mathbf{else}\ bound\ i}{\mathsf{shift}_l\ e\ (\mathsf{s\text{-}map}\ bound\ (f, j, k)\ as) \rightarrow \mathsf{s\text{-}map}\ bound\ (f, j+1, k+1)\ as}$$

(SHIFTR-RULE)

$$\frac{e = f\ [b_{0+j}, \ldots, b_{0+k}],\ N = length\ as,\quad b_i = \mathbf{if}\ 1 \leq i \leq N\ \mathbf{then}\ a_i\ \mathbf{else}\ bound\ i}{\mathsf{shift}_r\ e\ (\mathsf{s\text{-}map}\ bound\ (f, j, k)\ as) \rightarrow \mathsf{s\text{-}map}\ bound\ (f, j-1, k-1)\ as}$$

(ZIP-RULE)

$$\begin{aligned}&\mathsf{zip}\ (\mathsf{s\text{-}map}\ bound\ (f_1, j_1, k_1)\ as)\ (\mathsf{s\text{-}map}\ bound\ (f_2, j_2, k_2)\ as)\\ &\quad \rightarrow \mathsf{s\text{-}map}\ bound\ (f_3, j_1 \downarrow j_2, k_1 \uparrow k_2)\ as\\ &\qquad \mathbf{where}\\ &\qquad\quad f_3 = (f_1 \circ subarr\ j_1'\ k_1') \triangle (f_2 \circ subarr\ j_2'\ k_2')\\ &\qquad\quad j_1' = j_1 - (j_1 \downarrow j_2),\ k_1' = k_1 - (j_1 \downarrow j_2)\\ &\qquad\quad j_2' = j_2 - (j_1 \downarrow j_2),\ k_2' = k_2 - (j_1 \downarrow j_2)\end{aligned}$$

図 2　拡張並列スケルトンの融合規則

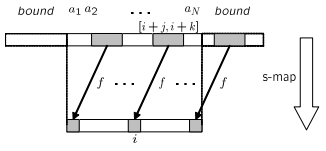

図 3　拡張並列スケルトン s-map の計算

トン

$$\begin{aligned}&\mathsf{s\text{-}map}\ bound\ (f, j, k)\ [a_1, \ldots, a_N] = [c_1, \ldots, c_N]\\ &\quad \mathbf{where}\\ &\qquad c_i = f\ [b_{i+j}, b_{i+j+1}, \ldots, b_{i+k}]\\ &\qquad b_i = \mathbf{if}\ 1 \leq i \leq N\ \mathbf{then}\ a_i\ \mathbf{else}\ bound\ i\end{aligned}$$

を導入する．なされる計算を図 3 に示す．

拡張並列スケルトン $s-map$ の融合規則を図 2 に示す．これにより前節の計算は，ひとつの計算

$$us^{n+1} = \mathsf{s\text{-}map}\ (boundary\ n)\ (f, -l, r)\ us^n$$

へと変換することができる．

この s-map の計算は，繰り返して行う場合には，通信をまとめて行いメモリアクセスの局所化をはかることでより効率的な実装を与えることができる．従って，s-map のは実装は，s-map を t 回繰り返す計算

$$\begin{aligned}&\mathsf{differential}\ 0\ boundary\ (f, j, k) = \mathrm{id}\\ &\mathsf{differential}\ t\ boundary\ (f, j, k)\\ &\quad = \mathsf{s\text{-}map}\ (boundary\ (t-1))\ (f, j, k)\\ &\qquad \circ \mathsf{differential}\ (t-1)\ boundary\ (f, j, k)\end{aligned}$$

に対して与える．

5　まとめと今後の課題

本手法により，簡潔に記述された並列スケルトンプログラムから複雑な実装を必要とする効率的なプログラムを融合変換によって得られるようになった．また，この変換を与えたことにより差分法計算をスケルトン並列プログラミングの枠組みで議論することが可能になった．

今後の課題としてはまず，扱える計算の拡大が挙げられる．多次元空間上の計算や陰解法スキームの計算などがその対象である．また，本手法とは別の抽象化を用いた差分法計算プログラムに対して，本手法が適用可能な形への変換を考えることも今後の課題である．

参考文献

[1] C. Grelck and S. B. Scholz. Merging Compositions of Array Skeletons in SAC. *Parallel Computing*, Vol. 32, No. 7+8, pp. 507–522, 2006.

[2] Z. Hu, H. Iwasaki, and M. Takeichi. An Accumulative Parallel Skeleton for All. In *Proceedings of the 11th European Symposium on Programming (ESOP' 02)*, Vol. 2305 of Lecture Notes in Computer Science, pp. 83–97. Springer-Verlag, 2002.