A-018

分散メモリマルチコンピュータによる線形及び非線形回帰演算の並列化アルゴリズム

Parallel Algorithm for Solving Liner and Nonliner Recurrence Equations on Distributed Multiple Computers

小林　寛[†]　若谷　彰良[‡]
Hiroshi Kobayashi　Akiyoshi Wakatani

1. はじめに

本稿では，三重対角行列の直接解法における線形および非線形回帰演算の並列化によるスピードアップを述べる．回帰演算はそのデータ依存性のため，並列化が難しく，並列化しても計算量が増加する等のデメリットもある．[1] 本稿では領域分割を行い，データ伝搬を先に計算し，その後各プロセッサで独立に計算するブロックイテレーションに基づく方法を考え，それを線形回帰演算だけでなく，非線形回帰演算にも適用する．

2. 回帰演算

次の対称三重対角行列の問題を考える．

$$x_0 = c_0 \tag{1}$$

$$-b_i * x_{i-1} + a_i * x_i - b_i * x_{i+1} = c_i \tag{2}$$

$$x_{N-1} = c_{N-1} \tag{3}$$

これらの式の配列 a，b，c はあらかじめ与えておく．N は要素数である．

これを並列化する手法として *CR*（*cyclic reduction*）や *RD*(*recursive doubling*)[1] が知られているが，計算量が大きく増えるという問題点がある．

そこで，補助配列 p，q を用いてガウスの消去法で式（1），(2)，(3) を直接解くと次のようになる．

$$p_0 = 0,\ q_0 = 0 \tag{4}$$

$$p_i = \frac{b_i}{a_i - b_i * p_{i-1}},\ q_i = \frac{c_i + b_i * q_{i-1}}{a_i - b_i * p_{i-1}} \tag{5}$$

$$x_i = x_{i+1} * p_i + q_i \tag{6}$$

式（5）は前進代入，式（6）は後退代入である．しかしこれらは一次回帰演算となり，データ依存により，連続的な並列化をおこなうことが難しい．すなわち，プロセッサ番号を P として，$N = P * M + 2$ というようにブロック分割すと，プロセッサ k は $k*M+1 \cdots k*M+M$ の M 個の要素を持ち，プロセッサ $(k-1)$ が $p_{(k-1)*M+M}$ を計算した後にプロセッサ k が p_{k*M+1} を計算することが出来る．

式（5），(6) は２つのタイプの回帰演算に分けられる．タイプ１は p_i のように，分母に p_{i-1} が存在する回帰演算である．(非線形回帰演算) タイプ２は q_i や，x_i のような式である．(線形回帰演算)

本稿では線形および非線形回帰演算を解く並列化アルゴリズムを提案し，元のガウス消去法と同等の計算の複雑さを目標とする．

[†]甲南大学大学院 自然科学研究科 情報・システム科学専攻
[‡]甲南大学 理工学部 情報システム工学科

3. 線形回帰演算の並列化

タイプ２の線形回帰演算は，以下の式で表すことができる．

$$w_0 = C \tag{7}$$

$$w_i = s_i * w_{i-1} + t_i \tag{8}$$

C，s，t は，あらかじめ与えておく配列である．この線形回帰演算は文献 [2]，[3] で示すように，次の３段階に分けることによりスケーラブルに並列化できる．

1）*pre − computation*
2）*propagation*
3）*determination*

まず１）*pre − computation* では，w_i を w'_i とおき，w_{k*M} を０として，すべてのプロセッサを同時に開始する．次に２）*propagation* では，w_i は０ではないので，w_i を修正するために、w_i と w'_i の差 Δw_i を以下の式のように求め，プロセッサ間で各プロセッサの初期値を計算し伝搬させる．

$$\begin{aligned}\Delta w_i &= w_i - w'_i = s_i * (w_{i-1} - w'_{i-1}) \\ &= \prod_{j=k*M+1}^{i} s_i * \Delta w_{k*M} \\ &= \alpha_i * \Delta w_{k*M}\end{aligned} \tag{9}$$

そして３）*determination* では，プロセッサ $k-1$ によって伝搬された w_{k*M} を使って，プロセッサ k の各要素を以下の式のように正す．

$$w_i = w'_i + \Delta w_i = w'_i + \alpha_i * w_{k*M} \tag{10}$$

4. 非線形回帰演算の並列化

配列 p は式（11）で示されるタイプ１の非線形回帰演算で計算される．

$$p_i = \frac{b_i}{a_i - b_i * p_{i-1}} \tag{11}$$

この非線形回帰演算は次のような手続きでブロックイテレーション法が適用できる．これを $P-scheme$ アルゴリズムとよぶ．線形回帰演算と同様な手法を適用するためにまず，p_{i-j}，p_i を補助配列 β_j，γ_j，δ_j を使って以下の式のように表す．

$$\frac{\beta_j + \gamma_j * p_i}{\delta_j + p_i} = (-1)^j * p_{i-j} \tag{12}$$

式（11）を式（12）に代入することによって，式（13）に表される関係が成立する．

$$\frac{\frac{\delta_j + (-1)^{j+1} * \frac{a_{i-j}}{b_{i-j}} * \beta_j}{\gamma_j} + \frac{1 + (-1)^{j+1} * \frac{a_{i-j}}{b_{i-j}} * p_i}{\gamma_j}}{\frac{\beta_j}{\gamma_j} + p_i} = (-1)^{j+1} * p_{i-(j+1)} \tag{13}$$

processor	No of proc.	memory
Celeron 1GHz	8	128MB
OS	MPICH	network
Linux2.4.18	1.2.4	100Mbit/sec

表 1: 実装における PC クラスタの性能

よって，β_j，γ_j，δ_j は以下のような漸化式で決定することができる．

$$\beta_1 = 1, \gamma_1 = -\frac{a_i}{b_i}, \delta_1 = 0 \tag{14}$$

$$\beta_{j+1} = \frac{1}{\gamma_j}(\delta_j + (-1)^{j+1} * \frac{a_{i-j}}{b_{i-j}} * \beta_j) \tag{15}$$

$$\gamma_{j+1} = \frac{1}{\gamma_j}(1 + (-1)^{j+1} * \frac{a_{i-j}}{b_{i-j}} * \gamma_j) \tag{16}$$

$$\delta_{j+1} = \frac{\beta_j}{\gamma_j} \tag{17}$$

ここで補助配列 β_j，γ_j，δ_j は各プロセッサで同時に計算できる．よって任意の i に対する p_i は，以下の式のように p_0 のみによって決定する．

$$p_i = \frac{-\beta_i + (-1)^i * p_0 * \delta_i}{\gamma_i - (-1)^i * p_0} \tag{18}$$

よって，β_i，γ_i，δ_i を先に計算しておくことによって，p_i は p_0 の決定後すぐに定まる．

以上より，非線形回帰演算に対する $P-scheme$ アルゴリズムは次の３段階からなる．まず，各プロセッサが同時に β_i，γ_i，δ_i の計算を行なう．(*pre − computation*) 次に p_0，β_M，γ_M，δ_M からプロセッサ０が p_M を決定し，それをプロセッサ１に送信する．受信後，プロセッサ１は p_M，β_M，γ_M，δ_M から p_{2*M} を決定する．それをプロセッサ２に送信する．プロセッサ２は，受信した p_{2*M} を使って p_{3*M} を決定する．このように，以下のプロセッサも同様にして続けていく．(*propagation*) ここで，各プロセッサは p_{k*M+1} $(i = 1 \cdots M-1)$ の計算なしで $p_{(k+1)*M}$ を決定することができる．($O(1)$ である) そして，すべてのプロセッサは受信した p_{k*M} から $p_{k*M+i}(i = 1 \cdots M-1)$ を独立かつ並列に決定することができる．(*determination*)

$P-scheme$ における *pre − computation* と *determination* は，完全に並列化できるが，*propagation* は逐次処理である．しかし，１データの伝搬なので通信コストもわずかである．($O(1)$ である) *propagation* のコストは，プロセッサ数に比例して大きくなる．よってすべての実行時間は，$O(\frac{N}{P}) + O(P) + O(\frac{N}{P})$ となる．一般的に $P \ll N$ であるので，十分大きい N に対しては *propagation* のコストは無視できる．

5. 実装および評価

$P-scheme$ アルゴリズムを表１に示した構成のクラスタシステムによって，プロセッサ数を変更して実行時間を測定する．もとの計算（式 11）の実行時間を１とおき，スピードアップを表２に示す．配列のサイズが 12800 以下の時は通信コストの割合が大きくなり，期待するスピードアップが得られない．しかし，それ以上では，*propagation* のコストは相対的に無視できるので，例えば配列のサイズが 204800 の時スピードアップは 3.62 となった．本アルゴリズムにおける計算量は，*CR* や *RD* のようにオーダが異なることはなく，同一オーダであるが，定数倍の差が生じるために，最大スピードアップは $\frac{\text{プロセッサ数}}{2.1}$ 程度となる．つまり $P-scheme$ は配列のサイズが十分大きい三重対角行列を解くことに効果がある．

	No of proc.			
SIZE	1	2	4	8
6400	1	0.66	0.74	0.83
12800	1	0.86	1.15	1.54
25600	1	0.88	1.61	2.18
51200	1	0.96	1.73	2.87
102400	1	0.96	1.83	3.28
204800	1	0.96	1.90	3.62

表 2: 配列のサイズとプロセッサ数を変更して測定したスピードアップ

6. おわりに

本稿では，三重対角行列の直接解法における線形および非線形回帰演算を並列化する $P-scheme$ アルゴリズムを提案した．これは回帰演算の並列化における問題点である領域分割における初期値の確定によるタイムロスを解消するために，線形だけでなく非線形回帰演算に対しても各プロセッサのデータ伝搬を先に計算し，初期値を各プロセッサに伝搬させた後に各要素の計算をするという手法である．また実装結果より，通信コストの問題で配列のサイズを 12800 以上にすれば十分なスピードアップが得られた．

今後の課題として，*propagation* における通信を工夫し，並列化のスピードアップを上げるということが考えられる．

参考文献

[1] 中村 素典，津田 孝夫：ベクトル計算機のための一次回帰演算の高速アルゴリズム，情報処理学会論文誌，Vol. 34, No. 3, pp. 669-680(1995)

[2] 川端 英之，湯之上 康一，津田 孝夫：ベクトル計算機のための一次回帰演算の高速アルゴリズムとその並列化，情報処理学会論文誌，Vol. 43, No. 4, pp. 971-985(2002)

[3] 川端 英之，谷口 宏美，津田 孝夫：バルク回帰並列処理：依存のあるループ並列実行方式，情報処理学会論文誌，Vol. 42, No. SIG 12, pp. 111-123(2001)