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

川端英之[†] 湯之上康一[†,☆] 津田孝夫[†]

本論文では，線形一次回帰演算のベクトル計算機向け高速化手法を提案する．この手法はベクトルレジスタを有効に利用する方法で，余分なベクトルロードおよびストア命令を排除するとともにスカラ処理が必要となる部分を削減することによって高速性能を実現する．従来，線形回帰演算のベクトル処理手法としては，計算量やベクトル長の観点から，ループアンローリングに基づく手法（アンローリング手法）が有効であるとされてきた．商用ベクトル計算機を単純化したモデルによる比較では，我々の手法はわずかなコード記述量であるにもかかわらず理想的なアンローリング段数（一般には非常に大きい段数）によるアンローリング手法と同等の演算性能であることが分かった．NEC SX-4 を用いた線形回帰演算の実測では，モデルによる予測結果どおり理論最大性能の 1/4 に達する演算速度が観測され，モデルの妥当性と提案手法の有効性がいずれも高いことが確認された．また，提案手法の自然な拡張による並列化の試みでは，8 プロセッサを用いて 5.9 倍の台数効果を得た．

Fast Execution of Linear Recurrences on Vector Computers and Its Parallelization

Hideyuki Kawabata,[†] Kouichi Yunoue[†,☆] and Takao Tsuda[†]

In this paper, we propose a new technique to calculate fast first-order linear recurrences on vector computers. Our method is intended to reduce redundant vector load and/or store operations. The objective is accomplished by restructuring recurrent loops with information such as the length of vector registers in mind, not relying on automatic strip-mining by vectorizing compilers. Although it has been said that unrolling approach is suitable for linear recurrences to run fast on vector computers, appropriateness of the unrolling factor hasn't been mentioned specifically in general. We construct a simple model of a vector machine to compare ordinary unrolled code and code created by our method. On the model, it is explained that codes by our method can perform as well as unrolled codes with optimal (typically unrealistically large) unrolling factors. Experimental results on the NEC SX-4 show that our code runs 1/4 times as fast as the peak performance of the machine, which is also predicted by the model. The results confirm that our method is efficient and our model of a vector processor is proper. In addition, we show that our method can be easily parallelized. Speedup ratio of 5.9 on 8 processors has been observed.

1. はじめに

線形一次回帰演算をはじめとする回帰演算は，科学技術計算コード中に頻繁に現れるため，ベクトル化や並列化による高速処理方式の開発が求められ続けている．本論文は，ベクトル計算機を対象とした線形一次回帰演算の高速化手法を提案するものである．並列ベクトル計算機のための計算方式もあわせて提案する．

線形一次回帰演算（以下，単に回帰演算と呼ぶ）は**図 1** に示すループで表現できる．図中 N は回帰演算によって算出する要素数である．図から明らかなように，ループの繰返しにまたがる回帰的なデータ参照による依存関係（ループ運搬フロー依存）が存在しているため，a_i の値を求める計算は個々の i についてまったく独立には実行できない．そこで通常は計算順序の変更による並列性の抽出に基づく高速化手法が検討される．

よく知られているのは巡回縮約法（cyclic reduction）や再帰二倍化法（recursive doubling）などを適用して計算処理中の並列性を高める方法である．こ

† 広島市立大学情報科学部
Faculty of Information Sciences, Hiroshima City University
☆ 現在，広島市立大学大学院情報科学研究科
Presently with Graduate School of Information Sciences, Hiroshima City University

```
real*8 a(0:N)
  ...
do i = 1, N
    a_i = p_i · a_{i-1} + q_i
enddo
```

図 1 線形一次回帰演算
Fig. 1 First order linear recurrence.

れら，回帰演算を並列 prefix 計算[1)~8)]と見なす方法の多くは，各時刻において並列処理可能な計算をすべて同時に実行することを 1 計算ステップと見なして，算出する要素数 N に対して要する計算ステップ数を削減（典型的には $O(N)$ ステップから $O(\log N)$ ステップに削減）する手法である．しかしながらこれらの手法の多くは総計算量を増加（たとえば $O(N \log N)$ に増加）させるため，多数の同時動作可能な演算装置を備えた環境でなければ高速化につながらない[9)]．実際，ベクトル計算機には向かないものも多い[10)]．

これに対し，スーパスカラや VLIW などを対象として，命令レベル並列処理機能を十分に使い切ることができるように演算量の増加を抑えたプログラム変換を行う手法が考案されている[11),12)]．この方法は，演算パイプラインの拡張による回帰演算専用ハードウェア（ベクトルマクロ命令）のアプローチ[13)]と同等な考え方である．ただしベクトルマクロ命令による高速化の度合いは高くない．

一方ベクトル計算機は，その実効性能の高さから，科学技術計算において重要視され続けている．ベクトル計算機による回帰演算の高速化についても，総計算量の増加分をベクトル処理による高速化によって吸収すべく，いくつかのアルゴリズムが考案されている．なかでも，ループアンローリング（ブロック化）に基づく手法（以降，単にアンローリング手法と呼ぶ）が有効であるとの報告がある[10),14)]．しかしながら，アンローリング段数がどの程度であるべきかなどの定量的な指摘はあまりなく，ベクトルレジスタ長との関連についての言及はほとんどない．

本論文では，ベクトル計算機を対象とする高速な線形一次回帰演算処理手法を提案する．提案手法はアンローリング手法の一般化に相当するが，(1) ベクトルレジスタ長を意識して無駄のないベクトル処理に留意し，(2) ベクトルロード/ストアの数を低減するために自動ベクトル化コンパイラによるストリップマイニング[15)]の適用を受けないように配慮し，(3) スカラ処理に要する時間をできるだけ短縮するよう配慮した方法である．アンローリング手法の欠点の 1 つはコード量の増大で，アンローリング段数にほぼ比例した量のコードを付加する必要がある．これに対して我々の方法は，コードを簡潔に記述できるうえに，適切な段数でのアンローリングを施したものと同等な演算速度で処理可能である．またコードは高級言語で記述できる．

本論文では，提案する手法の有効性の評価のために，対象とするベクトル計算機の簡単なモデルを提示し，それに基づく理論的評価を行う．さらに商用ベクトル計算機を用いた実測結果に基づく評価を示す．NEC SX-4 を用いた実測では，約 30,000 要素を算出する線形一次回帰演算が，SX-4 の理論最大性能の約 1/4 の速度，すなわち 500 MFLOPS で行われた．より多数の要素を求める場合にはさらに 1 割以上高速化できた．これは実際に行った総計算量から見積もられる理論性能に近い演算速度であり，我々の手法が無駄のない効率的なものであることを示している．

なお，本論文で提案する手法は，スカラ命令やベクトルマクロ命令と比較してベクトル命令が大幅に高速である環境を想定するものであり，ベクトルレジスタを持つ通常のベクトル計算機を対象としている．類似の名前の擬似ベクトル処理[16)]は，データ参照パターンがあらかじめ把握できるときにキャッシュミスを起こさせない仕組みであり，回帰参照を含むループも依存のないループと同程度の速度で実行される．擬似ベクトル処理機能を持つ計算機は，本論文では対象としない．

以下，まず 2 章では，線形回帰演算の並列性について概観し，従来考案されてきたベクトル計算機向けアルゴリズムについて述べる．3 章では，各種手法の理論的評価のために用いる簡単なベクトル計算機のモデルについて述べ，それを用いてアンローリング手法の欠点を概説する．4 章では，本論文で新たに提案する線形一次回帰演算のベクトル計算機向けの高速処理手法について述べる．5 章では，提案手法の自然な拡張としてまとめられる並列ベクトル計算機向け処理手法について述べる．6 章では実測評価に基づく考察を行い，今後の課題をまとめる．

2. 準備：回帰演算のベクトル処理

2.1 回帰演算における並列性

線形一次回帰演算は図 1 に示すループで表現できる．ここで $p_i, q_i (i = 1, 2, \ldots, N)$ はループ不変変数である．このループの各繰返し（イタレーション）間には依存距離 1 のループ運搬フロー依存があり，a_i の値を求める計算は個々の i についてまったく独立には実行できない．つまりそのままでは単純なベクトル化はできない．しかし，

```
   do i = 1, N, 5
* – step1 ————————————
      v = q_i
      u = p_i
      v = p_{i+1}· v + q_{i+1}
      u = p_{i+1}· u
      v = p_{i+2}· v + q_{i+2}
      u = p_{i+2}· u
      v = p_{i+3}· v + q_{i+3}
      u = p_{i+3}· u
      v = p_{i+4}· v + q_{i+4}
      u = p_{i+4}· u
* – step2 ————————————
      a_{i+4} = u ·a_{i-1} + v
   enddo
   do i = 1, N, 5
* – step3 ————————————
      a_i = p_i · a_{i-1} + q_i
      a_{i+1} = p_{i+1} · a_i + q_{i+1}
      a_{i+2} = p_{i+2} · a_{i+1} + q_{i+2}
      a_{i+3} = p_{i+3} · a_{i+2} + q_{i+3}
   enddo
```

図 2 アンローリング手法を適用した線形一次回帰演算コード
Fig. 2 Unrolled first-order linear recurrence.

$$\vec{a}_i \equiv \begin{pmatrix} a_i \\ 1 \end{pmatrix}, \quad P_i \equiv \begin{pmatrix} p_i & q_i \\ 0 & 1 \end{pmatrix}$$

とおくと，その回帰の様子は以下のように表せる．

$$\begin{aligned} \vec{a}_i &= P_i \cdot \vec{a}_{i-1} \\ &= P_i P_{i-1} \cdots P_1 \cdot \vec{a}_0. \end{aligned} \tag{1}$$

行列積は結合法則を満たす二項演算であり，式 (1) は図 1 の回帰演算が prefix 計算であることを示している．すなわち，中間積の計算順序の任意性から，一次回帰演算には分割統治法や巡回縮約法などの様々な並列 prefix アルゴリズムが適用できることが分かる．

しかしながら多くの並列 prefix アルゴリズムは，多数個のプロセッサの存在を仮定して計算ステップ数を削減しようとするものであり，一般に総計算量を増大させる．小規模構成の並列計算機やベクトル計算機では，高速化につながらない場合も多い[9]．

2.2 ベクトル計算機による回帰演算の高速処理

ここではベクトル計算機における回帰演算の既存の高速化手法について述べる．文献 10) には各手法についてその得失が実測データとともに示されている．そこでの指摘の要点は以下のとおりである．

- スカラ実行の含まれる割合が少ない方が有効．
- 大きなベクトル長でのベクトル実行が有効．
- 演算量を増大させないアルゴリズムが有効．

これらを総合して，計算量の増加量が少ないループアンローリング手法[14]の有効性が示され，その弱点を補う方法として，単純なベクトル処理のできない部分（スカラ処理部分）のステップ数削減のために逐次型巡回縮約を融合する手法が提案されている[10]．

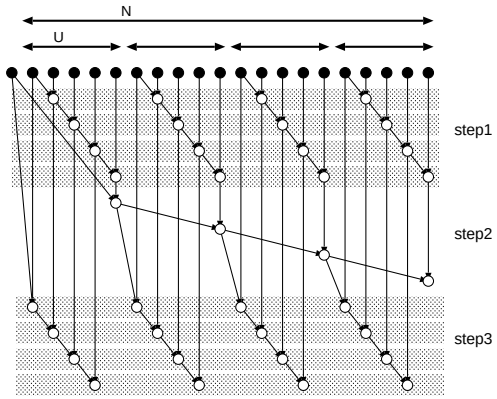


図 3 アンローリング手法のデータフロー
Fig. 3 Dataflow of unrolled linear recurrence.

アンローリング手法による回帰演算コードを**図 2** に示す（この手法の理論的な評価については次章でも触れる）．図 2 はアンローリング段数が 5 の場合の例である☆．図中 step1 および step3 の部分は，各々ループ分割を適用してベクトル化できる．その計算の進行の様子（データフローの概略）を**図 3** に示す．図中の黒丸は定数参照を，白丸は計算処理を表し，丸から丸への有向エッジはデータの参照関係を表す．step1 においては 1 つの白丸が乗算 2 つおよび加算 1 つを意味し，step3 においては乗算 1 つと加算 1 つに対応している．白丸から出るエッジは，そこでの計算結果が他の計算に引用されることを表している．横長の網掛けは，それぞれが覆っている計算がまとめてベクトル化できることを示している．

図 3 から分かるとおり，全体の処理中のほとんどの計算はベクトル処理できるが，スカラ処理の必要なデータフローの経路も存在する（step2）．このためアンローリング段数（図 3 の U）が小さい場合には，アンロール後のループの繰返し回数，すなわちベクトル処理可能部分のベクトル長 $\lfloor N/U \rfloor$ が大きいものの，スカラ処理時間がネックとなり，演算速度は低く抑えられてしまう．ベクトルマクロ命令[13]を持つ計算機では，それによって step2 を高速化できるが，ベクトルマクロ命令では通常のベクトル命令ほどの高速化率は得られないため，依然として step2 にかかる時間が問題となる．この点については次章で触れる．

なお本論文では，ベクトルマクロ命令の適用の余地

☆ 図 2 は 2 つのループで記述しているが，ループ融合をしても計算内容は変わらない．また図 2 では N が 5 の倍数であることを仮定している．

```
do i = 1, N, V_H
    VLD    VR1, p_i
    VLD    VR2, q_i
    VIMA   VR3, a_{i-1}, VR1, VR2
    VST    VR3, a_i
enddo
```

図 4 図 1 のループの単純なベクトル化例
Fig. 4 Simple vector code for Fig. 1.

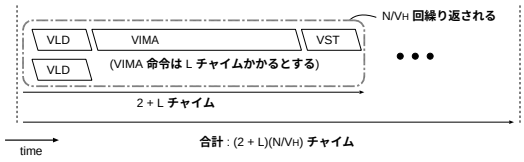


図 5 図 4 のコードの実行の様子
Fig. 5 Execution of the code of Fig. 4.

があっても通常のベクトル命令で置き換えられないコード片はすべて「スカラ処理部分」と呼ぶ．たとえば step2 はスカラ処理部分である．

3. ベクトル計算機のモデル化によるアンローリング手法の評価

本章では，ベクトルアルゴリズムを理論的に評価するためのベクトル計算機のモデル化と，そのモデル上での一次回帰演算のアンローリング手法の評価について，述べる．

3.1 ベクトル計算機のモデル化

次のようなベクトル計算機のモデル $\mathcal{V}$ を想定する．

- ベクトルレジスタ長は一定値 V_H である．
- ベクトルロード/ストアパイプラインを 2 本持ち，同時動作可能である．
- ベクトル乗算パイプラインおよび加算パイプラインを 1 本ずつ持ち，同時動作可能である．
- ベクトルロード/ストアにおける連続アクセスとストライドアクセスの差は微小で無視できる．
- ベクトル命令は，ロード VLD，ストア VST，乗算 VFMP，加算 VFAD，および，一次回帰演算のベクトルマクロ命令 VIMA を持つ．VIMA は他の命令と同時には実行できないものとする．
- ベクトル長 $V(\leq V_H)$ による 1 つのベクトル命令の実行にかかる時間はどれも等しく，V についての一次関数 $t_c(V) \equiv t_s + t_p \cdot V$ であるとする．ここで t_s および t_p は定数とする．VIMA 命令のみ例外的に $L(>1)$ 倍の時間（すなわち $L \cdot t_c(V)$）がかかるとする．

モデル $\mathcal{V}$ は，たとえば NEC SX-4 のアーキテクチャを単純化したものと見ることもできる[18)]☆．モデル $\mathcal{V}$ ではベクトルレジスタ数の指定をしていないが，適当な数のベクトルレジスタが利用可能であるとする．

ここでモデル $\mathcal{V}$ 上での一次回帰演算の実行を考える．まず，図 1 に対して単純なベクトル化を行った場合のモデル $\mathcal{V}$ 用のベクトル命令列を**図 4** に示す．図中の “VR” で始まる記号はベクトルレジスタを表す．図 4 に示されるように，図 1 の計算はベクトルレジスタ長 V_H ごとにストリップマイニングされて実行される（簡単のため N は V_H の倍数であるとする）．また図 4 のコードのモデル $\mathcal{V}$ 上での実行の様子は**図 5** のようになる．図 5 は横軸が時間を表し，菱形の 1 つ

☆ 同時動作可能なベクトルロード命令とベクトル演算命令の数の比が 1:1 である点や，ストライドアクセスが連続アクセスと同等な速度で行われる点，一次回帰演算のベクトルマクロ命令を持つ点などが類似している．

```
do i = 1, N, 5·V_H
# – step1 ————————————————
    VLD    VR00, q_i                  # stride 5
    VLD    VR01, p_i                  # stride 5

    VLD    VR12, p_{i+1}              # stride 5
    VFMP   VR13, VR12, VR00
    VLD    VR14, q_{i+1}              # stride 5
    VFAD   VR15, VR13, VR14
    VFMP   VR16, VR13, VR01
    ...
    VLD    VR42, p_{i+4}              # stride 5
    VFMP   VR43, VR42, VR35
    VLD    VR44, q_{i+4}              # stride 5
    VFAD   VR45, VR43, VR44
    VFMP   VR46, VR43, VR36

# – step2 ————————————————
    VIMA   VR50, a_{i-1}, VR45, VR46  # stride 5
    VST    VR50, a_{i+4}              # stride 5
enddo
do i = 1, N, 5·V_H
# – step3 ————————————————
    VLD    VR00, a_{i-1}              # stride 5

    VLD    VR10, p_i                  # stride 5
    VFMP   VR11, VR10, VR00
    VLD    VR12, q_i                  # stride 5
    VFAD   VR13, VR11, VR12
    VST    VR13, a_i                  # stride 5
    ...
    VLD    VR40, p_{i+3}              # stride 5
    VFMP   VR41, VR40, VR33
    VLD    VR42, q_{i+3}              # stride 5
    VFAD   VR43, VR41, VR42
    VST    VR43, a_{i+3}              # stride 5
enddo
```

図 6 ベクトル化されたアンローリング手法によるコード
Fig. 6 Vectorized unrolled code.

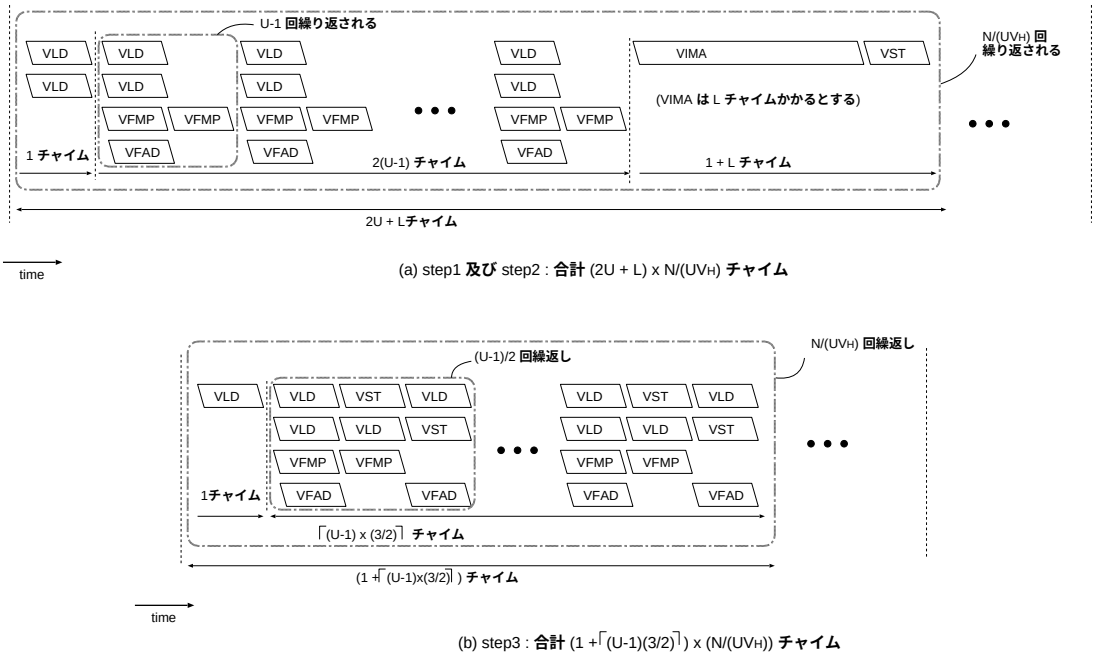


図 7 図 6 のコードの実行の様子
Fig. 7 Execution of the code of Fig. 6.

1 つがベクトル命令の実行を表している☆.

さて，1 つのベクトル命令の実行にかかる時間を「チャイム」と呼ぶ[14),17)]ことにし，全体の処理時間をチャイム単位で考える．独立に同時に（あるいはチェイニングによりほぼ同時に）実行可能なベクトル命令列があれば，1 チャイムで複数の処理が行われることになる．

図 1 を単純に図 4 のようにベクトル化した場合に実行に必要となるチャイム数 c_{org} は，図 5 より，次のようになることが分かる：

$$c_{org} = \frac{N}{V_H} \cdot (L+2).$$

これより，図 4 のコードの実行に要する時間 t_{org} は

$$\begin{aligned} t_{org} &= c_{org} \cdot t_c(V_H) \\ &= \frac{N}{V_H} \cdot (L+2) \cdot (t_s + t_p V_H) \end{aligned} \qquad (2)$$

となる．

3.2 アンローリング手法の効果について

次に，図 1 のループを図 2 のように記述しなおすことによる効果を検討する．

図 2 のコードに対してモデル $\mathcal{V}$ 用にベクトル化を施すと**図 6** のコードが得られる．このコードがモデル $\mathcal{V}$ 上で実行される様子は**図 7** のようになる．図 7 は，各ベクトル命令が実行されるタイミングを厳密に表しているわけではないが，特徴的な点を見てとることができる．たとえば図 7 (a) は連続したベクトル乗算がクリティカルパスになっていてロード/ストアパイプラインの稼働率が悪いことを示しているし，図 7 (b) では逆にロード/ストアパイプラインがネックになってベクトル演算器において待ち状態が生じていることが分かる．

図 7 から，アンローリング段数 U におけるベクトル実行時のチャイム数 c_{unrl} と実行時間 t_{unrl} が求められる．まず $N \geq UV_H$ の場合はおよそ次のとおり：

$$\begin{aligned} c_{unrl} &= \frac{(2U+L) \cdot N}{UV_H} \\ &\quad + \left(1 + \left\lceil \frac{3(U-1)}{2} \right\rceil\right) \cdot \frac{N}{UV_H} \\ &= \frac{N}{V_H} \cdot \left(\frac{7}{2} + \frac{L}{U} - \frac{1}{2U}\right) \\ &\qquad (N \geq UV_H \text{ のとき}), \\ t_{unrl} &= c_{unrl} \cdot t_c(V_H) \\ &= \frac{N}{V_H} \cdot \left(\frac{7}{2} + \frac{L}{U} - \frac{1}{2U}\right) \cdot t_c(V_H) \\ &\qquad (N \geq UV_H \text{ のとき}). \end{aligned} \qquad (3)$$

なお簡単のため U を奇数とする．$N < UV_H$ である場合には，ベクトル処理時のベクトル長が V_H ではな

☆ 図 5 では VIMA を他のベクトル命令と同じ高さの菱形で描いているが，VIMA のスタートアップ時間が他のベクトル命令と同程度であることを仮定しているわけではない．

く N/U となり，1チャイムの計算に要する時間が変わるので，c_{unrl} と t_{unrl} はおよそ次のようになる：

$$c_{unrl} = \frac{7U}{2} + L - \frac{1}{2} \quad (N < UV_H \text{ のとき}),$$
$$t_{unrl} = \left(\frac{7U}{2} + L - \frac{1}{2}\right) \cdot t_c \left(\frac{N}{U}\right) \quad (N < UV_H \text{ のとき}). \tag{4}$$

式 (2) と式 (3) との比較で明らかなように，線形一次回帰演算（図1）に対するアンローリング手法によるベクトル化（図2）の有効性は，L を含む項 NL/V_H の係数が $1/U$ 倍になることによる．すなわち，アンローリングの適用はスカラ実行にかかる時間の大幅な削減につながっている．

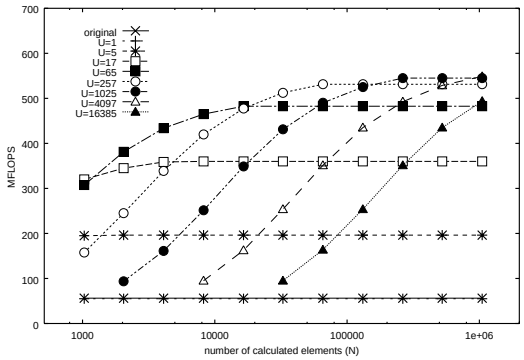
ここで，アンローリング段数を変化させた場合の図6のコードの処理速度の変化を見るため，次のようにパラメータを指定し，アンローリング段数 U を変化させ，演算速度 $2N/t_{unrl}$ を（単位を MFLOPS にして）プロットした様子を**図8**に示す：

$V_H = 256, t_s = 10\times10^{-9}$[秒], $t_p = 1\times10^{-9}$[秒].

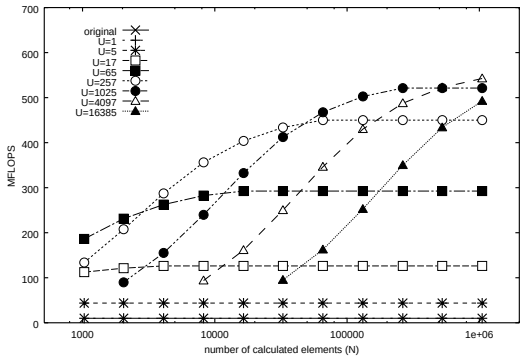
図8の横軸は N の値で，縦軸は演算速度である．各 U の値に対して $U < N$ の範囲で N の値を変化させて（いずれも $256(= V_H)$ の倍数）演算速度を求めた．図8(a)は $L = 32$ の場合で，ベクトルマクロ命令を装備したベクトル計算機での実行を想定している．一方図8(b)は，$L = 200$，すなわちベクトルマクロ命令が非常に遅い場合の見積りであるが，近似的にはベクトルマクロ命令がない環境での実行を表すと見なせる．

図8から，アンローリング段数 U の値によって演算速度の極限値が抑えられる様子が明確に分かる．また，N の値を固定したときの各曲線の示す演算速度を比較すると，U が小さい範囲（$N > UV_H$）では U の増加は演算速度の増加につながるが，U が大きくなり $N < UV_H$ となると，U が増加するにつれて演算速度は低下している．図8(a)と(b)の対比では，U が小さい場合，すなわちスカラ処理部が大きい場合，スカラ処理部の処理速度が全体の処理速度に大きく影響を与えている様子が分かる．

アンローリング段数 U を大きくすることはベクトルコード量の大幅な増大につながる☆ので，無制限なアンローリングの適用は現実的ではない．しかしながら文献10)，14)では，N が大きい場合であっても小さいアンローリング段数（たかだか11）しか考慮していない．図8によると，11段程度のアンローリング段数ではベクトル計算機の能力を引き出せないといえる．



(a) with fast vector-macro operations ($L = 32$)



(b) with slow vector-macro operations ($L = 200$)

図8 モデル $\mathcal{V}$ 上でのアンローリング段数と演算性能の関係
Fig. 8 Theoretical estimation of unrolling on model $\mathcal{V}$.

これに対し，次章で述べる高速な回帰演算処理手法では，コード量の増大をともなわずにスカラ実行部分の計算量を抑えて，理想的な段数でのアンローリングを適用したものと同等な効果を得ることができる．

4. 一次回帰演算の高速ベクトルアルゴリズム

4.1 本手法の概要

提案する回帰演算アルゴリズムに基づくコードの骨組みを**図9**に示す．図9(a)はその全容で，アンローリング手法と同様，step1 ～ step3 の部分からなる．図中のパラメータ V および B の決定方法は4.3節で述べるが，およそ $N = B\cdot V$ である．step1，step3における二重ループの内側ループおよび外側ループを，以降では単にそれぞれ「内側ループ」および「外側ループ」と呼ぶ．u，v，w は作業用の配列である☆☆．step3は作業用配列 w を排除して図9(b)のように記述する

☆ 図1と図2の比較で分かるとおり，アンローリング段数 U に対して $3\cdot(U-1)$ 個のステートメントが追加される．
☆☆ 配列 w は，u あるいは v を再利用してもかまわない．

```
* – step1 ————————————
  do i = 1, V
     v_i = q_{1+(i−1)·B}
     u_i = p_{1+(i−1)·B}
  enddo
  do j = 1, B − 1
     do i = 1, V
        v_i = p_{1+j+(i−1)·B} · v_i + q_{1+j+(i−1)·B}
        u_i = p_{1+j+(i−1)·B} · u_i
     enddo
  enddo
* – step2 ————————————
  do i = 1, V
     a_{i·B} = u_i · a_{(i−1)·B} + v_i
  enddo
* – step3 ————————————
  do i = 1, V
     w_i = a_{(i−1)·B}
  enddo
  do j = 0, B − 2
     do i = 1, V
        w_i = p_{1+j+(i−1)·B} · w_i + q_{1+j+(i−1)·B}
        a_{1+j+(i−1)·B} = w_i
     enddo
  enddo
```

(a) コードの全容

```
* – step3 ————————————
  do j = 0, B − 2
     do i = 1, V
        a_{1+j+(i−1)·B} = p_{1+j+(i−1)·B} · a_{j+(i−1)·B}
                          +q_{1+j+(i−1)·B}
     enddo
  enddo
```

(b) 作業用配列 w を用いない場合の step3

図 9 提案する線形一次回帰演算の高速アルゴリズム
Fig. 9 A fast algorithm for first-order linear recurrence.

ことももできる☆.

図 9 (a) は実質的には図 2 のループを任意の段数 B でのアンローリングができるように書き換えたものと見ることもできる．ただし，本手法の提案内容は，

(1) 図 9 (a) において V の値がベクトルレジスタ長以下であること（$V \leq V_H$），

(2) B の値が奇数であること，および

(3) step1 および step3 で導入している作業用配列がベクトルレジスタに割り付けられるようなコンパイルが行われること，

を制約条件として課すものである．データフローの概略は図 3 に示すものと同等となるが，各々の網掛け部

☆ 配列 w の導入は step3 の処理内容に変化をもたらしてはいないが，図 9 (b) の記述では，自動ベクトル化コンパイラが step3 の内側ループを我々の意図どおりにコンパイルしない例（不必要なベクトルロード/ストアの挿入など）があった．

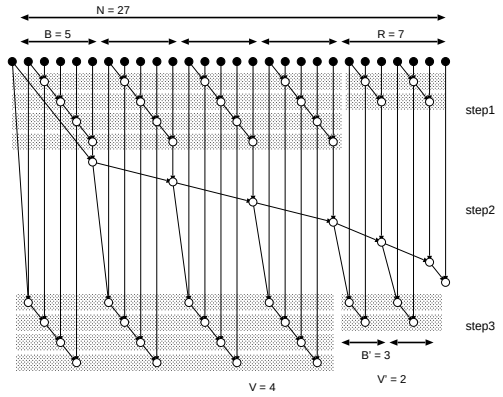


図 10 提案する高速アルゴリズムのデータフロー
Fig. 10 Dataflow of the fast algorithm.

のベクトル演算におけるベクトル長 V はベクトルレジスタ長 V_H を超えないので，N の値によらず，ストリップマイニングや余分な主記憶アクセスは行われない．またスカラ実行部分（step2）の回帰ループの繰返し回数もつねに V_H 以下になる．これらは上記の制約 (1) および (3) による．制約 (2) は，step1 および step3 における内側ループ内での配列要素のストライドアクセスがバンク/ポートコンフリクトを起こしにくくするためのものである．近年のベクトル計算機では連続アクセスとストライドアクセスでほとんどデータ転送速度に差が出なくなりつつある[18]し，モデル $\mathcal{V}$ もそれをふまえているが，制約 (2) は容易に満たせるうえに本手法の性能可搬性を高めるために効果があると思われるので，付加した．

step1 および step3 中の外側ループに対しては，任意段数のアンローリングを適用することもできる．後に実測結果を示すが，step1 および step3 中の外側ループのアンローリングはロードストアおよび演算パイプラインの稼働率を高めるのに有効である．本論文では，アンローリング手法（図 2）との混乱を避けるために，step1 および step3 中の外側ループへのアンローリングの適用操作をポストアンローリングと呼ぶ．

図 10 に，本手法でのベクトル処理の流れを示す．図中 R は，N が V の倍数でない場合の端部分である．この部分に関する後処理は図 9 のコード中の各 step の末尾で行う必要があるが，計算全体に占める割合がわずかになるように抑えることはできる．step1 および step3 の後処理では，図 10 のようにあらためてベクトル長 V' にてベクトル処理を行うか，スカラ処理を行うことになる．

4.2 アンローリング手法との比較

図 9 (a) に示すコードは，3.2 節で示したベクトル計

```
# – step1 ————————————————
  VLD    VR0, q_1                  # stride N/V
  VLD    VR1, p_1                  # stride N/V
  VST†   VR0, v                    # stride N/V
  VST†   VR1, u                    # stride N/V
  do j = 1, N/V
     VLD    VR2, p_{j+1}           # stride N/V
     VFMP   VR3, VR2, VR1
     VLD    VR4, q_{j+1}           # stride N/V
     VFAD   VR0, VR3, VR4
     VFMP   VR1, VR2, VR1
  enddo
  VST‡   VR0, v                    # stride N/V
  VST‡   VR1, u                    # stride N/V
# – step2 ————————————————
  VLD‡   VR0, v                    # stride N/V
  VLD‡   VR1, u                    # stride N/V
  VIMA   VR5, a_0, VR1, VR0
  VST    VR5, a_{i+N/V}
# – step3 ————————————————
  VLD    VR6, a_0                  # stride N/V
  VST†   VR6, w                    # stride N/V
  do j = 0, N/V − 1
     VLD    VR7, p_{j+1}           # stride N/V
     VFMP   VR8, VR7, VR6
     VLD    VR9, q_{j+1}           # stride N/V
     VFAD   VR6, VR9, VR8
     VST    VR6, a_{j+1}           # stride N/V
  enddo
  VST†   VR6, w                    # stride N/V
```

† および ‡ を付加した命令は，本来は不必要．

図 11 図 9 のコードをベクトル化した様子
Fig. 11 Vectorized code for Fig. 9.

算機のモデル $\mathcal{V}$ を対象とすると**図 11** のようにベクトル化される☆．**図 12** は図 11 のベクトル実行の様子である．この図から，図 11 のコードによるベクトル実行にかかるチャイム数 c_{fast} は，次のようになる：

$$\begin{aligned} c_{fast} &= \left(\frac{2N}{V}+L+2\right)+\left(1+\left\lceil\left(\frac{N}{V}-1\right)\cdot\frac{3}{2}\right\rceil\right) \\ &= \frac{7N}{2V}+L+\frac{3}{2} \quad (V \le V_H \text{ のとき}). \end{aligned}$$

本手法の実行に要する時間 t_{fast} は：

$$\begin{aligned} t_{fast} &= c_{fast}\cdot t_c(V) \\ &= \left(\frac{7N}{2V}+L+\frac{3}{2}\right)\cdot t_c(V) \\ &\qquad (V \le V_H \text{ のとき}). \end{aligned} \tag{5}$$

☆ 図 11 中，† および ‡ を付加した命令は本来は不必要であるが，NEC SX-4 における実測（後述）に用いた自動ベクトル化コンパイラはこれらに相当する命令を挿入するようであった．なお，† および ‡ を付加した命令に必要な総実行時間はわずかである．

式 (5) から，本手法で達成できる最大性能を見積もることができる．N を大きくすると，$2N/t_{fast}$ の値は $4V/(7\cdot t_c(V))$ に近づく．一方でモデル $\mathcal{V}$ の理論最大性能は $2V_H/t_c(V_H)$ である．つまり，ベクトル長 V を V_H に保って実行できたとすれば，本手法による演算速度は，理論最大性能の $\{4V_H/(7t_c(V_H))\}/\{2V_H/t_c(V_H)\} = 2/7$ 倍となる．

ここで，あるベクトルレジスタ長を仮定してパラメータ V を設定したコードを $V > V_H$ であるようなベクトル計算機で実行する場合についても考えておく（式 (5) は，$V \le V_H$ の範囲においてのみ有効である）．$V > V_H$ の状況で我々の手法によって生成したコードを実行する場合には，自動並列化コンパイラによるストリップマイニングが行われ，コード全体が V/V_H 回実行されることになるとともに，いくつかのベクトルロード/ストアが図 11 の step1 および step3 のループの内部で行われるようになる．その実行の様子は**図 13** のとおりで，図中に斜線を記した菱形に対応する命令，つまり step1 での VST 2 つと VLD 2 つ，step3 での VLD 1 つが，ストリップマイニングの適用の影響で新たに加わる命令である．これらはループ中で繰り返し実行されるため，総実行時間に大きく影響する．図 13 から，総実行時間は次のようになる（簡単のため V は V_H を割り切るとする）：

$$\begin{aligned} c_{fast} &= \frac{3N+LV+V}{V_H}+\frac{2N-V}{V_H} \\ &= \frac{5N+LV}{V_H} \quad (V > V_H \text{ のとき}), \\ t_{fast} &= \frac{5N+LV}{V_H}\cdot t_c(V_H) \\ &\qquad (V > V_H \text{ のとき}). \end{aligned} \tag{6}$$

ここで，式 (2)，(3)，(4)，(5)，(6) に含まれるパラメータについて，

$V_H = 256$, $L = 32$, $U = 11$,
$t_s = 10\times10^{-9}$[秒], $t_p = 1\times10^{-9}$[秒]

とし，N 個の要素を求める一次回帰演算（計算量 $2N$ に相当）を実行した場合の各手法の演算速度を図示すると，**図 14** のようになる．図中，`original` は式 (2) に，`unroll11` は式 (3) と (4) に基づくもので，その他は式 (5) と (6) から算出したものである．本論文で提案する手法に基づくコードはベクトルレジスタ長に基づいて他のパラメータを決定するので，ベクトルレジスタ長が 128，256，512，1024，2048 であることを想定したコードをそれぞれ `vr128`，`vr256`，`vr512`，`vr1024`，`vr2048` として区別して示している．図 14 の理論値は，各コードが「想定」しているベクトルレジスタ長を無視して $V_H = 256$ のベクトル計算機上で

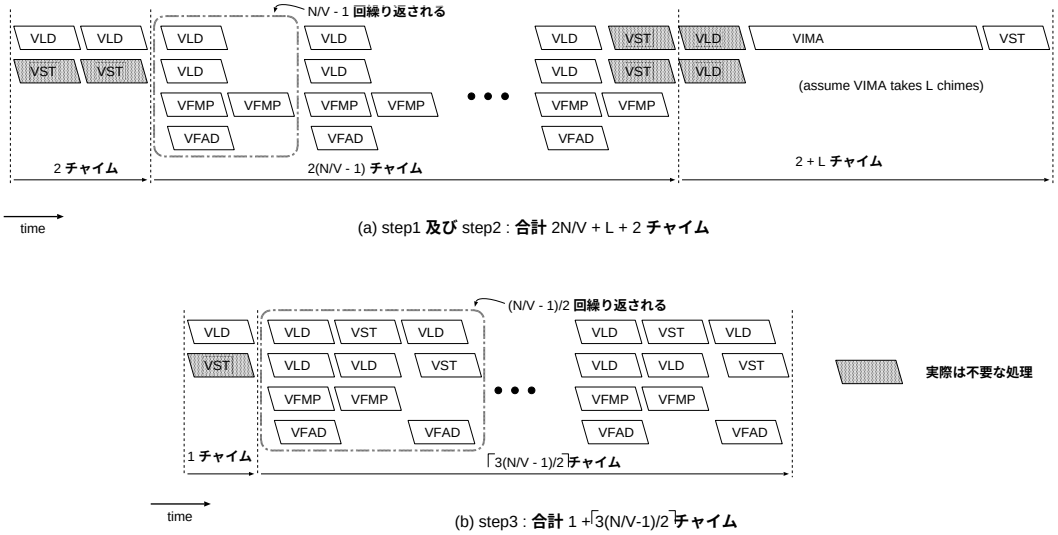


図 12 提案手法によるコード（図 11）の実行の様子
Fig. 12 Execution of the proposed code of Fig. 11.

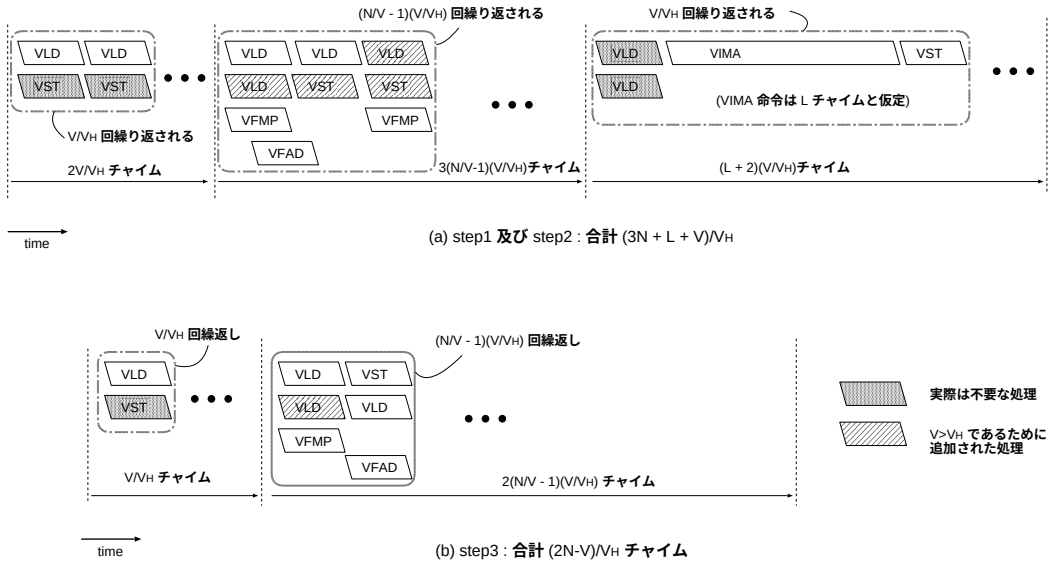


図 13 図 9 のコードを $V > V_H$ の環境で実行した場合の様子
Fig. 13 Execution of the code of Fig. 9 where $V > V_H$.

コンパイルおよび実行した場合の見積りである．

まず，提案手法は，単純なベクトルマクロ命令の適用（`original`）や段数 11 のアンローリング手法（`unroll11`）の演算速度を大きく凌ぐであろうことが図 14 から予想される．また，図 8 (a) との比較では，`vr256` は $U = 257$ のときの式 (5)，(6) による理論式に近いことが分かる．つまり `vr256` は，アンロール段数 $U = 257$ におけるアンローリング手法によるコードと同等な挙動をするといえる．実際に $U = 257$ におけるアンローリング手法のコードを高級言語レベルで生成することが非現実的であるのに対し，提案手法では，同等な振舞いの良い性能のコードをコード量の大幅な増加をともなわずに実現できていることが分かる．

我々の手法では，パラメータ V はベクトルレジスタ長以下の大きさに設定する．つまり $V_H = 256$ の場合には `vr512`，`vr1024`，`vr2048` は実際は提案手法の見積りを表していることにはならない．これらの，い

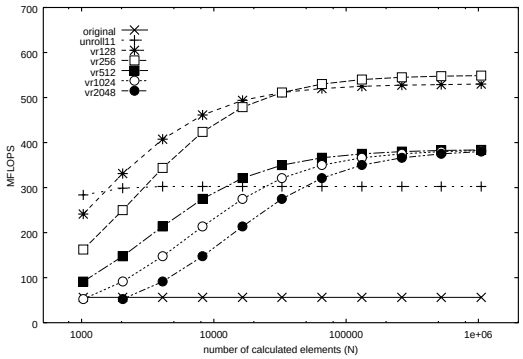


図 14 モデル $\mathcal{V}$ 上での線形回帰演算の演算性能の見積り
Fig. 14 Theoretical estimation of fast first-order linear recurrence algorithm on model $\mathcal{V}$.

わば「パラメータ決定時に想定したベクトルレジスタ長とコンパイル（および実行）するベクトル計算機のベクトルレジスタ長が異なる場合」の曲線と，vr128, vr256 の場合を比較すると，漸近的な性質はもとより N の値が小さい場面での性能も，vr128 や vr256 の方が優れている．この差はほとんど，前述のストリップマイニングの適用によって増加したベクトルロード/ストアの影響による．

4.3 各種パラメータの算出方法

ここでは，任意の N に対して，前節で述べた制約を満たす V および B の値，および後処理にまつわる V', B'（図 10 参照）を決定する方法について述べる．ここで示す手順に対する入力は，V の上限値 V_H と N である．V_H と N の値が定まった時点で各パラメータを決定できるので，図 9 (a) のコード全体を一次回帰演算のライブラリルーチンとして独立させておくこともできる☆．

なお，すべてのパラメータは整数値をとる．

まず，基本手順は以下のとおりである．

(1) $V \leftarrow V_H$ とする．

(2) $B \leftarrow \lfloor N/V \rfloor$ により B を求める．ここで B が偶数ならば $B \leftarrow B-1$ によって B を奇数にする．

(3) $R \leftarrow N - B \cdot V$ とする．$R \geq 0$ である．

(4) $R = 0$ ならば，$B' \leftarrow 0, V' \leftarrow 0$ として終了．$R > 0$ ならば，$B' \leftarrow 3$ とし，$V' \leftarrow \lfloor R/B' \rfloor$ として終了．なお $1 \leq R \leq 2V-1$ なので，$V' < V$ である．

☆ 図 9 (a) の各部のループの繰返し回数が変数の場合は，コンパイル時にはループ長が不定であると判断される可能性があるので，最適化指示行など（たとえば NEC SX 用コンパイラの SHORT-LOOP 指示など）を用いて，ストリップマイニングが不要である旨を明示する必要がある．

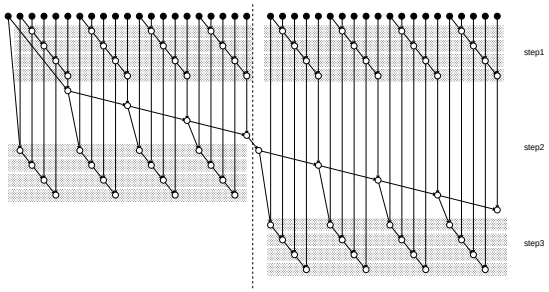


図 15 線形一次回帰演算に対する単純な並列化の適用
Fig. 15 Naive parallelization of the recurrence algorithm.

図 10 中のパラメータに添えた定数は，$N = 27$, $V_H = 4$ のときの設定例である．

$N < V_H$ のときは $B = 0$, $R = N$ となるが，(3) において B' および V' は手順どおり決定され，結果として全体に段数 3 のアンローリングを施したのと同等なコードとなる．

5. 並列ベクトル計算機向け高速アルゴリズム

本章では，4 章で述べた回帰演算の高速ベクトルアルゴリズムを基にした，ベクトルプロセッサを複数備えた並列計算機（並列ベクトル計算機）向けの回帰演算の高速な計算手法について述べる．ここでは簡単のため各 step における後処理は考えない．

図 10 に示したデータフローは，アンローリングの区切りの適当な位置で処理を分割することによって，**図 15** に示すように並列ベクトル環境での計算手順を構成できる．各プロセッサ内では 4 章で示した高速処理が行われ，プロセッサ間での通信量/頻度は，少ない．ただしこの単純な処理の分割では，プロセッサ間のデータ授受に絡むデータフローのクリティカルパス上に多くの計算処理が含まれるので，各プロセッサの受信待ち時間が増大し，並列処理による高速化の効率が著しく低下する可能性がある．

これに対し，各プロセッサの step2 の処理をさらに多段階に分割することによって，総計算量を変化させずにプロセッサ間にまたがるクリティカルパス上の計算処理を削減する手法を示す．**図 16** は本手法による並列アルゴリズムのデータフローである．図に示すとおり，先頭のプロセッサ以外では step2 の処理を step2a + step2b + step2c に分割する．step2a は，クリティカルパス上の計算処理を省くためにあらかじめ各プロセッサが行う前処理で，step2b は送受信とわずかな計算，step2c は各プロセッサ内における step3 でのベクトル処理の準備である．図 16 に示すとおり，step2a と step2b を除くほとんどの計算はベクトル処理がな

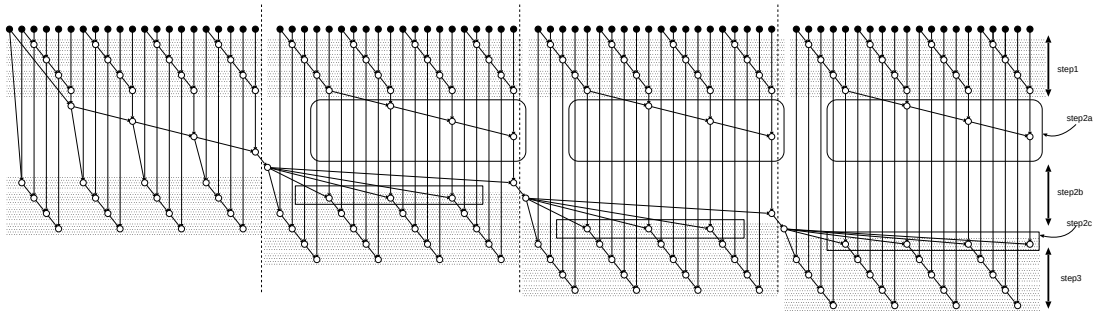


図 16 線形回帰演算の効率的な並列ベクトル処理
Fig. 16 Efficient parallel-vector processing of linear recurrence.

される．step2a はスカラ処理によるが，各プロセッサが独立に同時に計算できるため，総実行時間は短く保つことができる．

1 つのプロセッサ内に独立に制御可能な多数のベクトルパイプラインが存在する場合には，図 16 に示す処理全体を 1 つのベクトルプロセッサで行うこともできよう．ただし効率的な処理のためには，独立動作可能な多数のロード/ストアパイプラインも必要であるので，むしろ 4.1 節で述べたように step1 および step3 にポストアンローリングを施して並列実行可能な命令数を増やす方が得策であろう．

6. 実測および評価

提案手法の有効性の評価のため，商用ベクトル計算機である NEC SX-4 を用いて実測を行った．計算機の諸元は**表 1** に示す．SX-4 は SMP クラスタ構成をとるが，本実測における並列実行時には，1 つのクラスタ内の複数のプロセッサを用いて計算を行った．

測定は，単一プロセッサでの実行と，並列ベクトル実行とについて行い，回帰演算の実行時間を，N を $2^{10} \sim 2^{20}$ の範囲で変化させながら計測した．コード記述は FORTRAN77 を用い，演算はすべて倍精度実数で行った．並列処理は MPI を用いて記述した．計時には，単一プロセッサでの実行を含めてすべて `MPI_Wtime()` を用い，経過時間を測定した．並列実行時の実行時間測定は，各プロセッサに必要なデータを割り付けてバリア同期（`MPI_Barrier()` を使用）をとった時点から，各プロセッサがそれぞれの担当の計算を終了して再びバリア同期をとり終えるまでの時間を計測した．プロセッサ間通信は `MPI_Isend()` および `MPI_Recv()` を用いた．使用したコンパイラおよびバージョンは以下のとおりである：

`mpif90`（FORTRAN90/SX Version 1.0 for SX-4）．

共有メモリの利用やコンパイラによる自動並列化の適用は行っていない．ループアンローリングやループ交換などの自動適用は禁じた．また，ベクトル長 256 のハードウェア用のオブジェクトを生成するよう “`-pvctl vr256`” 指示を行った．

表 1 実測に用いたベクトル計算機のプロセッサ
Table 1 The vector computer used for the evaluation.

機種名	SX-4
ピーク性能	2 GFLOPS
ベクトルレジスタ長	256
ベクトル レジスタ容量	演算 8 本， データ 64 本
マシンサイクル	8 ns
同時実行可能な 実数乗算および加算	加算 1，乗算 1
ロードおよびストア パイプライン	ロード/ストア 8 本 （2 並列で速度飽和）
主記憶バンド幅	16 GB/sec

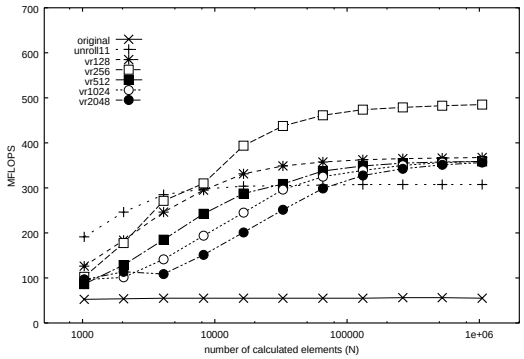


図 17 SX-4 における線形回帰演算の実測結果
Fig. 17 Performance of linear recurrences on the SX-4.

以下では，step1 ～ step3，および B，V などの記号は，すべて図 9 (a) 中のものを表す．

6.1 単一プロセッサ上でのベクトル処理性能

図 17 に，SX-4 の単一プロセッサを用いた実測結果を示す．横軸は N の値，縦軸は演算速度である．演算速度は総計算量 $2N$ を経過時間で割った値（単位 MFLOPS）である．図中の曲線は，図 1 のコードをそのまま実行した結果（`original`），図 2 に示

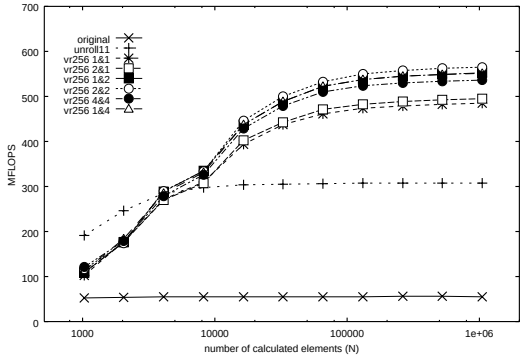


図 18 ポストアンローリングを併用したコードの SX-4 における実測結果
Fig. 18 Performance of the post-unrolled fast code on the SX-4.

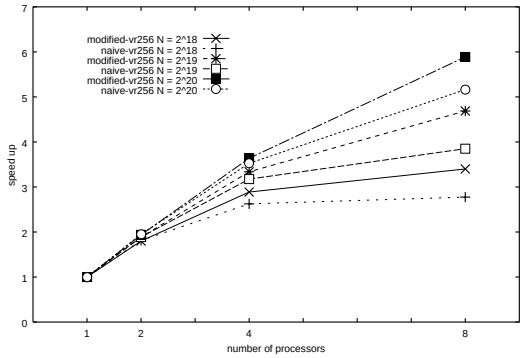


図 19 SX-4 における並列実行結果
Fig. 19 Performance of parallel execution on the SX-4.

したアンローリング手法によるコードで段数 11 のもの（`unroll11`），および，本論文で提案するコード（図 9 (a)）の実測結果である．`original` には，自動ベクトル化によってベクトルマクロ命令が適用されている．提案手法については，ベクトルレジスタ長が各々 128，256，512，1024，2048 であることを想定してパラメータを定めたコードについて実測した．以降，それぞれ `vr128`，`vr256`，`vr512`，`vr1024`，`vr2048` と呼ぶ．実機のベクトルレジスタ長は 256 であるので，`vr512`，`vr1024`，`vr2048` は step1 および step3 の内側ループにコンパイラによるストリップマイニングの適用を受けており，提案手法の本来の実行形態ではない．

図 17 から，ベクトルマクロ命令のみによる実行では高いベクトル処理速度が得られないことが分かる．アンローリング手法によるコードは N が小さい値でもベクトル化による加速が得られているが，漸近的には比較的低い演算速度にしか達していない．

一方，`vr128` および `vr256` では，N が大きくなるにつれて大きな加速が得られていることが分かる．特に `vr256`，すなわち実機のベクトルレジスタを最も効率良く利用するコードは，N が 10,000 を超えるあたりで他を大きく凌ぎ，$N > 100,000$ では他より 2 割以上高速である．`vr128` は，ハードウェア資源を十分生かしてはいないが，ストリップマイニングが適用されているコード（`vr512`，`vr1024`，`vr2048`）と比較すると若干高速である．`vr512`，`vr1024`，`vr2048` はいずれも約 360 MFLOPS の速度で飽和している．

なお，図 17 には，モデル $\mathcal{V}$ における見積りである図 14 で予測される傾向がはっきりと現れており，モデル $\mathcal{V}$ の妥当性が伺える．

6.2 ポストアンローリングを併用した高速化

図 18 には，step1 および step3 にポストアンローリングを適用した際の実測結果を示す．図中，たとえば `2&2` と記した曲線は，step1 および step3 においてそれぞれ 2 段アンローリングしたコードの実測結果である（段数 1 はアンローリングしていないことを意味する）．図では，step1，step3 ともに 2 段のポストアンローリングコードが最高速で，$N = 2^{15}$ で 500 MFLOPS に達し，より大きな N に対しては 565 MFLOPS まで高速化されている．この値は，SX-4 の理論最大性能である 2 GFLOPS に対し 28%以上の速度である．これは，4.2 節で示した理論的な上限値，すなわち「本手法では理論最大性能の 2/7 倍（$\sim 28.6\%$）の演算速度が達成できる」との予測に，きわめて近い値であり，高い効率でベクトル処理が行われたことを示している．また提案手法の計算量を図 9 (a) のアルゴリズムで実際に行っている浮動小数点演算数（およそ $5N + 2V$ 個）であると見なして演算速度を算出すると，1.4 GFLOPS であったことになる．つまり理論最大性能の 7 割の効率でのベクトル処理が実現されている．

6.3 並列ベクトル計算機による実測結果

5 章で述べた並列処理手法の SX-4 における実測結果を**図 19** に示す．横軸は使用したプロセッサ数，縦軸は並列実行による速度向上（台数効果）を示す．図中の曲線は，それぞれのラベルに示す数値を N として `vr256` を並列に実行した結果である．`naive-` を付けた曲線は，図 15 に示す単純な並列処理方式に基づく場合の実行結果で，`modified-`は図 16 によるコードである．`modified-`の方が `naive-`よりも台数効果が高く，8 並列時の性能向上は $N = 2^{20}$ では 14 % である．小さい N については `modified-`と `naive-`の差はより顕著である．台数効果は，$N = 2^{20}$ では 4 並列で 3.6 倍，8 並列で 5.9 倍であった．

本手法は，数台程度の規模の並列ベクトル計算機に

おいて，まずまずの台数効果を得ることができるといえよう．ただし図 19 からも分かるとおり，比較的大きな N に対する回帰演算でなければ効率が大幅に低下するようである．これは，ベクトル計算機の計算速度と通信速度（通信遅延）のギャップの大きさが大きく影響しているものと思われる．また，各プロセッサへのデータの分散処理や結果の収集処理は本実測では実行時間に含めていないため，この時間を回帰演算の総実行時間に含めると，特に分散メモリ型のマシンでは台数効果は大幅に低下するであろう．SMP クラスタの 1 つのノード内のように共有主記憶を持つ並列計算機において，共有データとしてプロセッサ間でグローバルに保持するデータを用いた回帰演算が行えれば，台数効果を損なうことなく並列実行できる可能性もある．共有メモリを使うベクトル計算機向け並列処理手法についての考察は今後の課題である．

6.4 考　　察

6.4.1 規模の小さい問題の場合について

本手法は，$N < 4,000$ ではアンローリング手法（段数 11）に劣っている．この原因は，N が小さい範囲ではスカラ実行部分の演算量が総実行時間に占める割合が増加することが原因であるといえる．ベクトルマクロ命令で 256 個の要素を求める回帰演算は 10 マイクロ秒近くかかるのに対し，`vr256(2&2)` の $N = 1024$ における実測時間は 17.6 マイクロ秒であった．つまり総実行時間に占めるスカラ処理時間の割合が大きい．すなわち，4.3 節で示したパラメータ決定方法には，特に N が小さいときの対処について改良が必要であることを示している．たとえば，以下の案があげられる．

- N が小さい値のときに B が小さくなり過ぎて相対的にスカラ実行部分の計算時間が増加すること防ぐため，B に下限値 B_{min}（たとえば 11 程度）を設けることが考えられる．そして，4.3 節の手順 (2) において得られた B の値がそれを下回る場合には，改めて $B \leftarrow B_{min}$ ，$V \leftarrow \lfloor N/B \rfloor$ とする．
- N と V_H の値によっては，R の値が増加してしまう．これを抑えるためにも，4.3 節の手順 (2) で $B \leftarrow \lfloor N/V \rfloor$ とした際に，B が偶数でかつ $B < V$ のときは $B \leftarrow B+1$ として $V \leftarrow \lfloor N/B \rfloor$ により V を決める方が良いと思われる．このときは $V < V_H$ となるのでベクトルレジスタを十分に使い切ることにはならないが，これによって増加するベクトル命令発行回数はわずかである．

これらの対処の効果の実測および評価は今後の課題である．

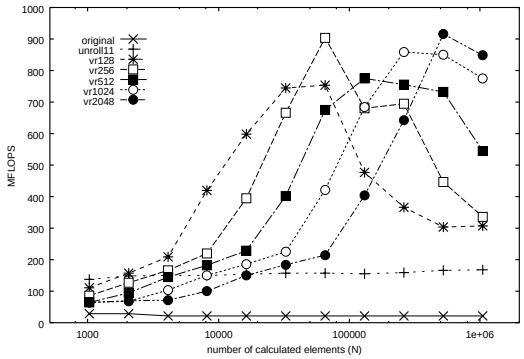


図 20 VPP800 における線形回帰演算の実測結果
Fig. 20 Performance of linear recurrences on the VPP800.

6.4.2 モデル $\mathcal{V}$ に合致しない計算機 — 富士通 VPP800 における実測結果

本論文の提案手法の実測は，富士通の VPP800 においても行った．VPP800 と SX-4 との主要な違いは以下のとおりである．

- ベクトルレジスタ長は最大 2048 の可変型である．
- メモリアクセスにおけるスループットが，アドレスの参照パターン（連続アクセスかストライドアクセスか，など）によって大きく影響を受ける[19]．
- ベクトルロードとストアのパイプラインが 1 本ずつである．
- 図 1 に示す線形回帰演算に直接対応するベクトルマクロ命令を持たない．
- 並列計算機としての構成は，SMP クラスタではなく分散メモリ型である．

図 20 は図 17 と同様の実測を VPP800 で行った結果である．VPP800 では，回帰演算に直接対応するベクトルマクロ命令はなく，図中の `original` はスカラ実行の結果で，速度は低い．`unroll11` も，ベクトルプロセッサの能力を生かし切っているとはいい難い．一方，`vr128` は，N が 1,000 を超える範囲で `unroll11` よりも高速である．

ところで図 20 では，提案手法によるコードは，仮定するベクトルレジスタ長の変化に応じてピークが次第にずれる曲線を描いている．すなわち各コードとも N の増加にともなって演算速度が低下している．これは図 14 や図 17 では見られない振舞いである．

VPP800 がモデル $\mathcal{V}$ に合致しない点を検討すると，たとえば t_s や t_p が定数ではない可能性があげられる．特にベクトルロードおよびストアに要する時間がロード/ストアする要素数だけから算出できない点が大きく異なる．また VPP800 は SX-4 と比較してマシ

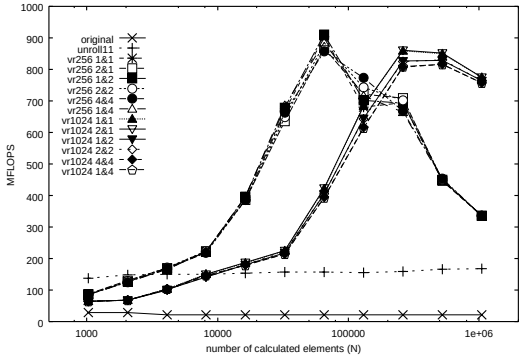


図 21 ポストアンローリングを併用したコードの VPP800 における実測結果
Fig. 21 Performance of the post-unrolled fast code on the VPP800.

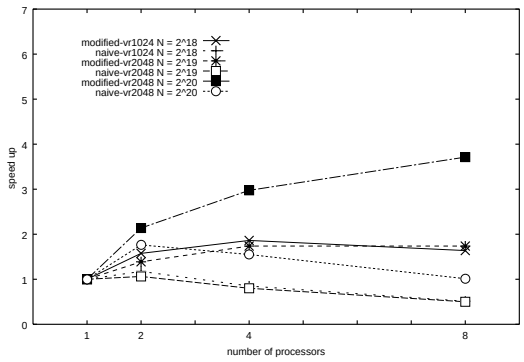


図 22 VPP800 における並列実行結果
Fig. 22 Performance of parallel execution on the VPP800.

ンサイクルが短いのに対してメモリアクセス時のバンクサイクルは大きく，スカラ処理とベクトル処理との競合によるアクセスの乱れの影響も受けやすいようである[19]．

また，図 20 の特徴的な点として，N が 2^{15}，2^{16}，2^{17}，2^{18}，2^{19} と変化するに従って，最高速度を出すコードがそれぞれ `vr128`，`vr256`，`vr512`，`vr1024`，`vr2048` と規則的に変わっていることがあげられる．つまり VPP800 では，B の値がおよそ 256 程度のときに各コード固有のピーク性能に達している．いい換えれば，実行中のベクトルロードやストアのほとんどのストライドが 256 程度☆の場合に高い処理速度が得られていることになる．しかしながら VPP800 が特に 256 前後のストライドの場合だけベクトルロード/ストアが高速に行われるわけではないようである．この挙動の詳細な検討は今後の課題としたい．

図 21 は，`vr256` および `vr1024` について，step1 および step3 にポストアンローリングを適用したコードを VPP800 において実測した結果である．SX-4 の場合（図 18）と異なり，VPP800 ではポストアンローリングによる実行時間の変化は顕著ではなかった．

VPP800 では，$N = 2^{16}$ における vr256 コードおよび $N = 2^{19}$ における vr2048 コードの実測で 900 MFLOPS を超える演算速度が観測されたが，これは VPP800 の理論最大性能である 8 GFLOPS の 1 割強の速度にすぎない．この低い効率の原因としては，提案手法によるコードにおけるベクトルロード/ストアがほとんどすべて連続アクセスではなくストライドアクセスであることがあげられよう．

☆ 4.3 節で述べたとおり，本手法ではストライドがちょうど 256 になることはない．

図 22 は，図 19 の場合と同じ並列処理の VPP800 上での実測結果である．$N = 2^{18}$ では `vr1024` を，$N = 2^{19}$ および $N = 2^{20}$ では `vr2048` を用いた．`naive-` では，並列化による高速化がほとんど得られていない．これに対し `modified-` では，高い並列度が得られているとはいい難いが，いずれの N の値の場合でも `naive-`に対して大幅に良い台数効果を示している．$N = 2^{20}$ において，プロセッサ 8 台で単一プロセッサに対して 3.7 倍の高速化が得られた．

7. おわりに

本論文では，ベクトル計算機向きの線形一次回帰演算の高速な処理手法について述べた．ベクトルロード/ストア量やスカラ処理部の割合を削減し，ベクトルレジスタを効率的に利用するコードを記述すれば，わずかなコード量で，理想的な段数のアンローリング手法と同等の演算性能が得られることが分かった．また，本手法に基づく並列処理では，数台のプロセッサで十分な台数効果が得られることが分かった．

今後の課題としては，回帰演算のループ長が短い場合の立ち上がりの悪さの改善があげられる．そのためにも 4.3 節で述べた各種パラメータの決定方法について，提示した改良案も含めて，実測結果を総合して定量的な評価を行う必要がある．

また，提案手法の評価に用いたモデル $\mathcal{V}$ に合致しない計算機に対して，我々の手法の有効性を明らかにすることや，より一般性の高いモデルを考慮してそのうえで高い性能を発揮できる高速処理手法を開発することなどが，今後の課題としてあげられる．

参考文献

1) Egecioglu, O., Koc, C.K. and Laub, A.J.: Prefix Algorithms for Tridiagonal Systems on Hy-

percube Multiprocessors, *Proc. 3rd Conference on Hypercube concurrent computers and applications*, Vol.2, pp.1539–1545 (1988).
2) 津田孝夫：数値処理プログラミング，第2章，岩波書店 (1988).
3) Nicolau, A. and Wang, H.: Optimal Schedules for Parallel Prefix Computation with Bounded Resources, *Proc. 3rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp.1–10 (1991).
4) 梅尾博司：SIMD 上の並列アルゴリズム，情報処理，Vol.33, No.9, pp.1042–1055 (1992).
5) Lakshmivarahan, S. and Dhall, S.K.: *Parallel Computing Using the Prefix Problem*, Oxford Univ. Press (1994).
6) Lakshmivarahan, S. and Dhall, S.K.: *Analysis and Design of Parallel Algorithms: Arithmetic and Matrix Problems*, McGraw-Hill (1990).
7) Fisher, A.L. and Ghuloum, A.M.: Parallelizing Complex Scans and Reductions, *Proc. ACM SIGPLAN '94 Conf. Programming Language Design and Implementation* (*PLDI*), pp.135–146 (1994).
8) Ben-Asher, Y. and Haber, G.: Parallel Solutions of Simple Indexed Recurrence Equations, *IEEE Trans. Parallel and Distributed Systems*, Vol.12, No.1 (2001).
9) Dongarra, J.J., et al.: *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM (1991).
10) 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.36, No.3, pp.669–680 (1995).
11) 海永正博，久島伊知郎：漸化式のスーパースカラ向け高速化，情報処理学会論文誌，Vol.34, No.12, pp.2592–2598 (1993).
12) 田中義一，前島英雄：命令レベル並列計算機のためのリカレンス演算の高速化手法とコンパイラへの実装，情報処理学会論文誌，Vol.37, No.9, pp.1657–1665 (1996).
13) Wada, H., et al.: High-speed Processing Schemes for Summation Type and Iteration Type Vector Instructions on HITACHI Supercomputer S-820 System, *Proc. Intl. Conf. on Supercomputing*, pp.197–206 (1988).
14) Tanaka, T., Iwasawa, K., Gotoo, S. and Umetani, Y.: Compiling Techniques for First-Order Linear Recurrences on a Vector Computer, *Proc.Supercomputing Conference*, pp.74–181 (1988).
15) Zima, H. and Chapman, B.: *Supercompilers for Parallel and Vector Computers*, Addison-Wesley (1991).
16) Nakazawa, K., Nakamura, H., Imori, H., Kawabe, S.: Pseudo Vector Processor based on Register-Windowed Superscalar Pipeline, *Proc. International Conference on Supercomputing*, pp.642–651 (1992).
17) 長島重夫，田中義一：スーパコンピュータ，第5章，オーム社 (1992).
18) 西川 岳，萩原 孝，安藤憲行，礒部洋子：スーパーコンピュータ SX-4 におけるデータ供給能力，情報処理，Vol.38, No.6, pp.472–478 (1997).
19) 坂井賢一：高並列スーパコンピュータ VPP700E におけるデータ供給能力，情報処理，Vol.38, No.6, pp.479–484 (1997).

(平成 13 年 8 月 31 日受付)
(平成 14 年 2 月 13 日採録)

川端 英之（正会員）

1992 年京都大学工学部情報工学科卒業．1994 年同大学大学院工学研究科博士前期課程修了．同年より広島市立大学情報科学部助手．高性能計算，自動並列化技術に関する研究に従事．ACM，IEEE-CS 各会員．

湯之上康一

2002 年広島市立大学情報科学部卒業．現在同大学大学院情報科学研究科博士前期課程在学中．高性能計算に興味を持つ．

津田 孝夫（正会員）

1957 年京都大学工学部電気工学科卒業．1979 年より京都大学工学部情報工学科教授．1996 年より同大学名誉教授，広島市立大学情報科学部教授．工学博士．モンテカルロ法，自動ベクトル化/並列化コンパイラ，並列数値処理等に関する研究に従事.「モンテカルロ法とシミュレーション」（培風館),「数値処理プログラミング」（岩波書店）等の著書がある．昭和 63 年度および平成 3 年度本会論文賞授賞．ACM，SIAM 各会員．