ベクトル計算機上でのRetry型アルゴリズム群について

今 村 俊 幸†

ベクトル計算機での間接アドレス参照を含む総和計算において有効なアルゴリズム群について述べる．本手法はRetryアルゴリズムと呼ばれる方法の拡張であり，2次元に拡大したインデックス表現により投機的なメモリ書き込みのペナルティを軽減する．従来方法と比べても，メモリ使用量，速度性能の面で優れた手法である．特に，従来のRetryアルゴリズムが苦手としてきた要素分布を持つ問題でアクセス競合を激減するとともに，場合によっては10倍以上の高速化が確認された．

A Group of Retry-type Algorithms on a Vector Computer

Toshiyuki Imamura†

In this paper, a group of effective algorithms for indirect summation on a vector computer is proposed. The method is an extended version of the conventional retry algorithm, and its two-dimensional address mapping reduces the penalty of its speculative executions. Comparison with the existing methods illustrates that it has advantages on memory usage and performance issue. Implementations on several types of vector computers show that proposed method reduces memory access conflict, and it performs successively up to ten times faster than the conventional retry algorithm.

1. はじめに

間接アドレス参照を含む加算式(1)は，カウンタiで代表されるオブジェクトのfへの寄与を計算する際に利用される．

$$f(index(i)) := f(index(i)) + a(i) \qquad (1)$$

上式は非常に多くの分野のプログラムにおいて出現するものであるが，特に，科学技術計算分野に絞ると

(1) バケッツソートなどのヒストグラム計算
(2) 粒子問題における粒子の場への寄与計算
(3) 有限要素法における剛性マトリックス生成
(4) 積分計算一般

などを取り扱った文献に登場する（もちろん例出以外の一般的な問題にも出現するごくありふれた計算式である）．例(1)は，準数値計算の典型例であるといえる．例(2)では，主にPIC（Particle In Cell）法を用いる粒子問題に本計算が出現し，数値解法との関連からCharge Depositと呼ばれている[1]．本論文でもPIC法でのCharge Depositに由来させて，式(1)を**deposit式**と呼ぶこととする．例(3)，(4)では分散する計算要素のアセンブリ部分を担う重要な計算部分といえる．

科学技術計算に能力を発揮する計算資源として，ベクトルプロセッサの使用が考えられる．しかしながら，ベクトルパイプラインを使用した場合の本式の処理には計算結果の誤りが含まれる可能性があるため，コンパイラが自動的にベクトル化することはない．利用者がアドレス衝突の有無をコンパイラに指示した場合のみ，ベクトルパイプラインを用いた高速処理が可能となる．

この制約はベクトルプロセッサを利用するうえで大きな障害となっており，古くからその解決手法が研究されてきた．プログラムの積極的な変更によりベクトル化を行った研究として，Nishiguchiらの Work Vector アルゴリズム[2] と Abe-Nishihara らの Retry アルゴリズム[3),4] が知られている．Work Vector アルゴリズム（止まり木法とも呼ばれる）は，配列fに対して人工的なインデックス拡張を施しデータの依存関係を解消する方法である．一方，Retryアルゴリズムはベクトルレジスタ–メモリ間の動作に着目して，強制的なベクトル処理による結果不正の検出を行うとともに不正項の再処理を行う方法である．この手法は村井らによって[5] Work Vectorアルゴリズムよりも使用メモリが少なく，さらにインデックスの分布がまばらなときに性能を発揮することがソーティング問題

† 電気通信大学電気通信学部情報工学科
Department of Computer Science, the University of Electro-Communications

```
1: array f(1:m), a(1:n), index(1:n)
2: do i=1, n
3:     f(index(i))=f(index(i))+a(i)
4: end do
```

図 1 対象となる deposit 式計算プログラム
Fig. 1 A target program of deposit calculation.

(上記例 (1)) への適用実験によって示されている．松村，Mochizuki らの報告では[6),7)]，分子軌道計算における積分式において本式が出現する．しかしながら，積分の構成上，同一要素へのアクセスが頻出するため，Retry アルゴリズムよりも Work Vector アルゴリズムの方が有効であると指摘している．また，Work Vector アルゴリズムは有効な手法だが，メモリ使用量が大規模問題への適用の障壁となることもあわせて報告されている．このように，従来の手法は問題ごとに有効であるかどうかの判断をしなくてはならなかった．本論文では，従来手法の問題点であったメモリ使用量，インデックスの分布の両問題点を改善するアルゴリズムの提案を行う．

以下，2 章で従来手法を紹介した後，3 章でそれらの問題点を考察し，4 章でそれらを改善する新アルゴリズムを提案する．5 章では，NPB [10)] IS の乱数データならびに分子軌道計算における数値積分での評価を行い，最後に 6 章でまとめを行う．

2. 従来アルゴリズムについて

2.1 準　　備

本論文では**図 1** に示すプログラムを対象とする．ここで，n は加算要素（配列 a）の要素数，m は f の定義域の上限であり本論文では問題サイズとも呼ぶ．配列 index の値域は $[1:m]$ に一致しているとする．また配列 index の値を**キー**と呼び，論文中ではキーがとりうる値の総数を記号 l で表現する．

実装上の表記において，ベクトル処理をする際のレジスタ長を N_v と記述し，アルゴリズム中でベクトル処理を陽に表現するために表記 '//' を使用する．i_b から i_e までの要素を持つベクトルは $/i_b{:}i_e/$ で表す．

2.2 Work Vector アルゴリズム（止まり木法）

Work Vector アルゴリズムは，加算対象である配列 'f' とは別に広い 2 次元ワーク配列を用意し，ベクトル長 K にストリップマイニングされた deposit 式中のアドレス指定 'index(i)' に対して，人工的に導入したインデックス 'j' を付加し 2 次元拡張する手法である[2)]．2 次元拡張されたインデックス (index(i), j) はループ依存性が完全に解消されるため，コンパイラに指示子を与えることなくベクトル化可能である．**図 2** に Work Vector アルゴリズムを示す．本アルゴリズムは，プログラムに対してわずかな修正で実現できる単純な手法であるためしばしば科学技術計算で使用されることがある．

```
Algorithm WorkVector
INPUT :: f(1:m), index(1:n), a(1:n);
OUTPUT :: f(1:m);

begin { MAIN }
  w(1:m,1:K):=0;
  for i_start:=1 to n step K do
    i_end:=min(i_start+K-1,n);
    for i:=/i_start:i_end/ vector_do
      j:=i-i_start+1;
      w(index(i),j):=w(index(i),j)+a(i);
    end for
  end for
  f(1:m)=f(1:m)+w(1:m,1:K);
end
```

図 2 Work Vector アルゴリズム
Fig. 2 The Work Vector algorithm.

2.3 Retry アルゴリズム

もう 1 つの代表的なアルゴリズムである Retry アルゴリズムは**「有限長のベクトルレジスタ上のデータがメモリにストアされるとき，書き込み順序はつねに一定の順序で行われる」**というベクトル計算機のハードウェア特性を利用する．たとえば，ストア操作が FIFO に従う場合は，重複するキーの要素はベクトルレジスタ内で最後に登場したもののみメモリ上に反映される．この手法の実現には，コンパイラによってベクトル化されないループをコンパイラ指示子を用いて強制的にベクトル演算する必要がある．強制的なベクトル演算と同時に結果の不正を検出し，それらがなくなるまで繰り返し処理する方法である[3),4)]．

図 3 に Retry アルゴリズムを示す．配列 f への加算と同時に配列 stamp に書き込みを試みたときのループカウンタの値を記録している．先に示したベクトル計算機の特性から，書き込みに成功した場合は配列 stamp にそのループカウンタ値が記録されていることになる．したがって配列 stamp の内容とループカウンタを比較し，結果不正項を検出することができる．

なお，本アルゴリズムはベクトル計算機のハードウェア特性を利用しているために，スカラ計算機では正しい結果を得られない場合があるが，スカラ機でも正しく動作する方法が文献 8) で報告されている．

3. 従来方法の再考

3.1 各アルゴリズムの作業メモリ使用量

前章で示した，2 つのアルゴリズムが使用する作業配列のメモリ使用量を評価する．Work Vector アル

```
Algorithm Retry
INPUT :: f(1:m), index(1:n), a(1:n);
OUTPUT :: f(1:m);

function retry_core(f,index,a,list,ix);
begin
   iy:=0;
   for j:=/1:ix/ vector_do
      i:=list(j);
      f(index(i)):=f(index(i))+a(i);
      stamp(index(i)):=j;
      if stamp(index(i)) != j then
      /* if results are falure,
         they are stack to list() */
         list(iy++):=i;
      fi;
   end for
   return iy;
end

begin { MAIN }
   list:=/1:n/; ix:=n;
   while ix>0 do
      ix:=retry_core(f,index,a,list,ix);
   end while
end
```

図 3 Retry アルゴリズム
Fig. 3 The Retry algorithm.

```
I=LIST(J)
IDX=INDEX(I)
FR=F(IDX)    ! レジスタ FR に F(:) をいったん退避
F(IDX)=J     ! F(:) を stamp(:) として利用
FR=FR+A(I)   ! 加算の結果はレジスタ FR 上に
FLAG=(F(IDX)/=J)
F(IDX)=FR    ! レジスタ FR 上の結果をメモリに
IF(FLAG)THEN
   IY=IY+1; LIST(IY)=I
ENDIF
```

図 4 配列 stamp を陽に使用しない実装例
Fig. 4 An example code on which the array 'stamp' is not used explicitly.

ゴリズムは mK を要する．一方，Retry アルゴリズムは，アルゴリズムを説明するうえでカウンタの内容を記録するための配列 stamp が必要であるが，実装時に**図 4** のようにベクトルレジスタを利用することで配列 stamp を主記憶上に確保しなくてもよい．また，図 3 では結果不正項のリスト（list）が登場するが，実際は n を適当なサイズに分割して実装できるため，その長さは定数（$\mathrm{N_s}$）と見なしてもよい．したがって，Retry アルゴリズムは長さ $\mathrm{N_s}$ の配列以外に特別な作業配列を必要としないアルゴリズムである．

Work Vector アルゴリズムはベクトル長 K によるベクトル化を行うため，性能向上には $K \gg 1$ が要求される．文献 6) では，性能と使用メモリ量とのトレードオフとして $K = 64$ が選択されている．応用問題の中には，配列 f が高次元配列であるとともに 256^3 のような大規模なものもある．$K = 64$ をそのままこの問題に適用すると，作業配列 w に 8 ギガバイトが必要となる．メモリ使用量があまりにも大きい場合には，性能面を犠牲にして K を小さくとる必要が生じる．これらのことからも Work Vector アルゴリズムの大規模問題への適用には適切な K の選択について問題点が残る．

3.2 キー分布とレジスタ長の関係

Retry アルゴリズムは，本来，データの依存性のため LD(ロード)–ST(ストア), LD–ST, ... と処理すべきところを LD, LD, ..., ST, ST, ... と変更しベクトルパイプラインの効果を引き出している．このような変更は結果不正のリスクも同時に負うため，通常のアルゴリズムと比較して投機的な側面を有している．ここで残項処理の回数が ST を飛び越えた LD の数，すなわちレジスタ長に依存するとともにレジスタ長とキー分布が性能に関係することが予想される．本節ではこれらの関係について評価を行う．議論を単純にするために，総要素数 n は十分大きくかつ，インデックスは周期 L と仮定する．

$L \geq \mathrm{N_v}$ のとき，レジスタ内の各要素はすべて異なるインデックスを保持するため，再処理は発生せず 1 回の処理で終了する．Retry アルゴリズム中はレジスタ長でのストリップマイニングが適用され，かつ依存関係から，他のロード–加算–ストアの手続きが重なることがない．したがって単一ベクトル演算器でのコストの倍数で評価される．総コストは次式のようになる．

$$T_{L \geq \mathrm{N_v}} = \frac{n}{\mathrm{N_v}}(\alpha + \beta \mathrm{N_v}) \qquad (2)$$

一方，$L < \mathrm{N_v}$ のとき，1 回の処理でレジスタ内の L 要素のみが正しく加算されるため，残項として $n \cdot (\mathrm{N_v} - L)/\mathrm{N_v}$ 個の要素が残る．以下各回の残項は等比数列として表せるので全体のコストは

$$T_{L < \mathrm{N_v}} = \sum_{k=0}^{c} \frac{n}{\mathrm{N_v}} \left(\frac{\mathrm{N_v} - L}{\mathrm{N_v}} \right)^k (\alpha + \beta \mathrm{N_v}) \quad (3)$$

となる．ここで，c は $n((\mathrm{N_v} - L)/\mathrm{N_v})^c \leq 1$ を成立させる最小の数である．$c = -\log n / \log((\mathrm{N_v} - L)/\mathrm{N_v})$ を代入すると，

$$T_{L < \mathrm{N_v}} = \frac{1}{L}(n + L/\mathrm{N_v} - 1)(\alpha + \beta \mathrm{N_v}). \qquad (4)$$

n は十分大きいと仮定しているので，第 2 係数は n と見なしてよい．したがって，式 (2)，(4) の違いは第 1 係数の分母部分となり，レジスタ長 $\mathrm{N_v}$ と L の比がコスト T に影響していることが分かる．

```
Algorithm Retry-WV
INPUT :: f(1:m), index(1:n), a(1:n);
OUTPUT :: f(1:m);

function retryWV_core(w,index,a,list,ix);
begin
  iy:=0;
  for j:=/1:ix/ vector_do
    i:=list(j); d:=mod(j-1,K)+1;
    w(index(i),d):=w(index(i),d)+a(i);
    stamp(index(i),d):=j;
    if stamp(index(i),d) != j then
      list(iy++):=i;
    fi;
  end for
  return iy;
end

begin { MAIN }
  list:=/1:n/; ix:=n;
  w(1:m,1:K):=0;
  while ix>0 do
    ix:=retryWV_core(w,index,a,list,ix);
  end while
  f(1:m):=f(1:m)+w(1:m,1:K);
end
```

図 5 Retry-WV アルゴリズム（斜体字部分は Retry アルゴリズムに対する変更点）
Fig. 5 The Retry-WV algorithm (obliquing parts are the enhancement to the Retry algorithm).

ここまで，レジスタ長 N_v をハードウェア固有の定数として扱ってきた．しかしながら，ストリップマイニングなどの手法により最内ループ長を変化させることが可能であり，式 (4) の形から L に近いレジスタ長を選択することで性能改善の可能性があることが示唆される．ただし，一般的にベクトル演算器の立ち上がり時間 α は無視することはできないため[9]，予備実験によりあらかじめ適切なベクトル長を求める必要がある．

4. 新規手法の提案

4.1 人工的なインデックスの導入：Retry-WV アルゴリズム

図 5 に示すアルゴリズムは，Retry アルゴリズム（図 3）の関数 `retry_core` に対して 2 次元作業配列へのマッピングを施したものである．カウンタ j とその K に対する剰余による 2 次元アドレス指定を行う．Work Vector アルゴリズムでの議論に従うと，このマッピングにより連続する K 区間内での同一アドレスへのアクセスを避けることができる．本手法を Retry アルゴリズムと Work Vector アルゴリズムの中間に位置することから Retry-WV アルゴリズムと呼ぶ．

本アルゴリズムでは書き込み結果の正当性を判断する配列 stamp に対しても 2 次元拡張を行い，加算先アドレスにループインデックスを上書きする．Retry アルゴリズムと同様に，ベクトル計算機のメモリへの書き込み動作の特性を利用しているので，最終的にメモリ上に書き込まれた結果とループインデックスを比較することで不正項を判別できる．

図 6 は Retry アルゴリズムと Retry-WV アルゴリズムの結果不正項検出の概念図を示している．それぞれの図の上段には計算過程，下段には結果不正項の検出過程を図示した．点線で囲まれた部分が，Retry アルゴリズム使用の前提であるベクトル計算機のレジスタからメモリへの書き込み動作の特性部分を示すものである．図 6 中の配列 index() の網掛け部分が結果正常となる部分であり，f(1) への書き込み成功数を比較すると左側の Retry アルゴリズムの 1 に対して Retry-WV アルゴリズムは 3 となる．結果不正の積み残しの数はそれぞれ，Retry アルゴリズムが 3 に対して Retry-WV アルゴリズムが 1 となっており，1/3（$= 1/K$）に減少する．

4.2 Retry アルゴリズムと Work Vector アルゴリズムの関係

本手法は，Retry に Work Vector の特徴である 2 次元作業領域へのマッピングを施しているため，次に示すようにパラメータを適切に設定することでそれぞれのアルゴリズムに帰着させることができる．

(1) $K = 1$ とした場合，作業領域 w と f とを同一視すれば，Retry-WV と Retry は一致する．
(2) $K = N_v$ のとき，コアのループが N_v でストリップマイニングされることを考慮すると，結果不正項は発生しない．したがって，図 5 の変数 stamp への代入ならびに if 文などは実質的に無意味となるため，$K = N_v$ の Work Vector アルゴリズムと一致する．

ところで，Work Vector アルゴリズムは K をベクトル長として処理するため，性能が K に依存する．Retry-WV アルゴリズムでは K とレジスタ長 N_v は無関係であり，ベクトル長とレジスタ長 N_v の一致を保証できる．そのため，ベクトル演算器の実性能として最良の性能が期待できる．また一方で，結果不正が増大する場合は Retry アルゴリズム同様再処理のペナルティを負うことになるが，先の図でも示したとおり結果不正の出現期待率が $1/K$ となるため，再処理回数は Retry アルゴリズムよりも少ないと期待できる．

4.3 使用メモリについて

Retry-WV アルゴリズムで必要な作業配列のサイズは，w と stamp のあわせて $2mK$ であるが，Retry と同様に配列 stamp を陽に確保する必要がないため

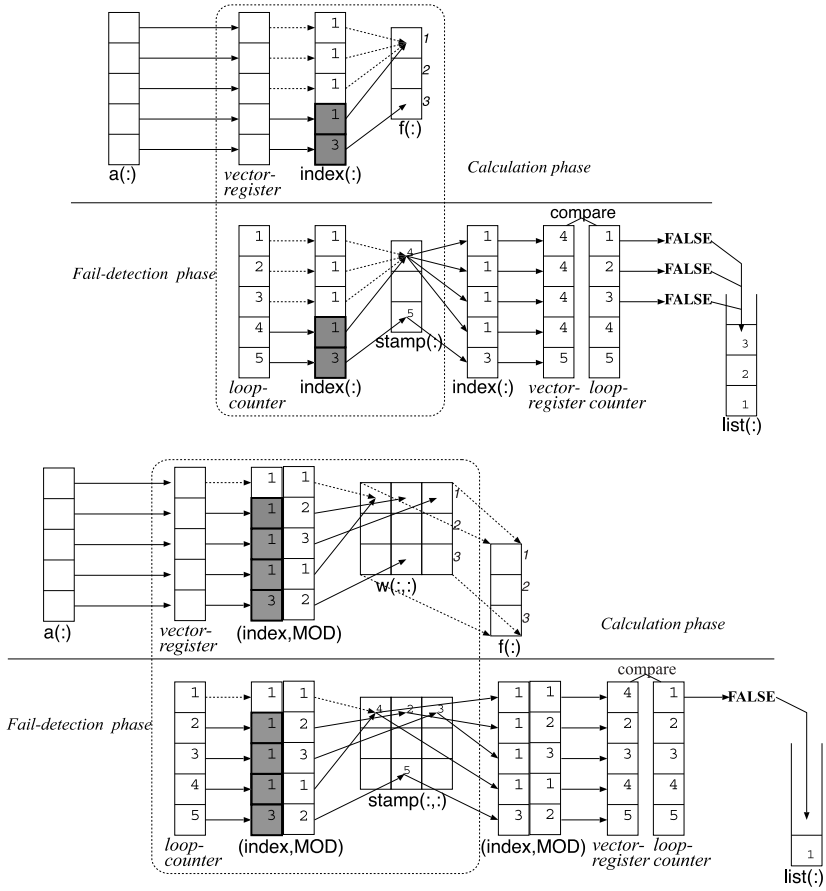

図 6 結果不正項検出のメカニズム（上段は Retry アルゴリズム，下段は Retry-WV アルゴリズム）
Fig. 6 Schematic view of the fail-index detection mechanism on the Retry algorithm (top) and the Retry-WV algorithm (bottom).

実質 mK である．これは，Retry アルゴリズムと比較すると K 倍，同一の K を選択した Work Vector アルゴリズムと比較して同一の作業配列が必要となる．当然，Work Vector アルゴリズム同様，メモリイータの性質を有するためメモリ使用量と実行性能とのトレードオフを考慮しなくてはならない．

4.4 動的なレジスタ長の選択について

3.2 節で示した Retry アルゴリズムにおけるキー分布とレジスタ長の関係より，キー数がレジスタ長以下の場合には，その性能が著しく劣化する可能性がある．これは同時に Retry-WV アルゴリズムにもあてはめることができ，レジスタ長を変化させるアルゴリズムを考えることができる

図 3，図 5 で示したアルゴリズムでは，変数 ix の値によって書き込みに成功した要素数を取得できることから，区間平均の意味で周期 l を予測可能である．しかし実データのように周期的な分布とならない場合は，予測値を各反復ごとにレジスタ長として利用することが性能向上を保証するとは限らない．また，予測される周期はレジスタ長よりも短いため，得られた周期情報からレジスタ長を縮小することはできるが伸長する判断を行うことはできない．そのため，本論文では要素の総処理数が閾値を超えた際にレジスタ長を変更し長レジスタ版 ↔ 短レジスタ版と 2 種類のレジスタ長の切替えを行うヒューリスティックス手法を導入する．インデックス周期がレジスタ長を超える場合は処理要素数は n であり，インデックスの重なりが最悪のケースとなる周期 1 の場合でも処理要素数が $\frac{n}{\mathrm{N_v}} \cdot K$ となる．したがって，総処理数は $\frac{n}{\mathrm{N_v}} \cdot K$ と n の間に分布することとなる．その分布区間の中から閾値を決めるが，それを次の式を用いてパラメータ a（$0 \leq a \leq 1$ の定数）で決定する．

$$\mathrm{tol} := an + (1-a)\frac{n}{\mathrm{N_v}} \cdot K \qquad (5)$$

アルゴリズムの切替えは初めの 1 回目の処理は長レジスタ版により処理を行い，処理された要素数が閾値

を下回ったときに短レジスタ版に切り替えることにする．後述する実験では，あらかじめ設定した b 回ごとに長レジスタ版に遷移し直すことで，データ分布の変動に対応するヒューリスティクスを採用している．

ここまで述べてきたように，Retry アルゴリズムには Retry-WV アルゴリズムならびに，レジスタ長を可変とするレジスタ可変アルゴリズムのバリエーションが存在することになる．本論文ではこれらを総称して **Retry 型アルゴリズム群**と命名する．

4.5 計算量についての考察

本節では，従来型のアルゴリズムも含めて Retry 型アルゴリズム群の計算量について定量化を試みる．ただし，3.2 節同様，議論を単純化するためインデックスデータは周期 L のデータを扱う．

まず，Wrork Vector アルゴリズムは要素数 n のループをベクトル長 K で分割し処理することから，核となるベクトル長 K での加算処理は n/K 回行われる．また，前後に作業領域の初期化とその総和計算があるため全体として処理時間は

$$T_{\mathrm{WV}} = \frac{n}{K}(\alpha + \beta K) + 2(\alpha + \beta K m) \qquad (6)$$

となる．ここで簡単化のために，核計算部分と初期化・後処理の総和計算でのベクトル演算器の振舞いは同一とし，両者における立ち上がり時間 α と処理速度 β は同じものを使用する．また，バンクコンフリクトは生じないと仮定する．

次に，Retry アルゴリズムは 3.2 節での議論を基にし，式 (2) や (4) を用いて近似すれば，

$$T_{\mathrm{Retry}} = \frac{n}{\min(L, \mathrm{N_v})}(\alpha' + \beta' \mathrm{N_v}) \qquad (7)$$

が得られる．ただし，本アルゴリズムではベクトル演算器よりも，主記憶–レジスタ間でのメモリ転送に要する時間の効果が大きいものと考えられるため，式 (6) の α，β とはダッシュをつけて区別している．式 (7) の第 1 項の分母は，第 1 回目の加算実行で処理可能な最大要素数から導かれた定数である．Retry-WV(K) アルゴリズムについて，最良の場合を考えると，分母の L の部分を LK に置きかえることができる．式 (7) の L を LK と置き換え，また，Work Vector アルゴリズム同様に作業配列の初期化と最後の総和計算を加えて，

$$T_{\mathrm{R\_WV}} = \frac{n}{\min(LK, \mathrm{N_v})}(\alpha' + \beta' \mathrm{N_v}) + 2(\alpha + \beta K m) \qquad (8)$$

が得られる．レジスタ長可変の場合は $\mathrm{N_v}$ が変化し，長ベクトル版と短ベクトル版とで評価式が異なる．厳密には，$\mathrm{N_v}$ を被積分パラメータとして確率分布を乗じて積分を計算することとなる．これら評価式は Retry アルゴリズムや Retry-WV(K) アルゴリズムの計算時間に対する最良の評価指針を与えるものとなる．

問題設定 n，m，L が決まっているときに，Work Vector に対して Retry-WV(K) を最良にする K は $F = T_{\mathrm{WV}} - T_{\mathrm{R\_WV}}$ を最大にする K となる．

$$F = T_{\mathrm{WV}} - T_{\mathrm{R\_WV}} = \frac{n}{K}(\alpha + \beta K) - \frac{n}{\min(LK, \mathrm{N_v})}(\alpha' + \beta' \mathrm{N_v}). \qquad (9)$$

i) $LK \le \mathrm{N_v}$ の場合，

$$\frac{\alpha' + \beta' \mathrm{N_v} - L\alpha}{\beta L} \le K \qquad (10)$$

のとき $T_{\mathrm{WV}} \ge T_{\mathrm{R\_WV}}$ となる．この不等式を満足する K が存在するためには，$(\alpha' + \beta' \mathrm{N_v} - L\alpha)/\beta \le \mathrm{N_v}$ が成り立つ必要がある．整理すると，$\mathrm{N_v} \ge L \ge (\alpha' + (\beta' - \beta)\mathrm{N_v})/\alpha$ が要請される．α'，β' は，Retry アルゴリズムなどで間接アドレス参照や配列 stamp による結果不正検出のコストを考慮したもので，α，β の数倍としてよい．一般的に，$\mathrm{N_v}$ や α/β は大きい値（数百から数千）なので[9]，L の存在を仮定してもかまわないと考えられる．

いま，L が式 (10) の左辺を正とすると仮定する，$\partial F/\partial K \ge 0$ であるので $K = \mathrm{N_v}/L$ で F が最大となる．一方，式 (10) の左辺が負の場合には $\partial F/\partial K < 0$ となるために，$K = 1$ が最良の選択となる．

ii) $LK \ge \mathrm{N_v}$ のとき，$\partial F/\partial K \le 0$ より $K = \mathrm{N_v}/L$ で F が最大となる．

ここまでの評価は，データの分布 L を 1 つの値に集中するとの仮定をおいたものではあるが，L があらかじめ分かっていて，十分なメモリの余裕がある場合には $K = \mathrm{N_v}/L$ を目安として設定すれば提案アルゴリズムの効果が得られることになる☆．

5. 性能評価

本章では Retry 型アルゴリズム群を実機上で評価する．評価に使用したベクトル計算機にはマルチプロセッサの計算機も含まれるが，本アルゴリズムの実測はシングルプロセッサ上で測定したものである．

5.1 Retry 型アルゴリズム群のレジスタ長と実行性能との関連

4.4 節で示した動的なレジスタ長の選択の効果を評価するため，Retry 型アルゴリズム群の基本をなす

☆ 1 つの目安にすぎず，実用には式 (10) の左辺が負の場合や実装による揺れや係数などの影響を考慮する必要がある．

表 1 VPP5000 におけるレジスタ長を変更した際の Retry アルゴリズムの実行時間（単位は秒）
Table 1 Elapsed time of the Retry algorithm with some register-length configurations on a VPP5000 (unit is in second).

		index キー数（l）			
		1	4	16	64
レジスタ長	2,048	13.02	8.422	3.790	.5364
	1,024	7.400	5.082	1.650	.3351
	512	5.400	3.286	1.026	.2301
	256	3.605	2.033	.6830	.1630
	128	2.982	1.562	.5377	.1349
	64	2.703	1.335	.4655	.1218
	32	2.563	1.199	.4207	.1292
	16	2.452	1.138	.4087	.1792
	8	2.400	1.130	.4644	.2722

256	1,024	4,096	16,384
.1223	.0364	.0148	.0096
.0774	.0240	.0114	.0097
.0577	.0196	.0118	.0112
.0425	.0179	.0131	.0137
.0409	.0208	.0200	.0206
.0492	.0369	.0327	.0320
.0772	.0628	.0550	.0546
.1289	.1087	.1026	.1020
.2188	.2044	.2007	.2003

$((m, n) = (2^{21}, 2^{14}))$

Retry アルゴリズムの中核ループを異なるレジスタ長でストリップマイニングし，8 種類の index データを用いて測定を行った．実験に使用したインデックスは，NPB（Nas Parallel Benchmark）[10] に含まれる問題 IS の疑似乱数生成ルーチンで生成した要素数 2^{21} の一様乱数である．富士通 VPP5000，日本電気 SX-6 上☆での結果を次の**表 1**，**表 2** に示す．表にはそれぞれのキー数での最良の結果を示したものに網掛け処理を行った．

これらの結果から，特に周期的データでない場合にもレジスタ長変更による効果が認められた．VPP5000 ではレジスタ長 2,048 から 256 へ 1/8 もの縮小が有効であり，SX-6 でも同様に 1/2 以下に縮小する（最大で 1/8 まで縮小する）ことでの性能向上が見られる．

5.2 ヒストグラムテストでの評価

次に，Retry 型アルゴリズム群の性能測定テストとしてヒストグラムテスト[5] による性能測定結果を以下に示す．加算要素数を 2^{21}（= 2,097,152）で固定するとともに，粒子問題で高密度（1 セルあたり 100 粒子以上）な例と中密度（1 セルあたり 10 粒子程度）な例となるよう配列 f のサイズを 2^{14}（= 16,384）と 2^{17}（= 131,072）の 2 種を選び，NPB の問題 IS と

☆ 日本原子力研究所計算科学技術推進センターのものを使用．

表 2 SX-6 におけるレジスタ長を変更した際の Retry アルゴリズムの実行時間（単位は秒）
Table 2 Elapsed time of the Retry algorithm with some register-length configurations on an SX-6 (unit is in second).

		index キー数（l）			
		1	4	16	64
レジスタ長	256	12.40	9.944	3.643	.4706
	128	7.100	5.444	1.917	.2921
	64	4.452	3.145	1.079	.1984
	32	3.039	1.975	.6710	.1615
	16	2.297	1.352	.4755	.1752
	8	1.920	1.028	.4405	.2611

256	1,024	4,096	16,384
.0740	.0201	.0123	.0112
.0577	.0248	.0181	.0165
.0586	.0356	.0297	.0273
.0760	.0583	.0506	.0482
.1170	.0961	.0893	.0891
.1935	.1819	.1725	.1661

$((m, n) = (2^{21}, 2^{14}))$

同様に，連続する 4 個の一様乱数列の平均を index の値とした．**図 7** と**図 8** の上部分は $m = 2^{14}$，下部分は $m = 2^{17}$ について，index の分布（l）を 2 のベキで変化させたときの VPP5000，SX-6 での EPS 値（Elements Per Seconds，1 秒あたりの加算要素数）をプロットしたものである．

それぞれのグラフには，

- レジスタ長可変 Retry-WV アルゴリズム（$K = 64$）
- レジスタ長可変 Retry-WV アルゴリズム（$K = 16$）
- レジスタ長固定 Retry-WV アルゴリズム（$K = 64$）
- レジスタ長固定 Retry-WV アルゴリズム（$K = 16$）
- Retry アルゴリズム
- Work Vector アルゴリズム（$K = 64$）
- Work Vector アルゴリズム（$K = 16$）

の 7 つの結果を重ねて示している☆☆．なお，固定レジスタ長 Retry-WV アルゴリズムではレジスタ長をハードウェアの持つレジスタの最大長に，可変長アルゴリズムでは最大長の 1/8 にまで縮小するように設定し，切替えパラメータは $a = 1/3$，$b = 10$ とした．また，m が 2 のベキでありバンクコンフリクトの危険があるので，実装上は 1 を加えて奇数としている．

☆☆ グラフ中でレジスタ可変アルゴリズムには (*) をつけて区別している．

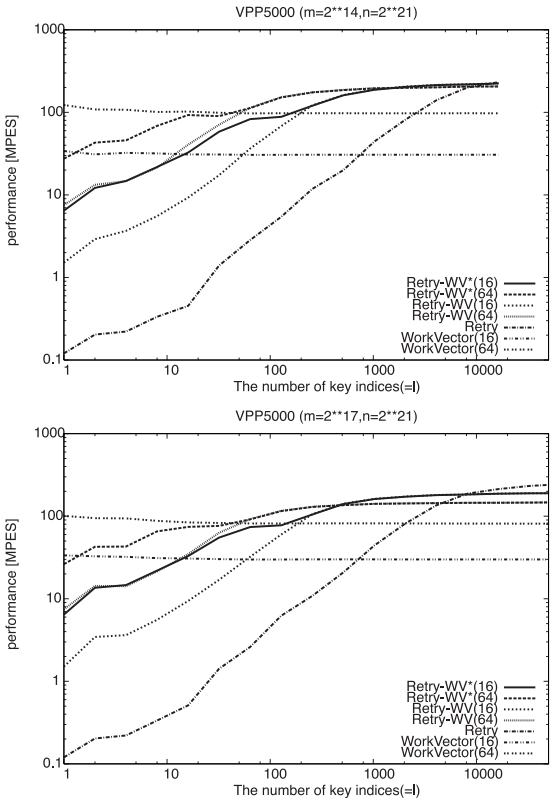

図 7 VPP5000 におけるヒストグラムテストの性能結果，上：$(m,n)=(2^{14},2^{21})$，下：$(m,n)=(2^{17},2^{21})$

Fig. 7 Performance results of the histogram test on a VPP5000, Top: $(m,n)=(2^{14},2^{21})$, Bottom: $(m,n)=(2^{17},2^{21})$.

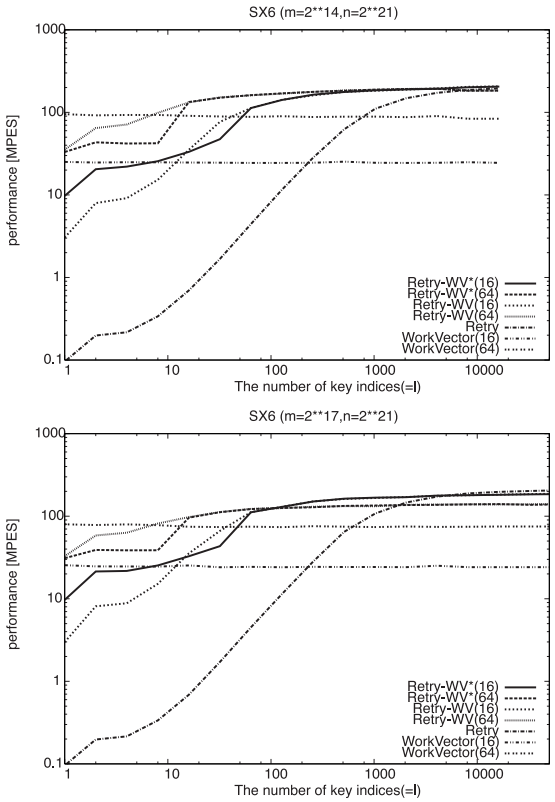

図 8 SX-6 におけるヒストグラムテストの性能結果，上：$(m,n)=(2^{14},2^{21})$，下：$(m,n)=(2^{17},2^{21})$

Fig. 8 Performance results of the histogram test on an SX-6, Top: $(m,n)=(2^{14},2^{21})$, Bottom: $(m,n)=(2^{17},2^{21})$.

使用したベクトル計算機の性能にも依存するが，結果は総じて次のようにまとめられる．

- n/m の異なる 2 ケースを設定したが，同一ベクトル計算機上での性能曲線はほぼ同じ結果である．
- Retry-WV 系アルゴリズムは Retry アルゴリズムを超える性能を示す．
- $l \leq 256$ でレジスタ可変の効果が現れており，レジスタ可変版は l が小さい領域でも性能の劣化は小さい．
- 同一のメモリ使用量，つまり同一の K を選択した場合，$l > 50$ では Retry-WV 系アルゴリズムが Work Vector アルゴリズムを超える性能を示すが，l が小さい領域では Work Vector アルゴリズムはレジスタ可変の Retry-WV アルゴリズムの 4 倍以上の性能を示す．
- m が増加した場合，Retry-WV 系アルゴリズムの最高性能が減少する傾向にある．Work Vector アルゴリズムにもあてはまるが，Retry-WV 系アルゴリズムの方が顕著である．

次に，各 Retry-WV 系アルゴリズムの K による性能依存性について調べるために，$(m,n)=(2^{14},2^{21})$ に限定し，K の値を変化させてみた．**図 9**，**図 10** に VPP5000，SX-6 それぞれにおける各アルゴリズムの測定結果を示す．実行性能の最良評価は評価式 (6)〜(8) から $S_* = n/T_*$ で求められ，次のようになる（ただし，$\gamma = \min(LK, \mathrm{N_v})$ とする）．

$$S_{\mathrm{WV}} = \frac{1}{(1/K)(\alpha+\beta K)+(2/n)(\alpha+\beta K m)} \tag{11}$$

$$S_{\mathrm{R\_WV}} = \frac{1}{(1/\gamma)(\alpha'+\beta'\mathrm{N_v})+(2/n)(\alpha+\beta K m)} \tag{12}$$

図 7 や図 8，図 9 や図 10 の上段と中段のグラフの傾向をこの評価式がよく表していることが分かる．たとえば，

- Work Vector アルゴリズムは K による加速の効果があるのだが，評価式からは K をいくらでも大きくとれるわけではなく，初期化・後処理か

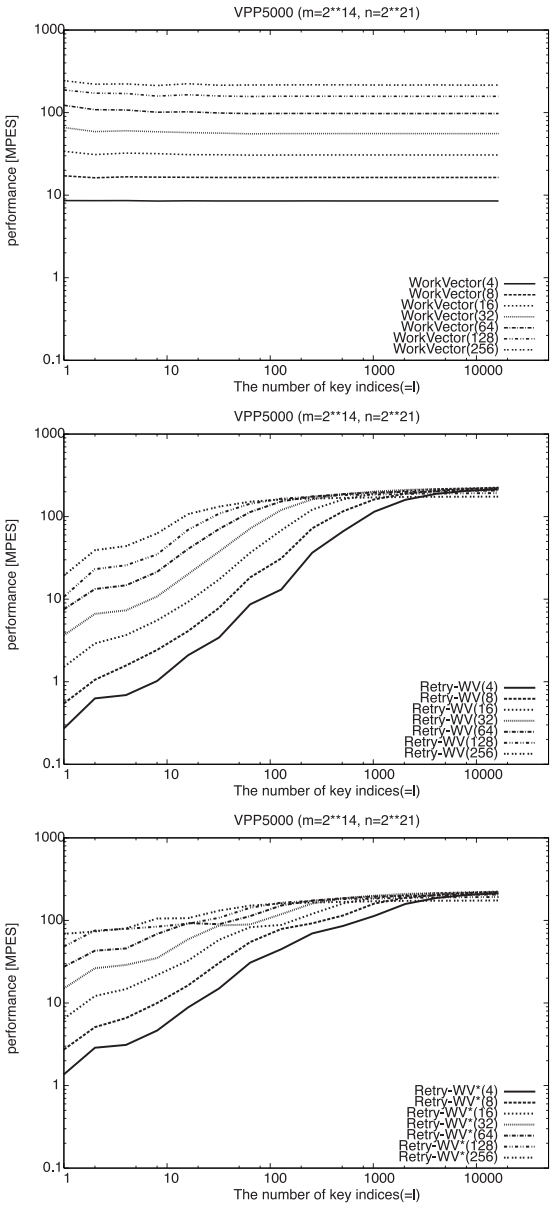

図 9 VPP5000 での各アルゴリズム性能の K 依存性（上段：Work Vector，中段：Retry-WV(K)，下段：Retry-WV*(K)）

Fig. 9 Dependency on the parameter K among three algorithms on a VPP5000 (Top: Work Vector, Middle: Retry-WV(K), Bottom: Retry-WV*(K)).

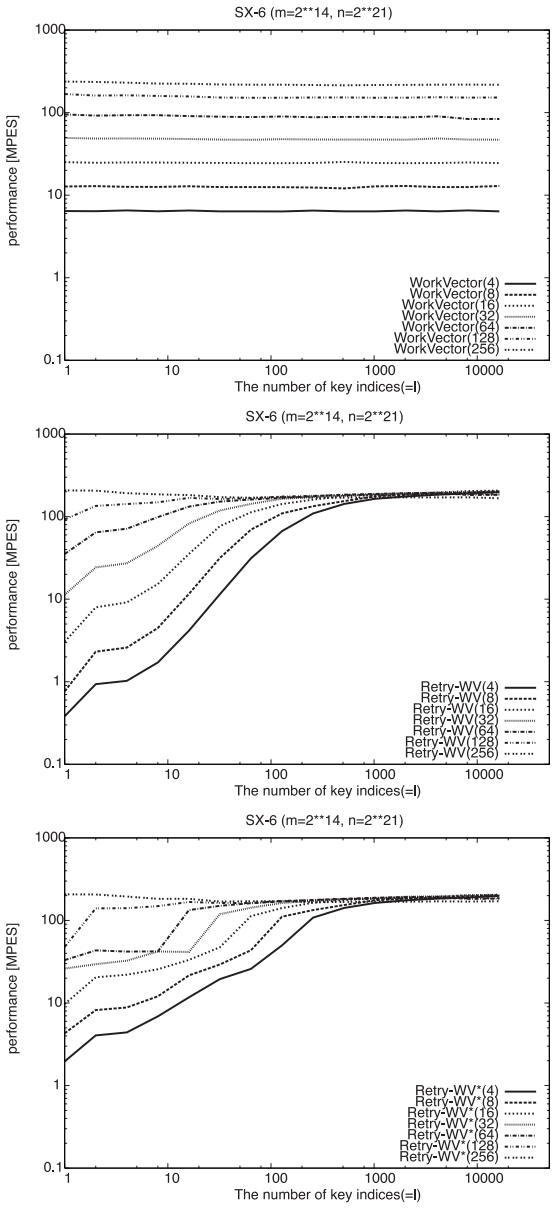

図 10 SX-6 での各アルゴリズム性能の K 依存性（上段：Work Vector，中段：Retry-WV(K)，下段：Retry-WV*(K)）

Fig. 10 Dependency on the parameter K among three algorithms on an SX-6 (Top: Work Vector, Middle: Retry-WV(K), Bottom: Retry-WV*(K)).

ら導入される式 (11) の分母内第 2 項の $K(m/n)$ が相対的に大きくなれば K の増加によって性能が落ちることがある．実際 VPP5000 において $(m,n) = (2^{17}, 2^{21})$ の場合，$K = 128$ で最高性能を記録するが，$K = 256$ では $K = 64$ 程度にしかならない結果を得ている．

- Retry-WV(K) アルゴリズム（レジスタ長固定版）では，$l = 1$ から傾きほぼ 1 で性能曲線が増加していき，K における最高性能で飽和している．この現象は，式 (12) での分子項 $\gamma = \min(LK, \mathrm{N_v})$ の L を l で置き換えれば理解できる．L はインデックスデータの周期を表すが，実際の処理ではレジスタ長単位でデータを分割するため局所的にはレジスタ内での要素数を周期と見なすことがで

表 4 分子軌道計算コード DIRAC の数値積分ルーチン実行時間（単位は秒），$m = 274^2 = 75{,}076$, $n = 610{,}722$

Table 4 Elapsed time of the numerical integral routine in DIRAC, a molcular dynamic simulation code (unit is in second), $m = 274^2 = 75{,}076$, $n = 610{,}722$.

	R-WV *(4)	R-WV *(8)	R-WV *(16)	R-WV *(64)	Retry	WV (4)	WV (8)	WV (16)	WV (64)	Scalar
SX-6	.089	.038	.028	.025	.37	.25	.13	.072	.029	.058
VPP5000	.076	.036	.024	.021	.54	.21	.11	.063	.023	.095
SX-5Be	.098	.067	.051	—	.38	—	.26	—	.051	.12
VPP300	.66	.30	.20	.15	4.6	.70	.38	.24	.14	.40

なお，SX-5Be の ‘—’ 部分は長時間の排他的利用ができなかったため未測定部分である．

表 3 分子軌道計算プログラム DIRAC の例題における初回処理要素数分布（左：VPP5000，右：SX-6）

Table 3 Distribution of the number of successful calculation in the first trial for the DIRAC MO code (Left: VPP5000, Right: SX-6).

処理要素数	(%)
1〜 50	0%
50〜 100	52.7%
100〜 150	44.7%
150〜 1,500	2.6%
1,500〜 2,048	0%
平均処理要素数	150.3

処理要素数	(%)
1〜 30	78.9%
30〜 100	18.4%
100〜 200	2.7%
200〜 256	0%
平均処理要素数	31.3

きる．評価式は最良ケースを想定するため，グラフ上では傾き 1 から飽和状態に至る時点で傾き 0 に漸近する．また，分母の第 2 項については Work Vector アルゴリズムとほぼ同様の性質を持つため，K が増加してもある時点で飽和しそれ以降は劣化することがある．$m = 2^{17}$, $n = 2^{21}$ の場合のグラフ（図 7 や図 8 の下部分）にその現象が現れている．

- Retry-WV(K) アルゴリズム（レジスタ長可変版）では，レジスタ長固定版と傾向は同様であるが，l が小さい領域でレジスタ長固定版よりも性能が高くなる傾向がある．しかし，K がレジスタ長程度まで大きくなると，その傾向は小さくなる．

5.3 分子軌道計算の数値積分での評価

最後に分子軌道計算プログラム DIRAC の積分ルーチン内に現れるインデックス分布を使用した数値実験結果を示す．この数値積分で用いられるインデックスは報告[6),7)] にもあるように，4 元数を基に計算を行うため，4 パラメータの組合せが現れインデックスの分布が局所的に偏ったものとなる特徴を持つ．実験に使用したデータのサイズは $m = 274^2$, $n = 610{,}722$ である．日本電気 SX-5Be，SX-6，富士通 VPP300，VPP5000 上☆での Retry-WV*($K = 4, 8, 16, 64$)，Retry，Work Vector ($K = 4, 8, 16, 64$) アルゴリズムの実行時間を**表 4** に示す．

表 3 に示したように，インデックスの分布が偏るために表 4 では，誤り検出率が非常に高く Retry アルゴリズでは性能が得られていない．一方，Retry-WV アルゴリズムでは局所的なインデックスの衝突を回避できるため，Retry アルゴリズムよりも優れた性能を出している．表 3 に示した 1 周目で処理される平均要素数を，局所的なインデックスの数 l と見なせば，VPP5000 では $l = 100$〜150，SX-6 では $l = 25$〜50 のあたりの性能を示すことが期待できる．図 7，8 から Retry-WV*アルゴリズムは Work Vector アルゴリズムとほぼ同等の性能を示しており，本問題でも同程度の性能を示していることが分かる．また，4.5 節で見積もった K の目安は，VPP5000 では $K = \mathrm{N_v}/L = 2{,}048/150 = 13.65$，SX-6 では $K = 256/31.3 = 8.17$ となる．同一の K をとる Work Vector と比較しても，その近傍の K に十分な効果が見てとれる．

VPP5000 での測定では $K = 128$，SX-6 では $K = 113$ で Work Vector アルゴリズムが最高性能を記録する．今，表内で最高で作業配列として確保可能な $K = 64$ について考えることにする．Work Vector アルゴリズム ($K = 64$) の 50%以上の性能を許容するならば，Retry-WV($K = 8$) もしくは ($K = 16$) 以上が実際の計算に利用できるパラメータ K の選択範囲になる．Retry-WV アルゴリズムで $K = 16$ を選択した場合，使用メモリは Work Vector アルゴリズム ($K = 64$) の 1/4 で済むため，Work Vector ($K = 64$) と比較すると，問題サイズ m を 4 倍にすることができる．実際，DIRAC は行列の成分計算を行うものであるから，行列の次元を 2 倍にすることが可能となり，高精度な計算を実現できるようになる．

6. ま と め

本論文では，ベクトル計算機上での高速な deposit 式

☆ SX-5Be はシュツッツガルト大学計算機センター RUS のものを使用．SX-6，VPP300，VPP5000 は日本原子力研究所計算科学技術推進センターのものを使用．

計算アルゴリズムについて述べた．従来手法の，Work Vector ならびに Retry アルゴリズムの問題点に言及し，Retry アルゴリズムにおけるアドレス衝突の回避が重要であることを示した．衝突回避の一手法として，Work Vector アルゴリズムで利用されている作業配列の導入と 2 次元拡張アドレスへのマッピングを行う Retry-WV アルゴリズム，さらに可変レジスタを導入し一般化した Retry 型アルゴリズム群を提案し，最良のコスト評価式を与えた．

以上の提案手法ならびに従来手法を，NPB IS の乱数ルーチンを用いたヒストグラム計算に適用したところ，Retry-WV(*) アルゴリズムは，Retry アルゴリズムよりも広い範囲で最良の結果を示した．特にキー数がレジスタ長以下になった場合は，Retry アルゴリズムの 2〜10 倍の性能を記録した．また，分子軌道計算での数値積分ルーチンに適用したところ，$K = 8$ の作業領域で Work Vector アルゴリズムにおける最高性能の 50%に達することが確認できた．この結果は，アドレスの衝突頻度が高くかつ大規模な問題に対して Work Vector アルゴリズム以外のベクトル化処理が可能になったことを意味している．同時に本アルゴリズムを適用し作業領域を削減することによって，より巨大な問題を計算することも可能となる．特に，プロセッサあたりの積載メモリが少ないベクトル機上で，本アルゴリズム群が有効と考える．

謝辞 本研究遂行にあたり，計算機環境をご提供いただいた日本原子力研究所計算科学技術推進センターならびにシュツッツガルト大学計算機センター（HLRS, RUS）の関係各位に感謝いたします．また，有用な助言をいただいた富士通株式会社折居茂夫氏，ならびに DIRAC のデータをご提供いただきましたアドバンスソフト望月祐志，日本電気情報システムズ松村昌幸両氏には心からお礼申し上げます．最後に，貴重なご意見をいただいた査読者各位にこの場を借りて感謝いたします．

参 考 文 献

1) Decyk, V.K.: Skeleton PIC Codes for parallel computers, *Computer Physics Communications*, Vol.87, pp.87–94 (1995).
2) Nishiguchi, A., Orii, S. and Yabe, T.: Vector Calculation of Particle Code, *Journal of Computational Physics*, Vol.61, pp.519–522 (1985).
3) Abe, Y.: Present Status of Computer Simulation at IPP, *Proc. Supercomputing 88*, Vol.II, pp.72–80 (1988).
4) Nishihara, K., Furukawa, M., Kawaguchi, M. and Abe, Y.: High accuracy particle-particle particle-mesh code and its application to laser-produced dense plasma, *Japanese Supercomputing*, Lecture Notes in Engineering, 36, Mendez, R.H. and Orsag, S.A., (Eds.), pp.59–72, Springer-Verlag (1988).
5) 村井 均，末広謙二，妹尾義樹：共有メモリ型ベクトル並列計算機上の高速整数ソーティングアルゴリズム，情報処理学会論文誌，Vol.39, No.6, pp.1595–1602 (1998).
6) 松村昌幸，望月祐志，与倉徹一，平原幸男，今村俊幸：相対論的分子軌道コード DIRAC における DHF 計算のベクトル化，情報処理学会研究報告，Vol.2001, No.49, pp.43–48 (2001).
7) Mochizuki, Y., Matsumura, M., Yokura, T., Hirahara, Y. and Imamura, T.: Vectorization of direct Fock matrix construction in DIRAC-DHF calculations, *Journal of Nuclear Science and Technology*, Vol.39, No.2, pp.195–199 (2002).
8) 折居茂夫：プラズマ粒子コードのためのベクトル並列計算法，プラズマ・核融合学会誌，Vol.75, No.6, pp.704–716 (1999).
9) 富田眞治：並列コンピュータ工学，昭晃堂 (1996).
10) Bailey, D., Harris, T., Saphir, W., Wijngaart, R., Woo, A. and Yarrow, M.: NAS Parallel Benchmarks 2.0, Technical Report NAS–95–020, (1995).

(平成 16 年 10 月 4 日受付)
(平成 17 年 1 月 24 日採録)

今村 俊幸（正会員）

1969 年生．1996 年京都大学大学院工学研究科応用システム科学専攻博士後期課程単位認定退学．同年日本原子力研究所入所．計算科学技術推進センターにて途切れのない思考を支援する並列処理基本システム STA の開発に従事．2001 年から 2002 年までシュツットガルト大学 HLRS にて招聘研究員．2003 年より電気通信大学講師．現在に至る．HPC とその周辺ソフトウェア，数値計算における並列・分散処理の研究に従事．博士（工学）．1999 年日本応用数理学会論文賞，同年石川賞企業部門受賞．日本応用数理学会，SIAM 各会員．