Vol. 35 No. 12 情報処理学会論文誌 Dec. 1994

# ベクトル計算機における拡張記憶の拡張主記憶としての仮想化

岡部寿男[†,*] 川端英之[†,**] 津田孝夫[†]

ベクトル計算機に実装されている拡張記憶を，コンパイラによるプログラム変換により仮想化する方法を提案する．記憶階層を意識せずに記述されたユーザプログラムから巨大配列の宣言や参照の情報を抽出し，巨大配列を拡張記憶に割り付け，主記憶上には配列データ用作業領域を確保して，プログラム中の適切な位置にデータ転送制御のためのコードを挿入するような変換を自動的に行う．データの分割方式として，参照コストが参照の方向によらない格子状の分割を採用する．格子状分割により分断されたデータに対してもベクトル化を可能とするため，リストベクトルを用いた間接参照を用いる．提案する手法の有効性を検証するために，プログラム変換を FORTRAN ソースコードレベルで行うプリプロセッサを開発した．ある LU 分解プログラムに適用して実測した結果，データ転送時間は総実行時間の約 20% に抑えられ，主記憶のみで実行した場合の約 50% の演算速度が得られた．

# Virtualization of Semiconductor Extended Storage as Extended Main Memory on Vector Supercomputers

YASUO OKABE,[†,*] HIDEYUKI KAWABATA[†,**] and TAKAO TSUDA[†]

This paper proposes a new method for virtualizing semiconductor extended storage (ES) as extended main memory by automatic program transformation done by a compiler, not by dynamic address translation mechanism supported by the OS. The automatic vectorizing compiler first extracts declarations of huge arrays and information on array references, allocates the huge arrays on ES, and work area on main memory (MM), and inserts codes for controlling data transfers between ES and MM. A huge array is divided into pages lattice-wise, so that vector reference at any direction of the array could be performed with the same cost. A vector of array elements on pages scattered on the work area by lattice-wise division can be accessed as a whole in vector mode by utilizing list-vector addressing. To confirm the availability of this method, a FORTRAN preprocessor which does the program transformation at source-code level has been developed. The experimental result on applying this preprocessor to an LU-factorization program tells us that the time for data transfer is about 20% of the total execution cost, and the computation speed is about 50% of that of the execution where all data were allocated on MM.

## 1. はじめに

近年，ベクトル計算機の演算性能は単一 CPU で 10 GFLOPS に迫ろうとしている．演算性能の向上に伴い，演算器と主記憶装置とのデータ転送速度の高さがこれまで以上に重要となる．しかし高速アクセスの可能な主記憶装置は技術的あるいはコスト的な要因からその実装容量に限界がある．主記憶に入り切らないデータを処理する場合，従来二次記憶として磁気ディスク装置が用いられてきた．磁気ディスク装置は，主記憶と比較してアクセス速度が非常に遅いため，両者の間でデータ転送が頻発する場合ベクトル計算機の高速演算性能を十分に引き出すことができない．そこで各社のベクトル計算機には拡張記憶と呼ばれる高速半導体補助記憶装置が装備されている[1),2)]．

主記憶・拡張記憶間のデータ転送は，主記憶・磁気ディスク装置間と比べ数百倍高速に行うことができる．データ転送回数や転送単位のサイズを最適化したアルゴリズムを設計し，巨大配列データを拡張記憶に割り付け計算の進行に応じて主記憶・拡張記憶間で

† 京都大学工学部情報工学教室
Department of Information Science, Faculty of Engineering, Kyoto University
* 現在，京都大学大型計算機センター
Presently with Data Processing Center, Kyoto University
** 現在，広島市立大学情報科学部情報工学科
Presently with Department of Computer Engineering, Hiroshima City University

データ転送を行うことにより，主記憶に入り切らない配列データを扱う計算が，データを主記憶上に配置した場合と同程度の計算時間で実行できる[3),4)]．しかし，主記憶・拡張記憶間のデータ転送制御をプログラム中に記述することは，ユーザにとっては大きな負担となる．

本論文では，主記憶のみを想定して記述されたユーザプログラムを，拡張記憶を利用するようにコンパイラのレベルで自動変換する方式を提案する．コンパイル時に目的コードにデータ転送命令が埋め込まれ，適切な時点でデータ転送を行いながら計算が実行される．これによりユーザは，拡張記憶と主記憶の間の記憶階層を意識することなく，主記憶に入りきらない巨大な配列データを扱う数値計算プログラムを実行させることができる．拡張記憶が主記憶の延長として区別なく扱えるという点で，一種の仮想化が実現される．

拡張記憶と主記憶の間のデータ転送はブロック転送で行われるので，巨大配列を適当なサイズの転送単位に分割する必要がある．本論文では，様々な配列参照の形態に柔軟に対応するために，格子状分割と呼ぶ分割方法を提案する．これは，短冊状分割と呼ぶ単純な分割[5)]に比べ，配列がどの次元方向へベクトル参照されても，参照される部分を含む分割単位をまとめて主記憶に読み込むことができるという点で効率が良い．さらに，格子状に分割することによりベクトル長が短くなることを避けるため，リストベクトルを用いたデータ参照方法を示す．また，データ転送制御のためのコードを，プログラムのループ構造を解析してデータ転送のコストおよびデータ転送の制御のオーバヘッドができるだけ小さくなるような位置に挿入する方法を示す．

本論文の提案の有効性を確認するために，巨大配列を扱う大規模数値計算プログラムを，拡張記憶を用いて実行するようにFORTRANソースレベルで自動変換するプリプロセッサを開発した．いくつかのプログラムにプリプロセッサによる変換を施し，それらの実行速度を測定した．あるベクトル計算機向きLU分解プログラム[6),7)]を用いて，3000×3000の行列についてその半分の大きさの作業領域を主記憶上に用意して実行したところ，HITAC S-3800/480上で約1GFLOPSの性能が得られた．これは同じプログラムを主記憶のみを用いて実行した場合の50%に達する値である．

以下，2章で準備としてベクトル計算機の拡張記憶装置について述べる．3章で拡張記憶の仮想化のためのプログラム変換について述べる．4章では変換をFORTRANソースレベルで自動的に行うプリプロセッサの実現について示し，5章では実際に変換を行い性能実測した結果を示す．

## 2. 拡張記憶

ベクトルスーパーコンピュータの多くの機種では，磁気ディスク装置と主記憶との間の容量およびアクセス速度のギャップを埋めるために，拡張記憶と呼ばれる高速半導体補助記憶装置を装備可能である．

拡張記憶はダイナミックRAMで構成され，スタティックRAMで構成される主記憶と比べてアクセス速度は一桁程度低いが，最大主記憶容量の数倍の大容量の実装が可能である．また，従来の磁気ディスク装置と比較して数百倍高速なアクセスが可能である．日立製作所のベクトル計算機HITAC S-3800に装備されている拡張記憶では，その最大容量は主記憶容量の8倍の16GBであり，主記憶と拡張記憶間のデータ転送速度は最大4GB/secである．**表1**に，現在の主要ベクトル計算機の拡張記憶のサポート方式を示す．

拡張記憶・主記憶間のデータ転送はブロック転送で行われる．このデータ転送は，ブロック長すなわち1回のREAD/WRITEで転送されるベクトルの量が，ある程度大きくなければ高速なデータ転送が行われないという特徴がある．HITAC S-820では8Mバイト程度，同じくHITAC S-3800/480では32Mバイト程度のブロック長によるデータ転送で，最大転送速度の8〜9割の速度に達する．

## 3. 拡張記憶の仮想化

### 3.1 プログラム変換による仮想化

主記憶に入り切らない多量のデータを扱うプログラムを実行する方法として従来広く用いられているものに，デマンドページングに基づく仮想記憶方式がある．しかし，ベクトル計算機においては，複数のベクトル命令の並列実行中にページフォールトが生じた場合，処理のある時点の状態の保存，中断および再開が困難であること，大規模数値計算においてはデータ参照の局所性は少ないといわれており[8)]，ページングが頻発する恐れがあることなどの理由から，通常の仮想記憶方式が採用されている例は少ない．

これに対し，本論文で述べる方式では，ユーザが拡張記憶と主記憶との記憶階層について意識することなく記述したプログラムを，言語処理系により拡張記憶

**表 1** 各社の拡張記憶の比較
Table 1 Specifications of extended storage on major vector computers.

| 機 種・名 称 | 最大容量<br>最大転送速度 | 主記憶容量 | 転 送 方 式 | サポート方式 |
|---|---|---|---|---|
| HITAC S-3800/480<br>拡張記憶 (ES) | 16 GB<br>4 GB/sec | 2048 MB | SPU による<br>同期／非同期転送 | FORTRAN<br>READ/WRITE |
| HITAC S-820/80<br>拡張記憶 (ES) | 12 GB<br>2 GB/sec | 512 MB | 同 上 | 同 上 |
| NEC SX-3/44<br>拡張記憶 (ES) | 16 GB<br>3 GB/sec | 2048 MB | SPU による<br>同期転送 | 同 上 |
| FACOM VP2600/20<br>システム記憶 | 8 GB<br>(不明) | 2048 MB | 同 上 | SSU 配列<br>および READ/WRITE |
| CRAY Y-MP8/8<br>半導体記憶 (SSD) | 4 GB<br>2.5 GB/sec | 1024 MB | 同 上 | SSD 配列<br>および READ/WRITE |
| FACOM VP400E<br>ベクトル記憶 | 0.67 GB<br>10 GB/sec | 256 MB | ベクトルレジスタへ<br>直接転送 | コンパイラが<br>自動割り付け |
| IBM ES/3090VF<br>拡張記憶 (ES) | 2 GB<br>0.1 GB/sec | 512 MB | SPU による<br>同期／非同期転送 | 仮想記憶 |

SPU：スカラプロセッサユニット

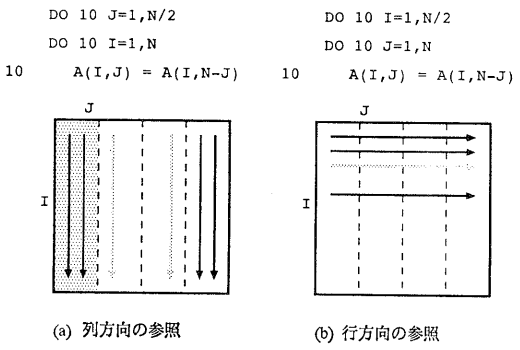
を利用するように自動変換する．対象としているプログラムは，主記憶に入り切らない巨大な配列を扱うもので，本処理系は巨大配列を二次記憶である拡張記憶に割り付けるように自動変換する．

ベクトル計算機では，ユーザプログラムはシステム側で提供される自動ベクトル化コンパイラによってコンパイルされた後実行されることが，事実上の前提となっている．自動変換の機能をコンパイラに組み込むことにより，大容量な拡張記憶の空間が主記憶と同じように扱えるという点において，一種の仮想化が実現される．

コンパイラレベルで拡張記憶を仮想化する他の方式として，富士通 VP 2000 シリーズのシステム記憶を用いた SSU 配列や Cray Y/MP における SSD 配列と呼ばれるものがある．拡張記憶との間のデータ転送命令がコンパイル時にオブジェクトに埋め込まれる点では，本論文で提案する方式と同じである．SSU 配列と本論文の方式との比較については 5.2 節で述べる．

### 3.2 巨大配列の分割

拡張記憶と主記憶の間のデータ転送はブロック転送で行われるので，巨大配列を適当なサイズの転送単位(以下**ページ**と呼ぶ)に分割する必要がある．FORTRAN では，2次元以上の配列の配列要素は，第1添字の順に連続して主記憶上に配置されると規定されている[9]．例えば2次元配列の場合，いわゆる列優先の配置となる．これに準じて，どれか一つの添字に着目し，多次元配列の要素を添字の値の増加に従って均等に分割し，分割単位（ページ）内では FORTRAN 式の番地づけをする方法を短冊状分割と呼ぶことにする．例として，2次元配列を第2添字に従って短冊状分割し，拡張記憶に配置した場合を考える (**図 1**)．あるループ中で配列参照が列方向に行われることが予測できれば，ループに制御が移る前に，当該列を含むページを主記憶中に読み込むことで，ループ中ではデータ転送を行うことなく配列参照を行うことができる（図1(a)）．これに対し配列参照が行方向に行われる場合には，参照される配列要素すべてをループの実行に先立って主記憶中に転送することができないので，ループ中で何度もデータ転送を行う必要が生じる．1行の参照であってもすべてのページが順々に主記憶に読み込まれることになり，多くの参照されない要素が転送される点で非効率的である．またループのベクトル化は，第2




(a) 列方向の参照　(b) 行方向の参照

**図 1** 配列の分割（短冊状分割）
Fig. 1 Data-division patterns and amount of date transfer.

添字の分割幅に相当する小さいベクトル長でしか行えない（図1(b)）.

このような配列の参照方向による実行速度の極端な偏りを避けるため，本論文では，配列をすべての次元方向に沿って分割する方式を提案する．この分割方法を格子状分割と呼ぶことにする（**図 2**）．ページ内の配列要素はFORTRANで採用されているような連続した番地付けを施す（2次元であれば列優先）.

巨大配列に対してある次元方向に沿った参照がある場合，参照される配列要素のベクトルを含むページすべてを主記憶に転送する．分割方向に関わらず，一つのベクトル参照において読み込まれるページは当該ベクトルを含むページのみである．さらに，繰り返し同一ベクトルを参照する場合や順々に隣接するベクトルを参照する場合には，データ転送を行うことなく直前の参照で読み込み済みのページ群がそのまま利用できる場合が多い.

### 3.3 リストベクトルを用いたデータ参照

格子状分割された配列をページごとに主記憶に転送した場合，本来等間隔に配置されるべき配列要素が，作業領域上では区分的にしか等間隔に配置されない（**図 3**）．主記憶中で等間隔に配置されていない配列要素への一連のアクセスはそのままではベクトル化できないので，単純なプログラム変換では，等間隔に配置されている区分ごとにループを分割しなくてはならず，配列参照時のベクトル長が小さくなる.

これに対処するため，主記憶上に転送されたデータに対してリストベクトル（インデックスベクトル）による間接参照を用いることを提案する．主記憶上では物理的に等間隔に配置されていない配列要素のベクトルに対し，それらのアドレスのベクトルを保持したリストベクトルを用意し，ベクトル間接ロード／ストア命令を用いて参照する．これにより，大きいベクトル長のままでのベクトル参照が可能となる（図3）．今日の多くのベクトル計算機では，高速なベクトル間接参照が可能である[10].

ある次元の方向に沿ったループ中での配列参照があった場合，参照される配列要素を含むページを主記憶に転送し，配列要素を間接参照するためのリストベクトルを設定する．参照される配列要素を含むページが以前のデータ転送により既に主記憶中に存在している場合にはデータ転送は不要であるが，リストベクトルの再設定は必要である．そこで次のようにして間接参照時のリストベクトル設定のオーバヘッドを抑えることを考える．**図 4**(a)のようなプログラムを考える．配列 **A** が $m \times m$ の小行列に格子状に分割されているとすると，**J** が 1, $m+1$, $2m+1$, … の時点で，**I** のループの外側でデータを転送すればよいことが分かる．まず始めに，データ転送を行うと同時に，参照される一連の配列要素の間接参照のためのリストベクトルを設定する．その後 **J** が $m$ 回インクリメントされるまでデータ転送は不要であるが，リストベクトルの各要素の値は **J** がインクリメントされる度に再設定されなくてはならない（図4(b)）．ここで，新たにリストベクトルの各要素をインクリメントする代わりに，スカラ変数に増分値を保持しておき（図4(c) **DIFF**）．間接参照の時点でリストベクトルの示す値にその増分値を加算して実際のインデックスを求めることにする（図4(c)）．この方法により，リストベクトル設定に伴うベクトルロード・ストアの回数が削減される．例えば，図4(c)では，ページ境界に

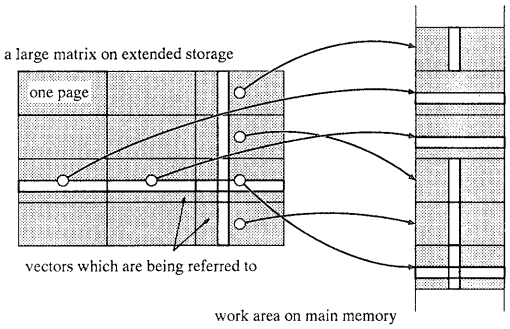



**図 2** 配列の分割（格子状分割）
Fig. 2 Multi-dimensional partitioning by lattice.

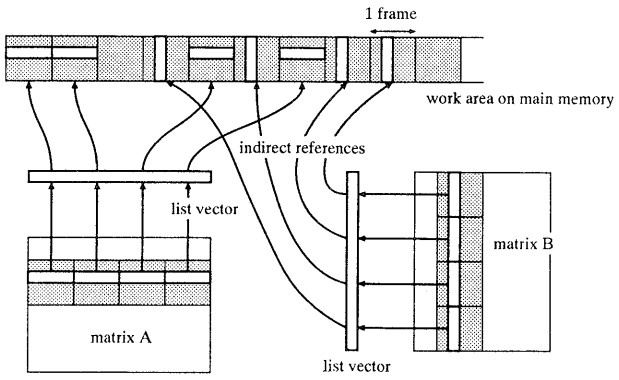



**図 3** 作業領域の共有
Fig 3 Sharing workarea on main memory.

沿う4つのベクトル（図中★印）に対するリストベクトルの設定だけで十分である．設定したリストベクトルをベクトルレジスタ上に保持し続けることが可能ならばリストベクトルのロードが不要になるため，間接参照の採用により生ずるオーバヘッドは演算時間に対して十分小さい値になると考えられる．

### 3.4 データ転送制御のためのコードの挿入

ベクトル演算では，ベクトルロードが行われる時点で参照される配列要素すべてが主記憶中に存在している必要がある．プログラムの各文で実行時に参照する配列要素をコンパイル時に完全に決定するのは不可能であるが，参照しようとするデータを含むページが主記憶上に存在するかどうかの判定は実行時のオーバヘッドを伴う．そこで，FORTRANにおけるDOループに代表される繰り返し構造（以下単にループと呼ぶ）に着目し，判定のためのコードをループのできるだけ外側に追い出すことにより，判定のためのコストが相対的に小さくなるようにする．

#### 3.4.1 参照列添字によるページ群の特定

ループ中での配列参照において，配列参照の各次元の添字のうち一つの添字を除き，添字式の値がそのループ中で不変である場合，配列はその添字の次元方向に沿って参照が行われる．以下このときの添字を**参照列添字**と呼ぶことにする．

プログラムの実行時において，参照列添字以外の添字式の値がすべて確定する時点で参照される配列要素の列が判明する．例えば列優先で割り付けられた2次元配列のループ中での参照において，最内側ループの制御変数が第2添字にのみ現れており第1添字はループ不変式である場合，最内側ループに入る前の第1添字の値が確定する時点で何行目を参照するかが判明する．そこで，被参照列が確定した時点で，それを含むページ群に対し，各ページが主記憶上にあるかどうかの検査，主記憶にないページの読み込み，および前節で述べたリストベクトルの更新の操作を行うことにする．読み込んだページ群は参照の直後で解放する．この際，参照列添字以外の添字式が同じであるような同一配列の複数の参照については，添字式に出現する変数の値が同じであることが保証されている限り一括して扱い，同一参照列に対する操作が繰り返されるのを避ける．すなわち最初の参照の直前でページ群を主記憶上に確保し，最後の参照の直後で読み込んだページを解放する．最内側ループ内にはデータ転送のためのコードは挿入されないので，変換後もループのベクトル化可能性は維持される．

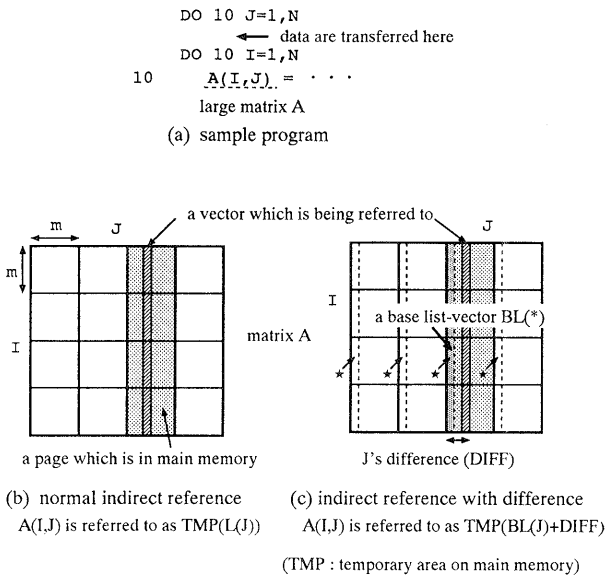



図4 リストベクトルの設定方式
Fig. 4 Generating list vectors.

配列要素が多重ループ中に出現しかつ参照列添字以外の添字式がループの不変式である場合には，データ転送制御に関するコードをループの外側に移動し，ループの直前で参照列を含むページ群を主記憶上に用意しリストベクトルと対応付け，ループ脱出後（ループの直後および飛び出しのあるループの場合には飛び出し直前）にページ群を解放するようにすることにより，同一参照列に対しデータ転送制御のための操作が繰り返されることを避けるようにする．

#### 3.4.2 DO型ループの場合の最適化

転列要素が多重ループ中に出現し，参照列添字以外の添字式の値が変化する最も内側のループが，FORTRANにおけるDOループのように制御変数値が繰り返しごとにある増分値だけ変化するループ（以下DO型ループと呼ぶ）である場合を考える．

そのループで変化する添字式すべてがそのループの制御変数の線形式☆でありそのループ中で値の変化する他の変数を含まない場合には，参照列は当該DO型

☆ 帰納変数の線形式で，制御変数に帰着できるものを含む．

ループの繰り返しに従って一定間隔ずつ順に（典型的には隣接の列に）移動する．格子状分割においては，参照列が隣ないし小さな間隔で移動しても，ページ境界を跨がない限り参照列を含むページ群は同一である．前述の方法を単純に適用すると，ページ群が同一の場合でも一旦解放されてしまうため効率が悪い．そこでこのような DO 型ループの場合には，ループの繰り返しの1回目に参照列を含むページ群の用意とリストベクトルとの対応付けを行い，繰り返しごとに新しい参照列が既に用意されているページ群に含まれるかの判定を行うようにする．この判定は，添字式の値とページ境界に対応する値の単純な比較で行え，かつページ群がそのまま利用できる場合には，前節で述べたようにしてリストベクトルに加える増分値の更新のみを行えばよいので，単純な方法に比べ効率がよい．例えば図4の場合，**I**のループに入る前にデータ転送が必要となる可能性があるが，データ転送を行う必要の有無は**J**の値のみから判定することができる．ページ群の解放はループ脱出直後に行う．

最内側ループが FORTRAN における DO ループのようにループに制御が移る以前に制御変数の上下限式が確定するようなループにおいて，その配列参照の参照ベクトル添字が最内側ループの制御変数の線形式の場合には，被参照列のうち実際に参照される要素をも特定できるので，当該参照列のうち実際に参照される要素を含むページのみを転送するようにすることにより参照されないデータが転送されることを避けることができる．

#### 3.4.3 不規則な配列参照

配列の参照がループ中で一つの次元方向に沿ったものでない場合には，次のような方法でページ群とリストベクトルとを対応付ける．配列参照時の最内側ループが DO 型ループで，ループ中で変化する添字式がすべて制御変数の線形式の場合，制御変数の上下限式からループ1回の実行で必要となるページを計算し，ループの外側でまとめて転送する．DO 型ループ以外の場合や配列参照時の添字式が非線形式を含む場合には，添字式のみを計算して参照される要素が存在するページの特定する前半のループと，実際の配列参照を行う後半ループにループを分割することによりループの最内側におけるデータ転送を避けループのベクトル実行可能性を保つことができる．

これらの方法が適用できない場合，および配列の単一要素に対する参照の場合には，参照する単一要素を含むページが特定できた時点で単一ページについてデータ転送を行う．

### 3.5 作業領域管理

巨大配列は，格子状に分割し拡張記憶に割り付ける．主記憶上にはデータを転送して参照するための作業領域を設ける．作業領域は，フレームと呼ぶ区画に分割する．各ページは任意のフレームに転送可能とする（図3）．

配列要素の参照に先だって，参照される配列要素を含むページを特定し，主記憶に存在しないものの数だけ空きフレームを取得してデータ転送を行う．参照中のデータが存在するページを別の配列参照で同時に参照する場合は，フレームを共用する．各フレームにはフレーム中のデータがいくつかの参照式から参照されているかを表す参照カウンタを設ける．配列参照とデータとの対応がとれた時点でカウンタをインクリメントし，一つの参照式からの配列参照が終了した時点でそのフレームのカウンタをデクリメントする．カウンタが0になれば空きフレームであるとみなす．空きフレームは空きフレームリストにつなぎ，その時点ではフレーム中のデータの書き戻しや破棄は行わない．空きフレーム中に存在するデータを再び参照する必要が生じた場合のデータ転送は不要である．

空きフレーム中に存在するデータは，そのフレームが別の配列参照用に取得された時点で，データが更新を受けていた場合は拡張記憶にページを書き戻し，受けていない場合は破棄する．空きフレームの取得は最も過去に使用が終了したフレームから行う．

## 4. ソースレベルでの自動変換

### 4.1 ソースレベルでのプログラム変換

提案する拡張記憶の仮想化手法の実現として，3章で述べた方法に基づき，拡張記憶を意識することなく記述されたプログラムを，拡張記憶を利用するようにソースレベルで自動的に変換するプリプロセッサを開発した．プログラム変換の対象としているのは，主記憶に入りきらないような巨大配列を扱う大規模数値計算プログラムである．言語は FORTRAN を対象にしている．

ユーザは基本的には拡張記憶に関して考慮することなくプログラムを記述すればよい．ユーザが書いたプログラムに対し，ソースレベルでのプログラム変換により，拡張記憶への巨大配列の割り付けやデータ転送制御を行うための命令を付加する．これを通常のベク

トル化コンパイラでコンパイルし実行する．

プリプロセッサによって以下のようなプログラム変換が行われる．

1. 巨大配列の宣言文を検出し，拡張記憶に割り付けるように宣言を変更する（拡張記憶上にオープンするファイルの宣言と主記憶に設ける作業領域の宣言文を挿入）．
2. 巨大配列の参照文を検出し，主記憶上に設ける作業領域への参照に置き換える（リストベクトルを用いた間接参照のための一連の処理を実行する文の挿入）．
3. 巨大配列の参照文から，データ転送を行うべき位置を決定し，データ転送ルーチンの呼び出し文，および各種補助関数の呼び出し文を挿入する．また，それらの補助関数のパラメータを設定し，ソースプログラムに付加して出力する．

ソースレベルでのプログラム変換であるため，既存のベクトル化コンパイラの最適化能力を利用することができる．また，変換後のプログラムをユーザが直接修正することも可能である．

ユーザは，拡張記憶に割り付けるべき巨大配列の指定や作業領域の宣言を設定するための各種のパラメータの指定を，プリプロセッサ指示行（特別な形式のコメント文）を用いて行うことができる．

### 4.2 データ転送制御ルーチン

データ転送を行うためのルーチン群を用意しておき，それらの呼び出し文を挿入する位置，およびこれらのルーチンの呼び出し文の引数を，プリプロセッサで決定する．

作業領域のフレーム中のデータおよびリストベクトルは，プリプロセッサによって巨大配列の参照式ごとに与えられる式番号で区別する．参照式ごとに式番号を確保し，その式番号と対応づけて作業領域を確保する．その参照式での配列参照が終了した時点で式番号は解放し，以降の配列参照式で再利用する．

データ転送ルーチンには以下の情報を引き渡す．

- 配列番号
- 配列参照式番号
- 参照される配列の添字式の，各次元の値
- データを拡張記憶に書き戻す必要の有無を示すフラグ
- 最後に確定する添字式の，上下限式の値

巨大配列は拡張記憶に固定レコード長の直接アクセスファイルとして割り付ける．複数の巨大配列を区別するために，このファイルの装置番号を配列番号として用いる．

式番号の開放，すなわちそれに対応する作業領域の区画の開放は，そのためルーチンを呼び出すことによって行う．

### 4.3 プログラム変換例

**図5**に，プログラム変換の例を示す．

変換前の図5（a）の4，5行目の参照文に現れる配列は拡張記憶に割り付けられ，変換後には主記憶上の作業領域（一次元配列 **XXTT**）の参照文に変更される（図5（b），15, 16行目）．配列 **A** についてははじめから8列目の参照であることが分かるため，最外側のループの外でデータ転送を行うことが可能である（図5（b），3行目）．配列 **B** については，最後に定まる添字式が2次元目の **J** であり，図5（a）の1行目を過ぎた時点で配列の **I** 行目に対するデータ転送を行うことができる（図5（b），1, 2, 5〜11行目）．とくに配列 **B** は，制御変数 **I** が添字式に含まれるため，最外側のループ中で **I** の値によりデータ転送の必要性の有無を判断することができる．実際にデータ転送が必要な時点になってからデータ転送ルーチンを呼び出すと

```
1        DO 10 I=I0,I1
2        DO 10 K=K0,K1
3        DO 10 J=J0,J1
4            A(K,8)=1.0D0
5            B(I,J)=3.0D0
6   10   CONTINUE
```

(a) Before transformation.

```
 1       NXTBLK(2)=(I0-1)/BLWD(B,1)
 2       CURBLK(2)=-1
 3       CALL XXCL(A,1,1,8,1,.TRUE.,1,K0,K1)
 4       DO 10 I=I0,I1
 5       IF(NXTBLK(2).NE.CURBLK(2)) THEN
 6          CALL XXCL(B,2,I,1,1,.TRUE.,2,J0,J1)
 7          CURBLK(2) = NXTBLK(2)
 8       ELSE
 9          XXDIFF(2)= I-LASTI1(2)
10       ENDIF
11       NXTBLK(2) = ((I+(1))-1)/BLWD(B,1)
12       DO 10 K=K0,K1
13  *VOPTION INDEP(XXTT), VIST
14       DO 10 J=J0,J1
15          XXTT(XXI(1,K)+XXDIFF(1))=1.0D0
16          XXTT(XXI(2,J)+XXDIFF(2))=3.0D0
17   10   CONTINUE
```

(b) After transformation.

**図5** プログラム変換例
Fig. 5 An example of source-level transformation.

いうことが可能である（図5（b），6行目）．

## 5. 評価および考察

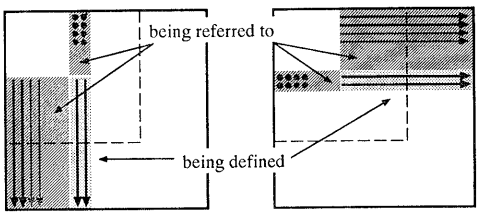
### 5.1 性能実測

提案する手法を評価するため，前節で述べたプリプロセッサを，FORTRAN で記述した LU 分解および行列積プログラムに適用し，実測を行った．以下，表2，3および4中に現れる CPU 時間，VPU 時間はそれぞれスカラプロセッサユニット，ベクトルプロセッサユニットの動作時間を表す．拡張記憶・主記憶間のデータ転送はスカラプロセッサユニットで処理されるので，転送時間は CPU 時間に含まれる．

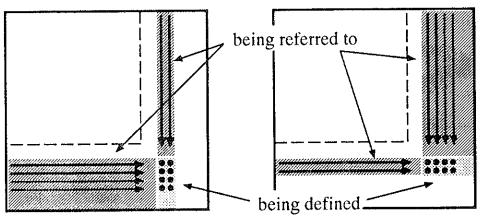
#### 5.1.1 LU 分解

サイズが3000×3000の行列の LU 分解を HITAC S-3800/480 上で行った実測結果を**表2**に示す．測定に用いたプログラム[6),7)]は2行2列を同時に分解するクラウト法を用いており，配列に対して縦方向の参照と横方向の参照を同時に行うものである（**図6**）．ベクトル実行に対する整合性は非常に高い．201×201(323 KBytes)，301×301（725 KBytes），401×401（1.3 MBytes）の三種類のページサイズで測定した．

なおここで扱った配列は全体を主記憶中に配置することが可能な大きさであった．表の最下行は，処理系を適用する前のプログラムをそのまま主記憶のみを使用して実行した結果である．表の左から二列目は，実行時の主記憶使用量およびその行列全体に占める割合を示している．また，右端の列は，各実行において計測された演算速度，および各演算速度を主記憶のみで実行した場合に得られた演算速度との比較を示している．巨大配列の半分の大きさの作業領域を主記憶に用意することにより，主記憶のみの場合の2倍程度の時間で実行された．また，巨大配列の 20% の大きさの作業領域でも，600 MFLOPS を超える演算速度が計測された．




(a) Reference patterns of the first half.




(b) Reference patterns of the latter half.

**図6** LU 分解におけるデータ参照パターン
Fig. 6 Data reference patterns of the LU-factorization.

**表2** LU 分解実行時間測定結果（1）
Table 2 Performance of LU-factorization on HITAC S-3800.

行列のサイズ：3000×3000

| ブロックサイズ MByte | 主記憶使用量 MByte | CPU 時間 sec | | | 転送速度 GB/sec | 転送量 GByte | 演算速度 Mflops |
|---|---|---|---|---|---|---|---|
| | | | VPU | 転送 | | | |
| 201×201 (323 KBytes) | 14.5( 20%) | 27.6 | 11.7 | 14.3 | 2.57 | 36.8 | 653( 29%) |
| | 19.4( 27%) | 24.4 | 11.7 | 11.1 | 2.52 | 28.0 | 739( 33%) |
| | 24.2( 34%) | 22.2 | 11.7 | 8.74 | 2.54 | 22.2 | 811( 36%) |
| | 29.1( 40%) | 20.1 | 11.7 | 6.35 | 2.54 | 16.1 | 896( 40%) |
| | 33.9( 47%) | 18.3 | 11.7 | 4.38 | 2.53 | 11.1 | 986( 43%) |
| 301×301 (725 KBytes) | 21.7( 30%) | 23.9 | 11.2 | 10.7 | 3.00 | 32.1 | 753( 33%) |
| | 29.0( 40%) | 19.9 | 11.2 | 6.51 | 3.00 | 19.5 | 905( 40%) |
| | 36.2( 50%) | 17.4 | 11.2 | 3.93 | 3.00 | 1.28 | 1035( 46%) |
| 401×401 (1286 KBytes) | 30.9( 43%) | 23.4 | 11.1 | 10.7 | 3.20 | 34.3 | 769( 35%) |
| | 41.2( 57%) | 18.7 | 11.1 | 5.29 | 3.20 | 16.9 | 962( 43%) |
| | 51.5( 72%) | 15.6 | 11.1 | 2.25 | 3.19 | 7.16 | 1152( 51%) |
| (主記憶のみ) | 72.0(100%) | 7.98 | 6.89 | — | — | — | 2250(100%) |

1993年10月東京大学大型計算機センター S-3800/480 にて測定

S-3800では，ハードウェア的には，データ転送はスカラプロセッサによってベクトル演算と並行に行うことができる．演算とデータ転送の並列実行がユーザレベルで実現されれば，さらなる速度向上が期待できる．

### 5.1.2 行 列 積

サイズが2000×2000である二つの行列の行列積をHITAC S-3800/480上で計算した実測結果を**表3**に示す．201×201(323 KBytes)，301×301(725 KBytes)の二種類のページサイズで測定した．測定に用いたプログラムは**図7**のように添字の記述をしたもので，配列**A**，**B**，**C**とも拡張記憶に割り付ける方法と，配列**C**あるいは配列**A**を主記憶中に配置した場合とで測定を行った．前節と同様に，拡張記憶を用いずに主記憶のみで実行した結果を表の最下行に示す．

扱う巨大配列の総量の10%の大きさの作業領域を主記憶に用意した場合，主記憶のみを用いた場合の12%の演算速度が得られた．また，データ転送速度とページサイズとの関係は，表2と同様である．

図7における配列**C**は，内側の二重ループで全体が参照され，三つの配列のうちで最も多くデータ転送が行われる．配列**C**全体を主記憶上に配置できる場合にはデータ転送量が大幅に削減され，実行速度が飛躍的に向上している．これに対し，配列**A**を主記憶に配置しても，実行速度の向上にはつながっていない．

### 5.2 他の自動化方式との比較

拡張記憶と主記憶との間のデータ転送を自動化する他の方式として，富士通VP2000シリーズのシステム記憶を用いたSSU配列と呼ばれるものがある．SSU配列上の連続する要素に対するベクトル参照では連続転送と呼ばれる極めて高速なデータ転送が行われる．しかし，SSU配列の使用にはさまざまな制限が伴うだけでなく，配列要素を一定間隔でベクトル参照する場合（多次元配列の参照で第1添字以外の添字を順に増やす場合を含む）やスカラデータとして参照する場合には，ディスタンス転送や要素転送とよばれるより遅い転送方式が用いられ，データ転送に演算時間の数十倍を要することもある[11]．これに対し本論文の方法は，格子状分割により参照の方向によらず同一のアクセスコストであること，配列参照の出現ごとにデータ転送を行うのではなくループ構造に着目してデータ転送回数ができるだけ少なくなるようにしていること，などの点では優れている．

SSU配列を用いた場合と本論文の方式との性能の比較のために，富士通VP2600を用いて実測を行った．3000×3000の行列のLU分解の実行結果を**表4**に示す．測定に用いたプログラムは5.1節のものと同様である．表の各行はそれぞれ上から順に，本論文で提案する手法を用いた301×301のページサイズによるページ分割での実行，ページ分割を行いかつ拡張記憶・主記憶間のデータ転送にSSU配列の自動転送機能を応用したもの，SSU配列を用いた実行，お

```
      DO 10 I=1,N
        DO 10 J=1,N
          DO 10 K=1,N
            A(I,J)=A(I,J)+B(I,K)*C(K,J)
10      CONTINUE
```

**図7** 行列積プログラム
Fig. 7 Matrix multiplication.

**表3** 行列積実行時間測定結果
Table 3 Performance of matrix multiplication.

行列のサイズ：2000×2000

| ブロックサイズ MByte | 主記憶使用量 MByte | CPU時間 sec | | | 転送速度 GB/sec | 転送量 GByte | 演算速度 Mflops |
|---|---|---|---|---|---|---|---|
| | | | VPU | 転送 | | | |
| 201×201 (323 KBytes) | 9.70( 10%) | 47.1 | 20.6 | 25.4 | 2.55 | 64.9 | 340( 12%) |
| | 38.8 ( 40%) | 35.6 | 20.6 | 12.8 | 2.54 | 32.5 | 450( 16%) |
| | 38.5 ( 40%)† | 17.8 | 16.1 | 0.0638 | 2.48 | 0.158 | 898( 33%) |
| | 38.5 ( 40%)‡ | 47.8 | 16.8 | 25.5 | 2.55 | 64.8 | 335( 12%) |
| 301×301 (725 KBytes) | 21.7 ( 23%) | 42.5 | 19.8 | 20.3 | 3.01 | 61.1 | 376( 14%) |
| | 60.9 ( 63%) | 29.7 | 19.8 | 6.86 | 2.98 | 20.5 | 539( 20%) |
| | 42.2 ( 44%)† | 17.4 | 15.4 | 0.593 | 2.91 | 0.173 | 921( 34%) |
| | 42.2 ( 44%)‡ | 44.5 | 16.1 | 23.7 | 3.00 | 71.2 | 360( 13%) |
| (主記憶のみ) | 96.0 (100%) | 5.83 | 4.10 | — | — | — | 2746(100%) |

† 配列Cを主記憶に配置，‡ 配列Aを主記憶に配置
1993年10月東京大学大型計算機センター S-3800/480 にて測定

よび主記憶のみを用いた実行の結果を示している．

表4の上から3番目は SSU 配列をそのまま用いた場合の結果であるが，VPU 時間がほかと同様であるのに対して総実行時間が非常に長い．これは，測定に用いたプログラムが配列の行方向参照を多用しており，転送速度の遅いディスタンス転送が頻出するためと考えられる．SSU 配列の使用には格別の注意を払う必要があり，少なくとも現時点では，主記憶の延長として自然に用いることができるとは言い難い．

**表 4** LU 分解実行時間測定結果（2）
Table 4 Performance of LU-factorization on FACOM VP 2600.

行列のサイズ：3000×3000

| ブロックサイズ MByte | 主記憶使用量 MByte | CPU 時間 sec | | 演算速度 Mflops |
|---|---|---|---|---|
| | | | VPU | |
| 301×301 | 29.0( 40%) | 54.7 | 19.5 | 329 ( 30 %) |
| 301×301† | 29.0( 40%) | 119 | 23.4 | 151 ( 14 %) |
| SSU 配列 | — | 314 | 24.9 | 57.3( 5.2%) |
| (主記憶のみ) | 72.0(100%) | 16.3 | 16.3 | 1101 (100 %) |

† 拡張記憶への巨大配列の割り付けに SSU 配列を利用
1993年4月京都大学大型計算機センター VP 2600 にて測定

表4の上から2番目は，データ転送を READ/WRITE 入出力文で行わず，SSU 配列の自動転送機能を利用したものである．すなわち，巨大配列を拡張記憶上に SSU 配列として割り付け，DO ループにより SSU 配列から作業用配列に代入あるいは作業用配列から SSU 配列に代入することでデータ転送を行った．データ転送はすべて連続転送であるが，表4の上から1番目の直接 READ/WRITE 文を用いた場合の方が，高速に実行されている．

## 6. まとめ

本論文では，大規模数値計算を意識した拡張記憶の新しい利用法について提案した．本手法では従来の OS による仮想記憶方式と異なり言語処理系レベルで記憶階層の隠蔽を行うため，プログラムの制御構造に応じたデータ転送制御が可能である．これにより，二次記憶装置である拡張記憶がユーザにとって主記憶の延長として仮想化され，主記憶に収まらない巨大配列データを扱う大規模数値計算を容易に高速に行うことが可能となる．

提案するプログラムの自動変換方式では，配列の格子状分割とデータの間接参照を組み合わせた手法により，拡張記憶の仮想化を様々な配列参照に対して有効に適用できる．提案する手法の一つの実現として，変換を FORTRAN 言語ソースレベルで行うプリプロセッサを開発した．いくつかのプログラム例に対しプログラム変換を行い性能実測を行った結果，データ転送時間が総実行時間に対して十分低く抑えられ，主記憶を超えるデータサイズの大規模数値計算が自動変換によっても実用的な性能で実行できることが確認できた．

今後の課題としては，データの分割方式や参照方式を可変にし，問題に応じて適用方式を自動選択することが挙げられる．具体的には，配列の短冊状分割と格子状分割の融合，直接参照あるいは間接参照などの参照方式の状況に応じた適用などが挙げられる．また，データのプリフェッチによる演算とデータ転送のオーバラップ，ループアンローリングやループ交換など，プログラムの制御構造の変更を含めたデータ転送の最適化が望まれる．

**謝辞** 日頃から御討論頂く本学國枝義敏助教授をはじめ津田研究室の諸氏に深謝します．

## 参考文献

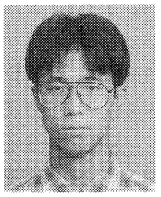
1) Okada, T., Nagashima, S. and Kawabe, S.: Hitachi Supercomputer S-810 Array Processor System, *SUPERCOMPUTERS Class IV Systems, Hardware and Software*, pp. 113-136, Elsevier Science Publishers (1986).
2) Watanabe, T., Katayama, H. and Iwaya, A.: Introduction of NEC Supercomputer SX System, *Ibid.*, pp. 153-167.
3) Tsuda, T. and Okabe, Y.: Use of Semiconductor Extended Storage as Extended Main Storage for Large-Scale Supercomputing, *Proc. 2nd International Conference on Supercomputing*, pp. 176-183 (May 1987).
4) Tsuda, T. and Seo, Y.: Supercomputing External Multidimensional FFT—Use of Semiconductor Extended Storage as Extended Main Storage, *J. Inf. Process.*, Vol. 11, No. 2, pp. 112-119 (1988).
5) 大西重行，岡部寿男，津田孝夫：スーパーコンピュータ用拡張記憶の拡張主記憶としての高度利用，第39回情報処理学会全国大会論文集，pp. 1275-1276 (1988).
6) 呉 永化：VP 向き二列同時消去 LU 分解法，京都大学大型計算機センター第2回ベクトル計算機応用シンポジウム論文集，pp. 72-77 (1986).
7) 津田孝夫：数値処理プログラミング，岩波講座

ソフトウェア科学9, 3.5節, pp. 126-130, 岩波書店 (1988).
8) Abu-Sufah, W. A. and Padua, D. A.: Some Results on the Working Set Anomalies in Numerical Programs, *IEEE Trans. Softw. Eng.*, Vol. SE-8, No. 2, pp. 97-106 (1982).
9) 森 正武: FORTRAN 77 数値計算プログラミング (増補版), 岩波コンピュータサイエンス, 第2.3節, pp. 24-26, 岩波書店 (1986).
10) Uehara, T. and Tsuda, T.: Benchmarking Vector Indirect Load/Store Instructions, *SUPERCOMPUTER*, No. 46, pp. 15-29, ASFRA, the Netherlands (Nov. 1991).
11) 永井 亨: ベクトル計算機入門(3)—SSU, 名古屋大学大型計算機センターニュース, Vol. 23, No. 3, pp. 205-213 (1992).

(平成5年11月25日受付)
(平成6年9月6日採録)

**岡部 寿男** (正会員)

1964年生. 1986年京都大学工学部情報工学科卒業. 1988年同大学院修士課程修了. 同年京都大学工学部助手. 現在京都大学大型計算機センター研究開発部助教授. 京都大学博士 (工学). 並列計算の理論, スーパコンピュータ用基本ソフトウェア等の研究に従事. 電子情報通信学会, 日本ソフトウェア科学会, IEEE, ACM, EATCS 各会員.



**川端 英之** (正会員)

1969年生. 1992年京都大学工学部情報工学科卒業. 1994年同大学院工学研究科情報工学専攻修士課程修了. 同年4月より広島市立大学情報科学部情報工学科助手. 自動ベクトル化／並列化コンパイラの研究に従事. 日本ソフトウェア科学会会員.

**津田 孝夫** (正会員)

1957年3月京都大学工学部電気工学科卒業. 現在京都大学工学部情報工学科教授. 計算機ソフトウェア講座担当. 工学博士. 自動ベクトル化／並列化コンパイラ, スーパコンピューティング, オペレーティングシステムの研究に従事. 「モンテカルロ法とシミュレーション」(培風館), 「現代オペレーティングシステムの基礎」(オーム社, 共著), 「数値処理プログラミング」(岩波書店) などの著書がある. 昭和63年度および平成3年度情報処理学会論文賞受賞. 元本学会関西支部長. ACM, SIAM 各会員. IFIP/WG 2.5 (Numerical Software) 委員.