ハイパフォーマンス
コンピューティング 57−7
（1995. 8. 24）

# 階層的コレクションに基づくオブジェクト指向分散ライブラリについて

佐藤 直人　松岡 聡　米澤 明憲

東京大学

並列性と分散とをどの様に適切に分離するかは，高性能超並列計算の実現ための重要な問題となっている．大規模な並列プログラムの作成の費用を考慮してプログラムから分散の詳細を隠蔽したいとする要請がある一方で，高い性能を達成するために計算環境に適応させられる様にする必要もある．我々は，このために，階層的コレクションというオブジェクト指向分散クラスライブラリ構築のための仕組を提案した．階層的コレクションは，分散の詳細をクラス利用者から隠蔽する一方で，様々な計算環境に応じたクラスの導出を可能にする．本稿では，いくつかの具体的な分散法をとりあげ，実際に階層的コレクションをどの様に定めればよいかについて述べる．

# Hierarchical Collection: a Simple Scheme for the Separation of Parallelism and Distribution

SATO, Naohito　MATSUOKA, Satoshi　YONEZAWA, Akinori

The University of Tokyo

Separation of parallelism and distribution is one of major concerns of efficient massively parallel computation. The details of distribution should be hidden from users of parallel / distributed class frameworks, but should be easily modifiable by (library) programmers. We have proposed a new scheme for building object-oriented parallel distributed class frameworks based on a simple but mathematically disciplined model called *hierarchy of collections*. Based on this model, classes can be easily derived to achieve high performance massively parallel computation on a variety of physical platforms. We describe in detail how to define hierarchical collections for typical examples of distributions.

## 1　はじめに

超並列計算の対象が多くの要素から構成されている場合には，通常，それらの間の相互関係を規定する構造が定められる．例えば，配列，リスト，木等は構造の一種である．これらの構造 - コレクションと呼ぶ - を利用する一つの方法として，種々のコレクションの雛型を集めてライブラリとして構築することが考えられる．その際，並列性 (parallelism) の記述と分散 (distribution) の指定とをどの様に扱うかが問題になり，更に，ライブラリを利用する側からの要請として，簡便性と拡張性とを実現する必要がある．このことを言い直すと，次の二点が重要な要請として挙げられる．

- ▷ 並列性と分散との直交性 (modularity)
  両者を分離した形で記述できる様にする．これにより異なる分散への適応のための拡張を独立して行なうことが可能になる．

- ▷ 可搬性 (code invariance)
  ユーザの記述したコードが，様々な分散環境の下で変更なしに実行できるようにする．これによりライブラリの利便性の向上を図ることができる．

## 2　階層的コレクション

我々は，計算環境に分散されたコレクションには元の構造が多くの場合再帰的に出現していることに注目し，その性質を用いて，直交性と可搬性とを両立させる様なオブジェクト指向クラスライブラリの構築の基礎となる階層的コレクションを提案した [1]．コレクションは，分散に応じて，階層をなす複数のコレクションとして再構成され，階層内のコレクションを共通の抽象的な空間を用いて特徴付けることや，コレクション間の関連およびコレクションに対する計算をその上で表現することができる様になる．階層的コレクションは，オブジェクト指向言語における *collection* (container) クラスと同様にして実現され，元のコレクションを，要素の集まり $E^0$ へのコンストラクタ $C^\perp$ の適用による構造化である $C^\perp(E^0)$ として表わす時，コレクションの分散は，$C^\perp$ の部分 (*part-of*) でありかつ同時にサブクラス (*is-a*) であるような $C^1, C^2, \ldots, C^{\exists k}$ により，階層構造をなす $C^k(\cdots(C^2(C^1(E^0)))\cdots)$ として実現され，実際の分散に関する処理は，それらの中に $C^\perp$ からは分離して - ユーザからは隠蔽されて - 記述されることになる．最も単純な場合，分散は，PE に対応する $C_L$ とそれらを合わせた $C_G$ によって $C_G(C_L(E^0))$ として実現される．例えば，分散行列については，PE には元の行列の部分である小行列が対応する一方で，全体がそれらの小行列の行列として表わされる様に，分散は，元の性質を保つ様な適当なコレクションの連鎖として実現される場合が多い．分散をこの様に捉える場合，次のことが本質的な問題になる．

- ▷ $C^\perp$ および $C^1, C^2, \ldots, C^k$ の属性として何を与えれば，すなわち何を可変にすれば，それらの間の相似性を十分に表現できるか．

- ▷ $C^k(\cdots(C^2(C^1(E^0)))\cdots)$ に対する「計算」が元の $C^\perp(E^0)$ に対するそれに一致することをどの様に保証するか．

後者については，既に [1] で一致のための条件を与えた．実際，コレクションに対して共通の定義領域と順序関係とを与えることで，異なる階層化がなされたコレクションに対する計算が等価であることを容易に保証できるようになる．以下では，前者について，どの様に抽象クラス $C^\perp$ を定義し分散法に応じてコレクションの属性を設定するかを，具体的に示していくことにする．

## 3　適用例

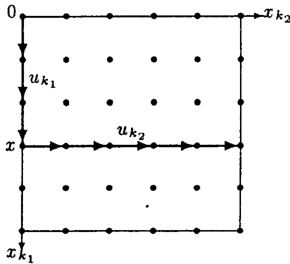
本節では，代表的な構造の階層的コレクションとして扱いについて述べ，いくつかの分散法の実現を示す．尚，これらのコレクションは適当なオブジェクト指向言語を用いて容易にクラスとして実現できるが，ここでは C++ に似た記法による定義を示すことにする．

### 3.1 配列

ベクトル空間 $\mathbb{Z}^n$ の領域 $D$ で定義された ($dim$ 次元) 配列 $A$ は，原点 ($\boldsymbol{x}_0 \in D$) および基底 ($\boldsymbol{E} = (e_1, \ldots, e_{dim}), e_i \in \mathbb{Z}^n$) によって定められ，その要素は，座標を指定して

$$A[\boldsymbol{x}] = A[i_1, \ldots, i_{dim}] \ where \ \boldsymbol{x} = \textstyle\sum_k i_k e_k + \boldsymbol{x}_0$$

として参照される．そして，配列に対するループを実行する際には，横断単位 ($\boldsymbol{E}_{trv} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n), \boldsymbol{u}_i \in \mathbb{Z}^n$) が用いられ，次元 ($k_1, \ldots, k_m (m \leq n)$)，開始点 ($\boldsymbol{x}_{init}$)，範囲 ($p$)，関数 ($f$) を指定すると，$A$ に対するループ $A.do$ は，領域 $\{\boldsymbol{x} \in D \mid p(\boldsymbol{x})\}$ の元に対して，$\boldsymbol{E}_{trv}$ を用いて順に $f$ を適用していく．例えば




$A.do(k_1, k_2, 0, \lambda x.1, f)$

次の図に示されたループは，通常の記法における

$$\mathrm{DO}(i_{k_1} = 0:5, i_{k_2} = 0:5) \ \{ \ f(i_{k_1}, i_{k_2}); \ \}$$

に等価である．次図に抽象配列クラスの定義を示す．

```
generic class Array⊥(Elem): public Collection {
private:
  int dim,size[dim];
    // dim≤n(=dimension of base space)
  Vec E[dim],x0; // basis,origin; Vec =def Z^n
  Vec trv(int,Vec),Etrv[n]; // for traversal
  Elem contents[size[0],...,size[dim-1]];
public:
// accessor
  Elem& operator[](Vec x) {
  // look up element at x∈ D ⊂ Z^n,
  // after translating Σk ik E[k] + x0=x
  // t(i1,...,idim) = E^-1(x - x0); if dim=n
    int i[dim]; translate(E,x0,x,i);
    return(contents[i[0],...,i[dim-1]]); }
// iterator
  virtual void doall(int k1,...,int km,
    Vec xinit,optional int p(Vec),void f(Vec));
  virtual void do(int k1,...,int km,
    Vec xinit,optional int p(Vec),void f(Vec));
};
```

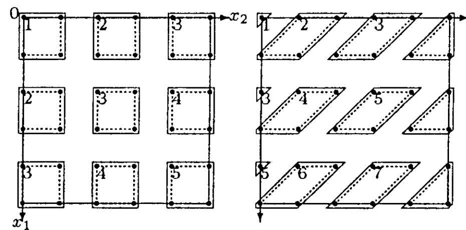
以下では，階層的コレクションとして定義した配列が十分な機能を備えていることを示すために，いくつかの例を考える．

#### Skew 変換

イタレーション間の依存性によっては，ループの形状を変更することで並列性を抽出できる場合がある．横断の実際を ($\boldsymbol{E}_{trv}$ により) 配列の属性とする我々の方法は，ユーザのコードの変更なしに skew 変換を容易に実現できる．例えば，二次元配列 $A$ を分散させて次の様な緩和計算を並列に行なう場合，

$$A[i,j] = \tfrac{1}{5} \cdot (A[i,j] + A[i+1,j] + A[i,j+1] + A[i-1,j] + A[i,j-1])$$

分散の方法は一意ではないが，階層化による分散指




$A.do(1, 0,$
$\lambda x.\{ \ A.doall(2, x, \lambda x'.\{ \ A[x'] = \cdots; \ \}) \ \})$

定の分離により，元のコードの変更や追加を行なわずにこれらの分散環境で計算を実行することができる．このことは，通常の方法を用いた場合に必要になる次の様な修正を考えると，利点の一つと言える．

```
DO(i)
  DOALL(j)
    DO(i') DO(j') { A[i'',j''] = ···; }
```

横断単位を変更することで様々な種類のループに対応する方法は，例えば [2] で提案されているイタレーション・テンプレートとの類似もあるが，階層内のコレクション毎に横断単位を定めることでより複雑な分散への対応が可能になる.

### BLOCK,CYCLIC 分散

$N$ 個の要素からなる一次元配列 $A[N]$ を，$N_{\mathrm{PE}}$ 個の PE に BLOCK / CYCLIC 分散させる場合，$\mathrm{PE}_p(0 \le p < N_{\mathrm{PE}})$ には $A$ の次の要素が対応する.

$$\begin{cases} A[p{\cdot}N/N_{\mathrm{PE}}], .., A[(p+1){\cdot}N/N_{\mathrm{PE}}-1] \\ \hfill \text{(BLOCK)} \\ A[p], A[p+N_{\mathrm{PE}}], .., A[p+(N/N_{\mathrm{PE}}-1){\cdot}N_{\mathrm{PE}}] \\ \hfill \text{(CYCLIC)} \end{cases}$$

この時，配列 $A$ を，階層的配列 $A_G(A_L(E))$ として次の表の通り定めればよい.

| | $A_G$ | $p$-th $A_L$ |
|---|---|---|
| $x_0$ | 0 | $p \cdot N/N_{\mathrm{PE}}$ |
| $E, E_{trv}$ | $N/N_{\mathrm{PE}}$ | 1 |

BLOCK 分散

| | $A_G$ | $p$-th $A_L$ |
|---|---|---|
| $x_0$ | 0 | $p$ |
| $E, E_{trv}$ | 1 | $N_{\mathrm{PE}}$ |

CYCLIC 分散

### Twisted Data Layout (TDL)

TDL は，配列の全次元についての繰り返し ($n$ 次元配列に対する $n$ 重の繰り返し) が複数あって，各々の中に唯一つの並列繰り返しが出現する状況を対象とした分散法の一つである [3].

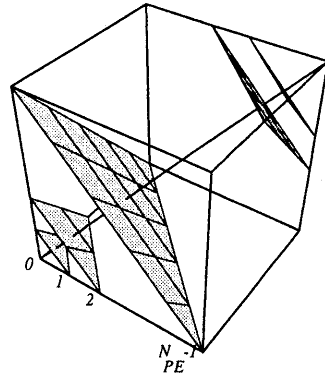
TDL では，$N_{\mathrm{PE}}$ 個の PE ($\{\mathrm{PE}_p\}_{0 \le p < N_{\mathrm{PE}}}$) への，$n$ 次元の配列 $A[N_{\mathrm{PE}}, \ldots, N_{\mathrm{PE}}]$(要素数 ${N_{\mathrm{PE}}}^n$) の分割として，$N_{\mathrm{PE}}^n$ 個の格子点からなる領域

$$I = \{(x_1, \ldots, x_n) \in \mathbb{N}^n \mid 0 \le x_{1 \le i \le n} < N_{\mathrm{PE}}\}$$

を，$(1, 1, \ldots, 1) \in \mathbb{N}^n$ を法線として周期をなす $N_{\mathrm{PE}}$ 個の $n$-1 次元平面:

$$\pi_p : \sum_{1 \le m \le n} x_m = p \ (\mathrm{modulo}\ N_{\mathrm{PE}}) \ (0 \le p < N_{\mathrm{PE}})$$

で切断して，$\{A[\boldsymbol{x}] \mid \boldsymbol{x} \in I \cap \pi_p\}$ を $\mathrm{PE}_p$ に割当てる，という分散が行なわれる.




そして計算は次の図の様に変換される.

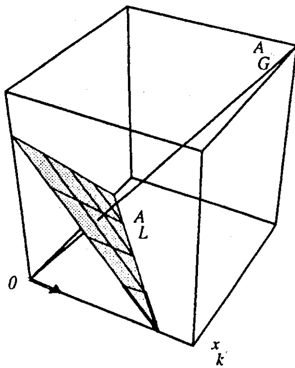
```
double A[NPE,...,NPE];
DO(i1=0,NPE-1) {
 ...
  DO(ik-1=0,NPE-1) {
   DOALL(ik=0,NPE-1) {
     DO(ik+1=0,NPE-1) {
      ...
       DO(in=0,NPE-1) {
        ...A[i1,...,in]...; }..}}}..}

                    ⇓

   // for the p-th PE (0≤p< NPE)
     DO(ik+1=0,NPE-1) {
      ...
       DO(in=0,NPE-1) {
        ik=(p + Σm≠k(NPE-im)) mod NPE;  (a)
        ...A[i1,...,in]...; }..}
```

TDL の通常のプログラム変換

この様な配列の分散及び変換は，階層的コレクションを用いて容易にい実現することができる．すなわ

| | $A_G$ | $p$-th $A_L$ |
|---|---|---|
| $\boldsymbol{x}_0$ | $\mathbf{o}$ | $p\boldsymbol{e}_1$ |
| $\boldsymbol{E}$ | $\{\boldsymbol{e}_1\}$ | $\{\boldsymbol{e}_m - \boldsymbol{e}_1\}_{2\le m\le n}$ |
| $\boldsymbol{E}_{trv}$ | $\{\boldsymbol{e}_m\}_{1\le m\le n}$ | $\{\infty,\ldots\}$ |

ち $n$ 次元配列 $A$ を，1 次元配列 $A_G$ と $n$-1 次元配列 $A_L$ により $A_G(A_L(\mathbb{R}))$ として階層化し，各々について次の様に定めると TDL の分散が得られる．ここで，$0 \le p < N_{\rm PE}, (\boldsymbol{e}_j)_j = (\delta_{ij})_{ij}$ とする．そして




$A.doall(k, 0, \ldots)$

第 $k$ 次元についての並列ループを実行する時に，

$$A_G.\boldsymbol{E}_{trv}[m] = \begin{cases} \boldsymbol{e}_k & (m = k) \\ \boldsymbol{e}_m - \boldsymbol{e}_k & (m \ne k) \end{cases} \quad \cdots \text{(b)}$$

と変更すればよい．このことの妥当性は，恒等式

$$\textstyle\sum_{m\ne k} i_m \boldsymbol{e}_m + (p - \sum_{m\ne k} i_m)\boldsymbol{e}_k$$
$$\textstyle = p\boldsymbol{e}_k + \sum_{m\ne k} i_m(\boldsymbol{e}_m - \boldsymbol{e}_k)$$

の両辺が，各々，上の (a) および (b) に対応することを見れば分かる．具体的なクラス定義は次の図に示す通りになる．

### 3.2 木

配列と同様にして，木についても，添字 (ラベル) の空間 $X$ の領域 $D$ の上で参照関数と横断関数を定

```
generic class TDL_Array_G(Elem): public Array^A_G {
public:
// accessor
 double& operator[](Vec x) {
  return((double)(*super)[x][x]); }
// iterator
 void doall(int k,Vec x_init,int p(Vec),void f(Vec)) {
  E_trv=(e_0-e_k,...,e_k-1-e_k,e_k,e_k+1-e_k,...,e_n-1-e_k);
  super→doall(k,x_init,p,f);
  E_trv=(e_m)_0≤m<n; }
};
```

めて，木を階層的コレクションとして構成できる．空間 $X$ は順序集合であり，ラベル間に $X$ の順序が入れられたものが木である．木に対する横断は関数

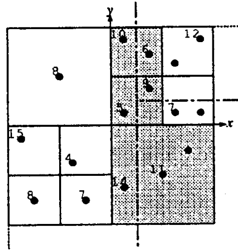
$$\mathsf{trv} : X \supset D \ni \boldsymbol{x} \mapsto (x'_1, x'_2, \ldots) \in D^*$$

により規定され，開始点の集合 $(\boldsymbol{xs}_{init})$，範囲 $(p)$，関数 $(f)$ を指定すると，木 $T$ に対するループ $T.do$ は領域 $\{\boldsymbol{x} \in D \mid p(\boldsymbol{x})\}$ の元に対して，trv を用いて順に $f$ を適用していく．木の場合，並列性は同位の節点 (siblings) に対する処理の分岐にあるが，下向き (top-down) と上向き (bottom-up) の横断に合わせて横断関数を用意する．抽象木クラスの定義を示す．

```
generic class Tree^⊥(Elem): public Collection {
private:
  List(X) trv(X); // x ↦ (x'_1,x'_2,...)
  List(Pair(X,Elem)) contents; // ((x_1,y_1),...)
public:
// accessor
  Elem& operator[](X x);
// traverser
  virtual List(X) do( // downward traversal
    List(X) xs_init,int p(X),void f(X,X,X));
  virtual List(X) do*( // upward traversal
    List(X) xs_init,int p(X),void f(X,X,X));
  virtual List(X) doall(
    List(X) xs_init,int p(X),void f(X,X,X));
  virtual List(X) doall*(
    List(X) xs_init,int p(X),void f(X,X,X));
};
```

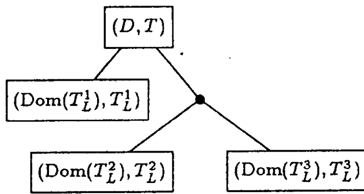
#### Recursive Bisection

空間 $\mathbb{R}^n$ 内の領域 $D$ に分布する要素を，PE あたりの要素数が均等になる様に分割するための代表的な方法として再帰的二分法がある．領域の分割に対応



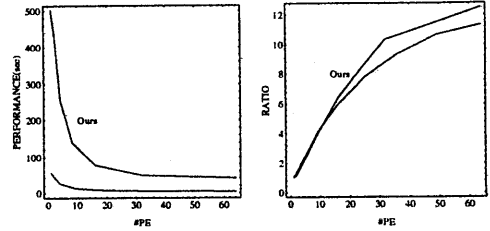
$T.doall(\{D\}, p, f)$

して，部分領域内の要素の集合に対して部分木 $T_L$ をつくり，全ての $T_L$ をあわせた全体木として $T_G$ をつくることができる．領域 $D$ 上につくられた元の木を $T$ とすると，例えば，三つの部分木 $T_L^k (k = 1, 2, 3)$ から成る $T_G$ は，次の図の様に階層的木として表わされ，$T$ に対して定められた計算をそのまま実行することができる．




## 4 考察

配列についていくつかの実験を行ない性能を評価した結果，階層的コレクションの実現には性能面で問題があることが分かった．性能向上の阻害要因としては，現在の C++ がこの種のコードの最適化に不向きであること – 具体的には，1) 動的に大きさが決定された配列の要素へのアクセスの最適化を行なわないことや，2) 二次元ベクトルの様な小さなオブジェクトをレジスタ上に生成しないことなど – があると判明しており，オーバーヘッドの解消のための最適化技法の研究が今後の課題である．




TDL による ELMHES 関数の実行

## 5 おわりに

並列オブジェクトの集まりを扱うための階層的コレクションを定め，実際の分散環境に応じた階層の構成法について述べた．配列の例についての具体的な検討から，BLOCK/CYCLIC 等の単純な分散だけでなく，Twisted Data Layout の様な複雑な分散も容易に扱えることが示され，階層的コレクションが高い汎用性を備えていることが判明した．

今後の研究として，まず，粗行列やマルチグリッド等を我々の仕組を用いて取り扱える様にしたい．これらは，不規則な性質を持ち扱いが困難であるが，数値計算において重要な役割を占めており，簡潔な取り扱いが期待される．また，コレクションをプロセッサの位相に対応させる指示法を定めたい．HPF の分散ディレクティブ (ALIGN) 等では機能が不十分なので，今後新たに検討していく必要がある．

## 参考文献

[1] 佐藤直人, 松岡聡, 米澤明憲. 超並列計算用オブジェクト指向分散クラスライブラリの構築. In *Joint Symposium on Parallel Processing*, May 1995.

[2] 末広謙二, 草野和寛, 蒲地恒彦, 妹尾義樹, 田村正典, 左近彰一, 渡辺幸光, 白戸幸正. HPF 処理系における計算処理マッピング. In *Joint Symposium on Parallel Processing*, pp. 369–376, May 1995.

[3] 進藤達也, 岩下英哉, 土肥実久, 萩原純一, 金城ショーン. Twisted Data Layout. In *Joint Symposium on Parallel Processing*, pp. 161–168, May 1994.