ハイパフォーマンス
コンピューティング 62-23
(1996. 8. 29)

C++ テンプレート・ライブラリを用いた行列解法の並列化

西川宜孝[†] 佐藤三久[††] 松田元彦[††]
石川 裕[††] 関口智嗣[†††]

HPC++ における配列データ構造を参考に、C++ テンプレートを用いた並列配列クラスライブラリの設計を行い、MPC++ ランタイム・ライブラリと PVM を用いた実装を行った。この並列配列クラスライブラリを用いて、行列およびベクトルを表現し、行列解法(直接法、反復法)の並列化を行い、ワークステーション・クラスタ上で実験した結果、処理効率の向上にはいくつかの最適化を行う必要があるが、並列配列クラスライブラリによるプログラミングの容易性が確認できた。

Parallelization of Matrix Computation using C++ Template Library

NOBUTAKA NISHIKAWA ,[†] MITSUHISA SATO ,[††]
MOTOHIKO MATSUDA ,[††] YUTAKA ISHIKAWA [††]
and SATOSHI SEKIGUCHI[†††]

We designed a parallel array class using C++ template refered HPC++ array class and implemented the parallel array class with the MPC++ runtime library and PVM on a workstation cluster. We parallelized matrix computation using the parallel array class and executed matrix computation on the workstation cluster. Though some optimizations are needed to improve execution speed, programing experiments show that the parallel array class is a useful class library.

1. はじめに

C++ テンプレートは、C++ にパラメータ型を導入したもので、型整合性を保持したまま柔軟なプログラミングを実現する機能である。コンパイル時に、インスタンス化された型ごとにコードを生成するため、動的ディスパッチのコストがかからず、実行においてはオーバーヘッドがない。C++ テンプレート機能を用いて、効率的なデータ並列処理ライブラリの実装の研究がされている[1]。Standard Template Library (STL)[2] は、標準的なデータ構造とその操作を C++ テンプレート機能を用いて実現しているライブラリである。

High Performance C++ (HPC++)[3] は、C++ の並列拡張を行っている並列プログラミング言語であり、Standard Template Library (STL) の並列拡張を行っている。

著者らは、HPC++ における配列データ構造を参考に、C++ テンプレートを用いた並列配列データ・クラスの設計および実装を行った。この並列配列データ・クラスを用いて、行列およびベクトルを表現し、行列解法(直接法、反復法)の並列化を行った。さらに、ワークステーション・クラスタ上において、MPC++[4] ランタイムライブラリおよび PVM を用いて、行列計算を行った。

本稿では、C++ テンプレートを用いた並列配列データ・クラスの設計および実装、ワークステーション・クラスタ上における実行結果について報告する。

2. 並列配列クラス

2.1 STL と HPC++

STL は、標準的なデータ構造とその操作を C++ 言語のテンプレート機能を用いて実現しているライブラリである。STL の主なクラスは、Iterator、Container、Generic function である。Iterator は、C のポインタの一般化したものである。コンテナクラスは、vector、queue、deque、set、multiset 等の特別なデータ構造を提供している。STL generic function は、共通のベクトルとリスト処理を一般化した関数テンプレートである。

HPC++ では、STL および A++/P++[5] に基づく配列クラスライブラリの並列拡張を行っている。また、

† 富士総合研究所 計算科学・小池クラスター
Fuji Research Institute Corporation
†† 新情報処理開発機構
Real World Computing Partnership
††† 電子技術総合研究所
Electrotechnical Laboratory

HPC++ では、共有メモリマシンでは、基本的な並列処理は、ディレクティブによるループの並列化によって実現することもできる。

本稿では、HPC++ における配列クラスライブラリの並列拡張を参考に、STL を拡張した配列クラスの実装を行った。

2.2 並列配列クラスの概要

行列計算用の配列クラスライブラリとして、STL のコンテナクラスの 1 つである vector クラスを基に 2 段階の拡張を行った。まず、第 1 段階は、行列計算に必要な 2 次元配列クラスおよびそれに関連するクラスへの拡張である。第 2 段階は、行列計算に必要なクラスの並列化拡張である。

逐次の行列計算に必要な拡張クラスは、2 次元配列クラス array2、iterator クラスである element である。vector クラスおよび allocator クラスは、STL のクラスをそのまま使用した。

次に、行列計算に必要なクラスの並列拡張クラスは、並列 1 次元配列クラス pvector、並列 2 次元配列クラス parray2、並列データ分散クラス pallocator、および GlobalPointer クラスである。これらの並列配列クラスは、SPMD 処理を行うライブラリとして設計している。

2.2.1 array2 クラス

array2 クラスは、STL の vector クラスを 2 次元に拡張した 2 次元配列クラスであり、行列を表現するのに用いる。先頭の要素をさす`start`、終端の要素をさす`finish`、行方向、列方向の要素の個数を示す`size[2]`をメンバとしてもつ。配列の各要素をたどるには、iterator クラスを用いる。array2 クラスは、実際には、データは 1 次元配列で持ち、基本的には vector クラスと同様である。

2.2.2 element クラス

element クラスは、vector クラスおよび array2 クラスをたどるために導入された iterator クラスである。element クラスは、以下のような記述で用いる。

```
array2<double> a(row_size,column_size,0.0);
array2<double>::iterator e = a.begin();
while( e++ ) {
        /* e に関する処理  */
}
```

2.2.3 pvector クラス

pvector クラスは、vector クラスを並列拡張したクラスである。pvector クラスでは、複数の部分配列パーティションにデータを分割し、それぞれを各 PE に割り当てる。配列の分割数と PE 数は、必ずしも一致する必要はない。各パーティションは、vector クラスと同じように、先端の要素をさす`start`、終端の要素をさす`finish`によって表現される。vector クラスと異なるのは、要素を GlobalPointer でもっていることである。pvector クラスは、各パーティションの`start`、`finish` の配列のポインタ、パーティション数`num_of_partitions`をメンバにもつ。図 1に、pvector クラスのデータ構造を示す。

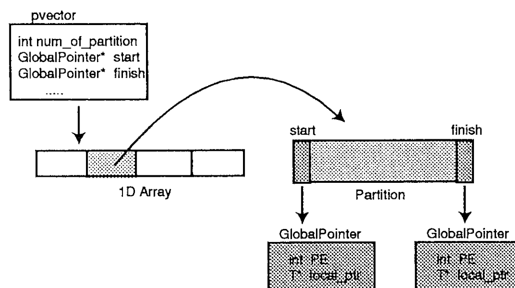

図 1 pvector クラスのデータ構造

pvector クラスでは、GlobalPointer を介することにより、各 PE に分散している配列データを参照できる。実際のプログラミングにおいて、GlobalPointer を介してのデータ参照は、ローカルメモリ上のデータに対しては、GlobalPointer を介することによるオーバーヘッドが生じ、リモートメモリ上のデータに対しては、データサイズの小さい通信が頻繁に行なわれることになり効率的ではない。配列データを参照する際、parray クラスの各パーティション単位にローカルメモリ上の配列にコピーして、そのローカルな配列を参照した方が効率的である場合がありうる。そこで、pvector クラスには、各パーティションごとにアクセスできる iterator を用意している。

2.2.4 parray2 クラス

parray2 クラスは、array2 クラスを並列拡張したものであり、pvector クラスを 2 次元配列用に拡張したものでもある。

parray クラスも、pvector クラスと同様に、複数の部分配列パーティションにデータを分割し、それぞれを各 PE に割り当てる。配列の分割数と PE 数は、必ずしも一致する必要はない。各パーティションは、vector クラスと同じように、先端の要素をさす`start`、終端の要素をさす`finish` によって表現される。parray2 クラスは、各パーティションの`start`、`finish` の配列のポインタ、パーティション数`num_of_partitions`、行方向および列方向の要素の個数`size[2]` をメンバにもつ。図 2に、parray2 クラスのデータ構造を示す。

parray2 クラスは、pvector クラスと同様に、各パーティションごとにアクセスできる iterator を用意している。

2.2.5 pelement クラス

pelement クラスは、element クラスを並列化用に拡張したクラスであり、pvector クラスおよび parray2 クラスをたどるために導入された iterator クラスである。HPC++ の element クラスを基にしている。

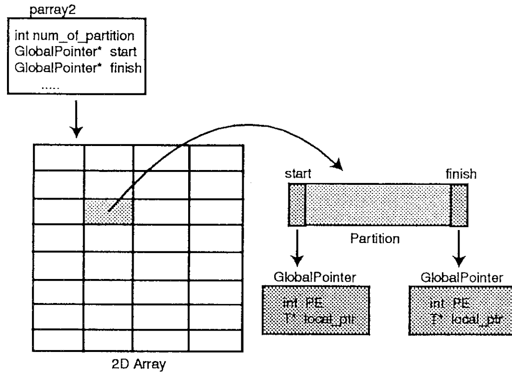

図2 parray2 クラスのデータ構造

MPC++ のランタイムライブラリを用いた実装においては、他の PE が管理する配列の要素に対しても、アクセスできるようになっている。pelement クラスは、以下のような記述で用いる。

```
parray2<double> a(row_size,column_size,
                  number_of_partitions0.0);
parray2<double>::iterator e = a.begin();
while( e++ ) {
        /* e に関する処理  */
}
```

また、pvector クラスおよび parray クラスの各パーティションにアクセスする機能ももつ。以下の記述の場合、配列a の、パーティションpid の先頭のポインタが iterator に設定される。

```
parray2<double> a(row_size,column_size,
                  number_of_partitions0.0);
parray2<double>::iterator e = a.begin(pid);
while( e++ ) {
        /* e に関する処理  */
}
```

2.2.6 pallocator クラス

allocator クラスを並列化用に拡張したクラスである。pallocator クラスのコンストラクタでは、各 PE において、allocator クラスの関数を用いて、自分が担当する配列の領域を確保する。そして、他の PE が管理する配列に対しては、通信によってその配列のポインタを得る。領域のデータ型は GlobalPointer である。

2.2.7 GlobalPointer クラス

各 PE のローカルアドレスを共有するために導入するクラスである。メンバとして、プロセッサ番号pe、ローカルアドレスptr をもつ。並列拡張クラス pvector および parray2 は、配列の要素は GlobalPointer でもっている。

2.2.8 行列表現クラス

本研究で並列化を行なった行列解法ルーチンは、行列の性質およびデータ分散方法に依存した部分があるため、parray2 クラスによって行列を表現するのは不十分である。そこで、parray2 クラスを拡張して行列を表現するクラスを作成した。拡張した行列表現クラスは、SBS データ分割された密行列の表現する sbsmatrix クラス、対称帯行列を表現する pbandarray2 クラスである。これらのクラスは、parray2 クラスのメンバの他に、行列の性質およびデータ分散方法に関する情報をメンバとしてもつ。

3. 並列行列計算法

C++ テンプレートを用いた並列配列クラスを用いて、行列解法の並列化を行った。行列解法は、直接解法である LU 分解法、反復法である SCG 法、ICCG 法である。直接法では密行列を、反復法では対称疎行列を対象の行列とした。以下に、各行列解法の並列化方法について説明する。

3.1 直 接 法

連立一次方程式の直接解法として LU 分解法 (ガウスの消去法) を並列化を行った。対象とする行列は非対称密行列とし、LU 分解法として外積形式ガウス法を用いる。データ分散方法として、2D Square Block Scattered Data Decomposition (SBS 分割) を採用した。図 3に SBS 分割を示す。

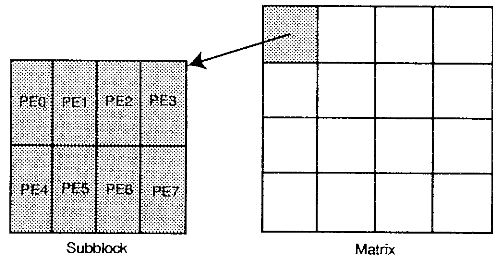

図3 2D Square Block Scattered Data Decomposition (SBS 分割)

連立一次方程式 $Ax = b$ において、行列 A を並列配列クラス parray2 を拡張した sbsmatrix クラスで表現し、右辺ベクトル b および解ベクトル x を pvector クラスで表現する。sbsmatrix クラスは、parray のメンバの他に、SBS 分割に関する情報をメンバとしてもつ。行列解法ルーチンは、行列 a、解ベクトル x、右辺ベクトル b の要素を設定した submatrix クラス、pvector クラスを引数とし、SPMD モデルで行列計算を行う。実際には、以下のように記述する。

```
sbsmatrix<double> a(n,na,nb,np.0.0);
pvector<double> b(np,pid,nn,n,0.0);
pvector<double> x(np,pid,nn,n,0.0);
/*  a, b, x の要素の設定  */
par_directSolve(a,b,x);  /*LU 分解法ルーチン */
```

行列 A や右辺ベクトル、解ベクトルのように大域的に配列データを管理したいものに対しては、並列配列クラスを用いる。LU 分解では、以下のような処理を行う。

(1)　分解列 j 列の中で最大の a_{ij} を探す。
(2)　分解行 i 行と最大の a_{ij} がある行を交換する。
(3)　対角成分 a_{jj} で分解列 a_{ij} を割る。
(4)　分解計算 $a_{ik} = a_{ik} - a_{ij} \times a_{jk}$ を行う。

分解列の最大要素を求める場合は、並列配列クラスを用いずに、reduction 関数と broadcast 関数を使用している。分解行の交換および分解計算において、ブロック単位のデータ通信を行っている。この場合、自分が担当する行列 a の要素の更新に必要な他 PE のもつ行列 a の要素を、並列配列クラスの iterator である pelement を用いて参照している。以下の記述により、行列 a のパーティションpid の先頭の GlobalPointer が得られるので、この GlobalPointer およびデータサイズを用いて、ブロック単位でデータ通信を行うことができる。

```
parray2::iterator e = a.begin( pid );
```

データの通信方法として、MPC++ ランタイム・ライブラリと PVM を用いた。MPC++ のランタイム・ライブラリを用いる場合は、データ転送には、MPC++ ランタイム・ライブラリ中の関数_mpcRemoteMemRead と関数_mpcRemoteMemWrite 等を用いている。これらの関数は、直接リモートメモリに read、write する関数である。PVM を用いた場合において、データの転送方法には、メッセージ・パッシング方式を採用している。基本的にはデータの転送には、pvm_send、pvm_recieve 等の通信関数を用いている。

3.2 反 復 法

連立一次方程式の反復解法として前処理付き共役勾配法 (PCG 法) の並列化を行った。対象とする行列は、対称疎行列とする。

不完全コレスキー分解付き共役勾配法 (ICCG 法) では、前処理によって反復回数が減少するが、行列とベクトルの積において、前進代入と後退代入を逐次的に行わなければならず、並列化向きではない。そこで、前処理として対角スケーリングを用いる対角スケーリング付き共役勾配法 (SCG 法) の並列化も試みた。SCG 法では、行列とベクトルの積において、各 PE で並列実行ができる。図 4に、反復法におけるデータ分散を示す。

連立一次方程式 $Ax = b$ において、行列 A を並列配列クラス parray2 を拡張した pbandarray2 クラスで表現し、右辺ベクトル b および解ベクトル x を pvector クラスで表現する。pbandarray2 クラスは、parray2 のメンバの他に、バンド幅等の行列の性質に関する情報をメンバとしてもつ。pbandarray2 クラスは、行列のサイズ × バンド幅の配列データをもつ。

行列解法ルーチンは、LU 分解法と同様に、行列 a、解ベクトル x、右辺ベクトル b の要素を設定した pbandarray2 クラス、pvector クラスを引数とし、SPMD モデルで行列計算を行う。実際には、以下のように記述する。

```
pbandarray2<double> a(npart,pid,nn,
                      band_width,n,pos,0.0);
```

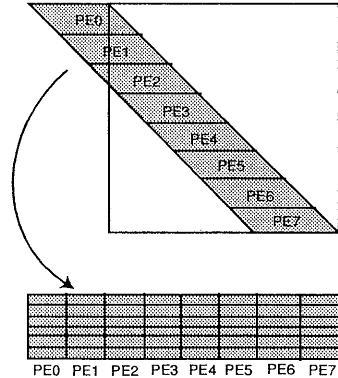

図 4　反復法におけるデータ分散

```
pvector<double> b(np,pid,nn,n,0.0);
pvector<double> x(np,pid,nn,n,0.0);
/* a,b,x の要素の設定 */
int rc = par_scg(a, b, x, itr, eps);
```

PCG 法では、隣接する PE のデータの参照が生じるが、このような場合、必要なデータをブロック単位で参照している。MPC++ ランタイム・ライブラリを用いた場合、以下のような記述によって、pvector クラス x の隣接 PE のデータを参照している。

```
_mpcRemoteMemRead(me-1,(char*)
                  (x.end(me-1).get_ptr()-m),
                  tkn, m*sizeof(T));
```

PCG 法における内積の計算は、並列配列クラスを用いずに、reduction 関数と broadcast 関数を使用している。

4. 実 験 結 果

4.1 計算機環境

SUN SparcStation 20 Model 71 (SuperSPARC 75MHz、メモリ 32MB) 32 台から構成されるワークステーション・クラスタ上において、MPC++ ランタイム・ライブラリおよび PVM(ver 3.3.9) を用いて、行列解法ルーチンの処理時間の計測および評価を行った。

ワークステーション・クラスタは、100Base-T または Myrinet のどちらかにより接続される。100Base-T で接続した場合は、9 台のワークステーションが HUB を介して接続され 1 つのクラスタを構成し、4 つのクラスタが ether switch を介して接続されて合計 36 台のクラスタを構成する (図 5)。Myrinet で接続した場合は, 4 から 5 台のワークステーションがクロスバスイッチで接続されて 1 つのクラスタを構成し, それらのクラスタがさらにクロスバスイッチで接続されて合計 36 台のクラスタを構成する (図 6).

MPC++ ランタイム・ライブラリを用いた場合は、各ワークステーションは Myricom 社製の Myrinet によ

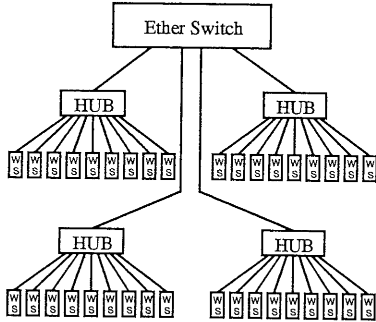

図5 ワークステーション・クラスタの構成 (100Base-T)

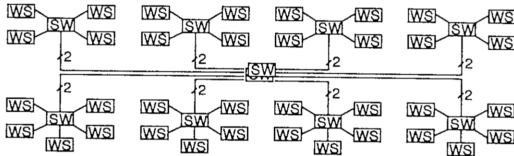

図6 ワークステーション・クラスタの構成 (Myrinet)

り接続されている。通信ライブラリ / ドライバには、ワークステーション・クラスタ上のMPC++ 実行環境のために設計された、PM を使用している[6)7)]。

PVM を用いた場合は、各ワークステーションは100Base-T で接続されており、通信プロトコルはTCP/IP である。

4.2 LU 分解法

LU 分解法ルーチンに対し、行列サイズが 800×800 の計算を行った場合の処理時間を図7に、対逐次効率を図8に示す。Linpack ベンチマークで用いられる行列を例題とした。SBS 分割におけるブロック数は、4×4 である。

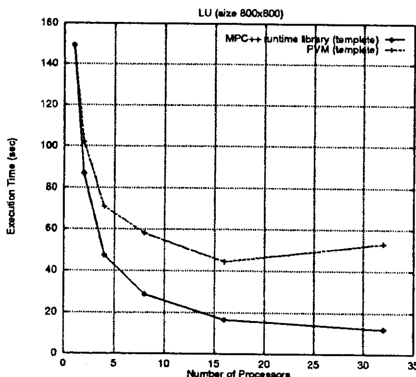

図7 LU 分解法の処理時間

MPC++ ランタイム・ライブラリとPVM を用いた場合では、通信に関する処理以外は同じ処理を行っているので、MPC++ ランタイム・ライブラリを用いた方が処理時間は少ないのは、通信性能の相違からである。

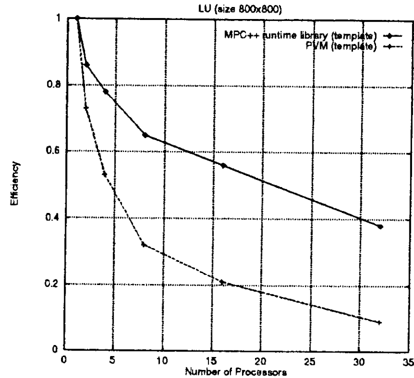

図8 LU 分解法の対逐次効率

4.3 SCG 法

SCG 法ルーチンに対し、行列サイズが 200000×200000 の計算を行った場合の処理時間を図9に、対逐次効率を図10に示す。例題に5点差分によって得られる対称疎行列[8)] を使用したため、実際の配列クラス parray2 は、200000×5 の配列を表現している。

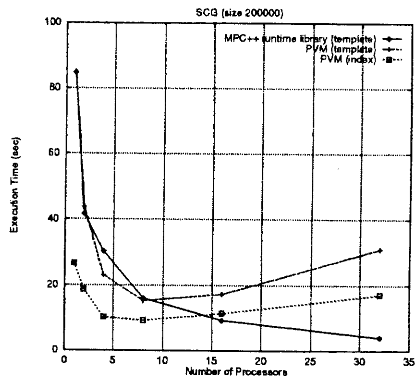

図9 SCG 法の処理時間

全体的に、LU 分解法と同様に、MPC++ ランタイム・ライブラリとPVM の通信性能の相違から、MPC++ ランタイム・ライブラリを用いた方が処理時間が少なく、良い並列化性能が得られている。MPC++ を用いた場合、2PE から 4PE にかけて速度向上が低下している。これは、Myrinet および PM の特有の性質であると考えられる。

SCG 法ルーチンに対し、C++ テンプレートによる並列配列クラスを用いた場合と、並列配列クラスを用いずに一般の配列を用いた場合との比較を行った。通信ライブラリとして PVM を使用した。並列配列クラ

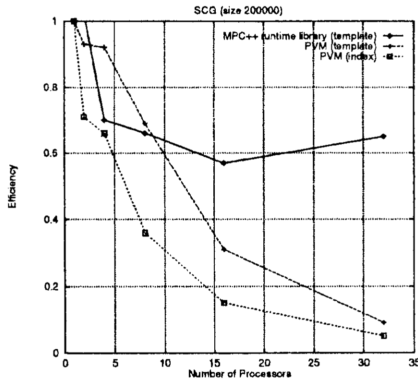

図 10　SCG 法の対逐次効率

スを用いない方が処理時間が少なく、並列配列クラスのオーバーヘッドが顕著に現れている。原因の 1 つは、GlobalPointer を介してのデータ参照を減少させ処理効率を向上させるため、並列配列クラスのデータをローカルメモリ上の配列にコピーする等の処理を行っていることと考えられる。

4.4 ICCG 法

ICCG 法ルーチンに対し、行列サイズが、200000 × 200000 の計算を行った。ICCG 法は、計算負荷の大きい行列の積の演算が、前進代入と後退代入を行うため、ほとんど並列化効率は得られない。図 11に処理時間を示す。例題は、SCG 法の場合と同じ行列を使用した。

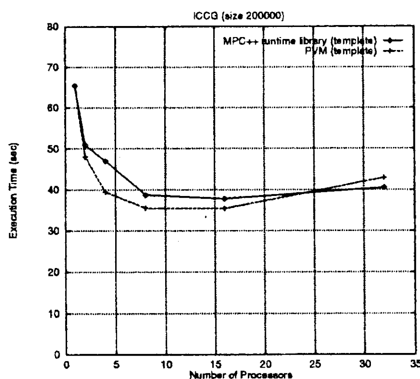

図 11　ICCG 法の処理時間

5. おわりに

HPC++ における配列データ構造を参考に、C++ テンプレートを用いた並列配列データ・クラスの設計および実装を行った。この並列配列データ・クラスを用いて、行列およびベクトルを表現し、行列解法 (直接法、反復法) の並列化を行った。そして、ワークステーションクラスタ上で、実行評価を行った。並列化には、MPC++ ランタイム・ライブラリおよび PVM を用いた。SUN SparcStation 20 クラスタ 32 台を用いて、並列化した行列解法ルーチンの評価を行った結果、全体的に MPC++ ランタイム・ライブラリを用いた方が、PVM を用いた場合より、処理時間が少なく並列化性能が良かった。

並列配列クラスを用いることにより、大域的なデータ構造を実現した。要素単位にアクセスすることが可能であるが、処理効率を向上させるためには、要素単位のアクセスは回避しなければならない。そこで、要素単位にデータ参照を行なわずに、ブロック単位にデータ転送を行い、並列配列クラスの配列データをローカルメモリ上の逐次配列クラスにコピーし、逐次配列クラスを参照する等の最適化を行う必要がある。

最適化方法に検討の余地があるが、C++ テンプレート、オペレータのオーバーロード、GlobalPointer を用いることにより、各 PE 上のローカルな配列のみを用いる場合と比較して、配列データを大域的に扱えるなど、プログラミングが容易になることは確認できた。

謝辞　本研究を遂行するにあたり御指導、御討論いただいた新情報処理開発機構超並列ソフトウェア研究室、超並列パフォーマンス研究室の諸氏、富士総合研究所 松田 勝之 主事研究員に感謝します。

参 考 文 献

1) 松田, 石川, 佐藤: C++ テンプレートを使ったデータ並列処理ライブラリの効率化手法, 並列処理シンポジウム JSPP'96 (1996).
2) A.Stepanov and M.Lee: The Standard Template Library, *http://www.cs.rpi.edu/~musser/stl.html* (1996).
3) The HPC++ working group: HPC++ Whitepapers and Draft Working Documents, *http://www.extreme.indiana.edu/hpc++/docs/hpc++wp/hpc++wp.ps* (1996).
4) 石川, 堀, 小中, 前田, 友清: 並列プログラミング言語 MPC++ の実現, 並列処理シンポジウム JSPP'94 (1994).
5) R.Parsons and D.Quinlan: A++/P++ Array Classes for Architecture Independent Finite Difference Computations, *The 2nd Annual Object-Oriented Numerics Conference* (1993).
6) 堀, 手塚, 石川, 曽田, 小中, 前田: 並列プログラム実行環境のワークステーションクラスタ上での実装, 並列処理シンポジウム JSPP'96 (1996).
7) 手塚, 堀, 石川: ワークステーションクラスタ用通信ライブラリ PM の設計と実装, 並列処理シンポジウム JSPP'96 (1996).
8) 村田, 三好, Dongarra, 長谷川: 行列計算ソフトウェア –WS、スーパーコン、並列計算機–, 丸善 (1991).