社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

クラスタ及び共有メモリ型並列マシンのための ポータブル並列プログラミングI/F MpC

緑川　博子†

† 成蹊大学 工学部, 東京都 武蔵野市吉祥寺北町３－３－１

E-mail: †midori@is.seikei.ac.jp

あらまし　メタプロセスモデルという並列プログラミングモデルをもとにしたMpC言語を開発し，計算機クラスタおよび共有メモリ型並列マシンにおける一貫性のある並列プログラミングを可能にした．メタプロセスモデルは，従来の共有メモリプログラミングモデルが持つ並列プログラムの可読性の良さと記述の簡便性，さらにメッセージパッシングモデルの持つ並列動作記述上の柔軟性の両方を提供する．実装には計算機クラスタにはユーザレベルソフトウエアDSMを，共有メモリ型並列マシンにはpthreadを用いており，幅広い計算機アーキテクチャやOSへの対応が可能である．またMpCコンパイラは下位実装にgccを使用しており，コンパイラの移植性も高い．MpCは，クラスタや共有メモリマシンにおけるOpenMPやUPCとの比較でも同等以上の性能が得られている。特別な通信ハードウエア，OSの変更などを行わず，一般ユーザレベルの権限で，容易にMpCを用いる並列処理環境を構築でき，MpCプログラムの移植性も高い．

キーワード　分散共有メモリ，性能評価，SDSM，クラスタ，SMP，並列処理言語，並列プログラミングモデル

A Portable Parallel Programming Interface MpC for Cluster Computers and Shared Memory Parallel Machines

Hiroko MIDORIKAWA†

† Faculty of Engineering, Seikei University, 3-3-1 Kichijojikita-machi, Musashino-shi,Tokyo180-8633, Japan

E-mail: †midori@is.seikei.ac.jp

Abstract　This paper proposes a new portable parallel programming interface MpC, Meta Process C, for Meta Process model. The Meta Process model gives us good readability/writability of parallel programs like shared memory programming model and high flexibility for parallel program description like message passing model. Meta Process model installation employs user level software DSM for cluster computers and pthread for shared memory parallel machines. This enables MpC programs portable to wide variety of computer architectures and UnixOSs. The MpC compiler uses gcc in its under layer, so it also has high portability. Moreover, the paper shows good MpC performance in comparison with OpenMP on cluster computers and gnuUPC on a SMP machine. Without special communication devices or OS modification, the Meta Process model and MpC can easily build a parallel programming environment with high portability.

Key words　Distributed shared memory, Evaluation, Software distributed shared memory, Cluster, SMP, Parallel programming language, Parallel programming model

1. はじめに

並列プログラムのための代表的なプログラミングモデルには，PVM, MPI に代表されるメッセージパッシングモデル（MP モデル）と OpenMP，pthread などの共有メモリモデル（SM モデル）がある．MP モデルでは，MPI が計算機の物理的メモリ構成に拘わらず標準 API として多くのシステムで使用できるようになり，並列プログラムの一般化と普及に大きく貢献した．一方，SM モデルは，従来の逐次プログラムからの継続性がよく，MP モデルに比べ，煩雑なメッセージ送受信の記述がない点で記述が簡便で，可読性が良い．このような背景から，SM モデルの標準化をめざして OpenMP が提案され，幾つかの専用並列計算機で実装されてきている．また，性能価格比の良い並列システムとして広く用いられるようになった計算機クラスタなどにおいても，ソフトウエア分散共有メモリ（SDSM) を構築し，その上で OpenMP を利用する研究が行われている．しかし，計算機クラスタでは，CPU 速度に比べ遅いネットワークでノードがつながれており，ノード内外のメモリアクセス速度差が大きいため，SM モデル用に提案された API や言語をそのまま適用しようとすると，分散メモリを意識したデータの分散割機能がなかったり，メモリ一貫性のための無駄な同期が自動挿入されたりなど，速度性能を低下させる要因を作る．

そこで我々は，SM モデルとは異なる分散共有メモリプログラミングモデルであるメタプロセスモデルを提案し [1], これを実現するための API として，MpC 言語を開発した．このモデルでは，shared というプロセス共有データ型を導入し，プロセス局所データと明確に区別する．また，無駄なメモリ一貫性同期を避け，記述の柔軟性を増すために，プログラマによる明示的な並列処理記述を前提とする．このため従来の SM モデルに比べ，計算機クラスタなどでの並列処理を，より効率よく実行させることが可能である．また実装に関しては，ユーザレベルソフトウエア DSM や，pthread を用いて，クラスタのみならず共有メモリ型並列マシンにおいても実行できるように移植性を高めた．これにより pthread やここで用いる代表的な SDSM が稼働する幅広いアーキテクチャと OS のプラットホーム上で MpC プログラムを実行することができる．

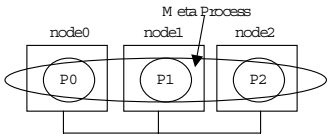

図 1　メタプロセスとプロセスの関係
Fig. 1　Meta Process and its constituent processs

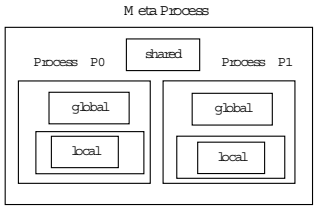

図 2　shared 変数と階層データスコープ
Fig. 2　Shared variables and the hierarchical data scope

2. メタプロセスモデル

通常，クラスタ上での並列処理は，図 1 のように，各計算ノードにあるプロセスが複数で共同して一つの処理を行う．メタプロセスモデルでは，ある処理を共同で行う複数のプロセス全体をメタプロセスと名付けている．メタプロセス中のプロセスは，通常のプロセスと同じで，ファイルやメモリなどの資源に関連づけられた OS 上の一つの実行単位である．一方，メタプロセスはここで導入した新しい概念である．通常 OS では認識されないが，分散共有メモリ機能をサポートするなんらかのシステムソフトウエアなどによって認識される，ユーザにとっての実行単位で，一つの応用処理に対応する．メタプロセスモデルでは，ある一つのメタプロセスに属するすべてのプロセスからアクセスが可能な，shared という共有データを導入し，プロセス局所（process-local）とプロセス共有（process-shared）の 2 種のデータを，階層スコープを用いて明確に区別する．しかし，メタプロセスを構成するすべてのプロセスは，共有データに関して共通の単一アドレス空間を持つ．このため，従来の MP モデルとは異なり，煩雑なメッセージパッシングの記述は不要である．メタプロセスモデルは，従来の MP モデルの持つ並列動作記述上の柔軟性と，SM モデルの持つ並列プログラムの可読性の良さや記述の簡便性の両方を兼ね備える．

2.1　共有データのための共有アドレス空間

メタプロセスとその中のプロセスという二つの実行単位に対応させ，プロセス共有データとプロセス局所データを区別し，そのデータスコープを図 2 の

ようにした．これにより共有データへはすべてのプロセスからアクセスができ，アドレスも同一でCにおけるポインタ操作も可能になっている．

2.2 緩和型メモリー貫性

共有データに関しては緩和型メモリ一貫性モデルと前提とする．これらは各種のSDSMで用いられているものであるが，この前提はクラスタなどの分散メモリ型並列マシンなどでの実行において特に性能上の利点をもたらす．一方，UMA型，SMP型，共有メモリ型並列マシンに対しても性能上なんら害は及ぼさない．

2.3 共有データのレイアウト

共有データは，メタプロセス内のプロセスのいずれかに関連づけることができる．あるプロセスがある特定の共有データや，共有データの特定部分に多頻度でアクセスすることがわかっているとき，プロセスと共有データを関連づけることで，メタプロセスを分散メモリ型マシンで実行する際に，実装システム（SDSMなど）にその情報（ヒント）を与え，より効率的なデータの配置を行うことができるようにしている．データへのアクセスパターンに特徴がなかったり，ユーザがアクセスパターンを把握できない場合には，このような関連付け記述を省略することもできる．その場合には実装系に依存した任意のプロセスに関連づけられる．

2.4 高移植性

メタプロセスモデルは分散メモリ型だけでなく共有メモリの並列マシンなどに対しても実装可能である．代表的SDSMやpthreadのAPIと同様なAPIをMpCのAPI標準に組み込むことで，MpCプログラムは，SDSMから他のSDSMへ，あるいは共有メモリ型マシンからクラスタへと移植が可能で，SDSMやpthreadの利用が可能な様々なアーキテクチャやOS上でMpCプログラムの実行ができる．

3. MpC言語

MpC(メタプロセスC）は，ANSI Cを拡張した言語で，メタプロセスモデルをクラスタや共有メモリ型マシン上で実現するためのポータブルな並列プログラミングインターフェースである．

3.1 shared storage class specifier

sharedをCのstorage class specifierとして組み込み，shared型のデータのscopeはメタプロセス全体で，図2のようにshared，global，localデータの順に階層になる．同じ名前が使用されると最内データを示すものととして扱われる．MpCのポインタは従来のCと同じでポインタの中身のstorage typeは考慮せず，type specifier（int, float, double, charなど）にのみに注目する．すなわち，ポインタの指す中身がsharedかprivateかについては関知しない．このような区別をしても，性能上，有利に扱うような仕組みが多くのSDSMにはない上，多段間接参照のポインタに対してのそのような区別はきりがなく，さらにデータ宣言記述を複雑にするだけだからである．

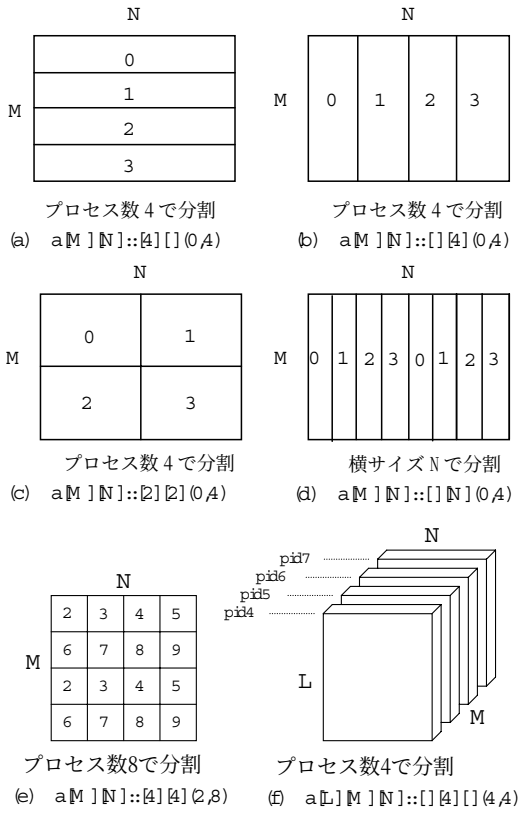

図3 共有変数のプロセスへの分散割付例
Fig. 3 Shared variable distributed mapping examples

3.2 分散データマッピング

スカラー，ベクタを問わず共有データを任意のプロセスへ割り付けることができる．特に配列データに関しては，柔軟性のある割付が可能で，バンド，タイル，ライン，キューブサイクリック割付などの，図3に示すような様々規則的な割付も容易に行えるようにしている．

3.3 MpC標準ライブラリ関数

MpCのAPIとして表1に示すような関数を用意している．実際のMpCの簡単なサンプルプログラムを図4に示す．

4. MpCコンパイラ

MpCコンパイラは，MpCからCへの変換と，MpC実行モジュール生成の２ステップがある．第一ステッ

	MpC 標準関数（一部）
初期化	mpc_init(int argc, char *argv)
終了	mpc_exit(int value)
エラー終了	mpc_err(int value, char *msg)
バリア	mpc_barrier(int value)
ロック	mpc_lock(int lockid)
アンロック	mpc_unlock(int lockid)
条件シグナル	mpc_cond_signal(int condid)
条件シグナル	mpc_cond_broadcast(int condid)
条件シグナル待ち	mpc_cond_wait(int condid, int lockid)
共有データ割付	void *mpc_alloc(size)
共有データ割付	void *mpc_alloc(char *declare)

表 1　MpC 標準ライブラリ関数（一部）
Table 1　MpC standard library functions

```
#include <stdio.h>
#include <mpc.h>
#define M 1024
#define N 2048

shared double matrix[M][N]::[NPROCS][](0,NPROCS);
shared double sum ::(0);

int main(int argc, char **argv)
{
    FILE fp;
    double mysum=0;
    int start, end, i, j;

    mpc_init(argc, argv);
    if(MYPID == 0){
        fp=fopen("initial.dat", "r");
        for(i=0; i<M ; i++)
            for(j=0; j<N ; j++) fscanf(fp,"%f", &matrix[i][j]);
        sum = 0;
    }
    mpc_barrier(0);

    start = M/NPROCS*MYPID;
    end = start+M/NPROCS;
    for(i=start; i<end; i++)
        for(j=0; j<N ; j++) mysum += process(matrix[i][j]);

    mpc_lock(0);
      sum += mysum ;
    mpc_unlock(0);

    mpc_barrier(0);
    if(MYPID == 0)  printf("Result=%lf¥n",sum );
    mpc_exit(0);
}
```

図 4　MpC プログラム例
Fig. 4　A MpC program sample

プでは，MpC の並列 API をすべてユーザの指定した下層の実装システムライブラリ関数へ変換する．ヘッダーファイルなどを展開後，shared 型のデータと通常の C のデータとの階層スコープチェックを行い，通常の C 文法に適合するようにリネームやプログラムコードの挿入を行う．第二ステップでは，ターゲットマシンの C コンパイラを用いて，指定実装システムのライブラリ関数とリンクする．現在 gcc と icc を C コンパイラとして想定している．gcc は多くのアーキテクチャで広く利用できるため，MpC コンパイラのターゲットマシンは広く，移植性が高い．実装ライブラリとしては，クラスタ用には代表的なユーザレベルソフトウエア DSM(SMS [2], TreadMarks [3], JIAJIA [4]）を用い，共有メモリ型マシン用には pthread を用いている．

Progs	Parameters	Data Size	Barrier /proc	Lock /proc
ep	M=28,MK=10	44B	2	1
tsp	19cities(19b)	100MB	4	75-122
lu	2048x2048 double, 32bloks	34MB	135	0
mm	2048x2048 double	96MB	3	0

表 2　ベンチマークプログラムとパラメタ
Table 2　Benchmark programs and parameters

gcc version 2.96 -O3	
SMS 0.4.16	
JIAJIA2.2	
TreadMarks 1.0.3.2	

CPU	Intel Pentium III -S 1.13GHZ
Memory	512MB
Network	Intel Proc 1000T 3Com SuperStack3 switch
OS	RedhatLinux 7.1.2 kernel 2.4.7.10

表 3　実行環境
Table 3　Experiment environment

5. 計算機クラスタでの実行

現在，クラスタにおける MpC プログラム実行には SMS, TreadMarks, JIAJIA などの SDSM を用い，MpC プログラムの API を各 SDSM の API に変換して実行できる．各 SDSM の API，記述方式には多少の違いがあるが，barrier, lock/unlock, alloc などの基本 API はほぼ同等である．多くの既存の SDSM プログラムはこの基本 API のみを使用して書かれているため，異なる SDSM プログラムへの変換が可能である．さらに，ここで用いた SDSM はユーザレベルソフトウエアなのでインストールも容易で，いずれも移植性が高く，多くのアーキテクチャや OS 上ですでに稼働ずみである．

MpC プログラムを SMS, TreadMarks, JIAJIA の３つの SDSM で稼働させた性能結果を図 5，6，7，8 に示す．実行プログラムは表 2，実行環境を表 3 に示す．MpC プログラムは異なる３種の SDSM で，変更なしに実行することができ，移植性は高い．lu や ep といった計算中心の応用では性能向上比は高い．tsp は実装 SDSM による違いが現れている．

MpC の API は，ここで用いた各 SDSM の基本 API とほぼ同等なので，MpC プログラムの実行と，SDSM プログラムを直接実行した場合との差は，一般には，ほとんどない．

さらに，Score5.6.1 を実装したクラスタ上において，MpC(SMS 使用) と OpenMP(Omni) との性能を

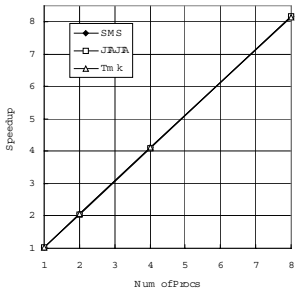

図 5　各種 SDSM における ep の性能向上比
Fig. 5　Speedup of ep on SDSMs

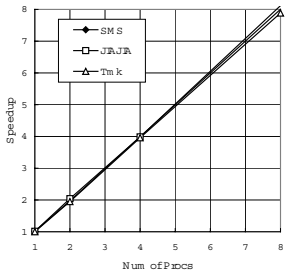

図 6　各種 SDSM における lu の性能向上比
Fig. 6　Speedup of lu on SDSMs

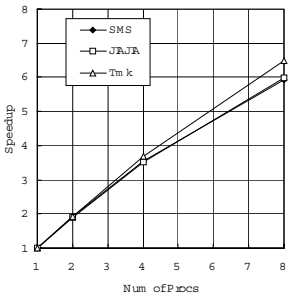

図 7　各種 SDSM における mm の性能向上比
Fig. 7　Speedup of mm on SDSMs

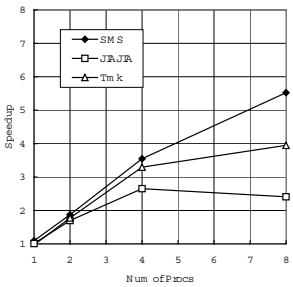

図 8　各種 SDSM における tsp の性能向上比
Fig. 8　Speedup of tsp on SDSMs

比較した．表 4 のように，6 種の OpenMP の得意とする規則的なプログラムの実行時間を計測した．MpC はギガイーサで TCP/IP を使用し，Omni はギガイーサと myrinet（PCI32C, Lanai4）で PM を用いている．ギガビットイーサの場合の MpC と比較すると，OpenMP では最短経路問題 Floyed，laplace，

Program Code		MpC	OpenMP			SCASH	
Compiler		mpcc	omcc		gcc		
Network		giga	giga	myri		giga	myri
ep	1	11.57	14.45	14.45	13.99		
S class	2	5.79	7.69	7.28			
	4	2.90	4.57	3.71			
	8	1.45	2.76	1.95			
Floyd	1	66.65	65.21	65.12	66.80		
1024x1024	2	35.01	69.80	46.57			
	4	19.42	44.94	25.30			
	8	11.38	41.62	14.71			
laplace	1	3.36	3.18	3.24	3.26	3.79	3.83
1024x1024	2	2.06	2.73	2.18		2.65	2.33
50ite	4	1.10	1.61	1.21		1.84	1.25
	8	0.75	1.40	0.63		1.39	0.66
mandel	1	1.39	1.44	1.44	1.37		
1024x1024	2	0.87	0.84	0.79			
dynamic	4	0.54	0.50	0.42			
	8	0.43	0.35	0.23			
mm	1	12.18	13.50	14.41	12.32		
blocked	2	5.16	5.75	5.22			
1024x1024	4	3.22	3.87	2.92			
	8	2.16	3.60	2.20			
galaxy	1	8.52	9.02	9.02	8.75		
1000-body	2	4.32	2.98	2.47			
10steps	4	1.88	1.61	1.25			
100time	8	0.96	1.08	0.65			

表 4　MpC と OpenMP の性能比較
Table 4　performance of MpC and OpenMP

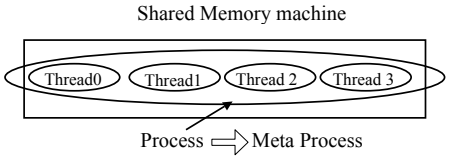

図 9　共有メモリ型マシン上でのメタプロセスモデル
Fig. 9　Meta Process Model on shared memory machines

行列乗算 mm など，共有データサイズが大きくアクセスが多いプログラムで，性能が低くなることがわかる．このようなプログラムでは myrinet と比較しても，MpC のギガイーサでの性能よりも劣る場合がある．

6. 共有メモリ型マシンでの実行

共有メモリ型並列マシンにおけるメタプロセスモデルの実装には pthread を使う．図 9 のようにメタプロセスモデルにおけるメタプロセスを通常プロセスに，メタプロセス内プロセスをスレッドに対応させる．したがって，メタプロセス内のプロセス局所データは，pthread 実装ではスレッド局所データとして表現し，shared データのみをスレッド間で共有するデータとしている．各階層データの実装を，クラスタの場合と比較したのが表 5 である．

表 6 は，2 種の共有メモリ型マシンとクラスタ

	SDSM，クラスタでの実装	pthread，共有メモリ並列マシンでの実装
一つの応用（メタプロセス）	プロセス群	プロセス
並列実行単位	1プロセス	1スレッド

メタプロセスモデルにおけるデータ型	SDSM，クラスタでの実装	pthread，共有メモリ並列マシンでの実装
shared	プロセス共有（SDSM 機能利用）	スレッド共有（global データ）
global	プロセス局所（global データ）	スレッド局所（local データ）
local	プロセス局所（local データ）	スレッド局所（local データ）

表 5　メタプロセスモデルの実装方式
Table 5　Meta Process Model implementation

で，幾つかの MpC プログラムを実行した結果である．表 6 の 3，4 列目は，SMP マシン (Pentium ,2CPU,Linux) と，共有メモリ型のサーバ (IBM rs6000,4CPU,AIX5.2) で，5 列目は比較のために，3 列目の SMP をノードとするクラスタで実行した場合の結果である．これにより，OS やアーキテクチャに依存せずに pthread をサポートする計算機システムでは，変更なしに MpC プログラムのコンパイルと実行を行うことができる．

また右端の列は，3 列目と同じ SMP マシンにおいて，gnuUPC [6] を実行した結果である．UPC [5] は，MpC とは実装や設計思想が異なるが API が似ている言語で，幾つかの組織が開発している．gnuUPC は現在 Linux，X86 に関しては SMP 版しか用意されていない．gnuUPC は，ep と mandelbrot は MpC とほぼ同性能であるが，多体問題 galaxy や mm は，MpC に比べ非常に遅い．そこで，UPC で用意されている様々な共有データマッピング（hb：水平バンド割付，vb:垂直バンド割付，1ele: 1 要素サイクリック割付 (UPC の default)，0proc：THREAD0 への集中割付）を試してみたが，いずれの方式でも gnuUPC の性能は著しく低い．これらプログラムは共有データアクセスが比較的多いが，gnuUPC の x86 系 SMP への適用が十分行われていないのかもしれない．

7. おわりに

異なる SDSM や pthread を使用して，MpC プログラムを変更なしにクラスタや共有メモリ型並列マシン上で実行できることを示した．ここで用いた SDSM は様々な OS，アーキテクチャ上で稼働が確認されており，ここでは試せなかった様々な並列マシンについても，MpC プログラムをそれぞれの SDSM プログラムや pthread プログラムにソース変換することにより，実行させることが可能である．

(sec)

programs	Num of Procs	MpC pthread smp 2CPUs Pentium3 LINUX2.4.20 -6smp RedHat9	MpC pthread rs6000 4CPUs AIX5.2	MpC SMS GigaEther pc cluster LINUX 2.4.20-6 RedHat9	gnuUPC 3.2.3.5 smp 2CPUs Pentium3 LINUX2.4.20-6smp RedHat9
ep S class	1	10.82	10.45	11.19	10.15
	2	5.54	5.23	6.09	5.09
	4		2.65	2.56	
	8			1.28	mapping 1ele
galaxy 1000body 10 steps 100time	1	9.02	8.81	8.17	64.05/62.96
	2	4.52	4.42	4.53	32.89/31.5
	4		2.24	2.53	
	8			1.63	mapping div/0proc
mandel d 1024x1024 0.3<x<0.4 0.5<y<0.6	1	1.35	1.03	1.39	1.56/1.47
	2	0.68	0.53	0.87	0.79/0.75
	4		0.38	0.54	
	8			0.43	mapping 1ele/0proc
mm512 512x512 double array blocked	1	0.77	0.61	0.80	15.2/15.2/35.26/9.21
	2	0.40	0.31	0.47	8.16/7.69/17.83/4.64
	4		0.17	0.31	mapping
	8			0.29	vb/hb/1ele/0proc
mm1024 1024x1024 double array blocked	1	6.18	4.9	6.90	132.63
	2	3.17	2.47	3.47	66.35
	4		1.49	2.04	
	8			1.57	mapping 0proc

表 6　MpC と UPC の性能比較
Table 6　Performance of MpC and UPC

また，MpC 言語と類似した API を持つ UPC や，共有メモリモデルを基本とする OpenMP とも比較しても，同等以上の性能が得られた．さらに OS の変更や特別な通信デバイスを用いることなく，ユーザレベルで容易に並列処理環境を構築できる．

メタプロセスモデルと MpC は，従来のメッセージパッシングモデルと純粋な共有メモリモデルとは違ったもう一つの選択肢になりうる可能性を示した．

文　献

[1] H.Midorikawa : "Meta Process Model: A New Distributed Shared Memeory programming Model", Proc. of The 15th IASTED International Conference on Parallel and Distributed Computing and Systems, pp.295-300, 2003

[2] 緑川，飯塚 : ユーザレベル・ソフトウエア分散共有メモリ SMS の設計と実装, 情報処理学会論文誌ハイパフォーマンスコンピューティングシステム, Vol.42, No.SIG9(HPS 3), pp.170-190, 2001.

[3] P. Keleher, S. Dwarkadas, A.L. Cox, and W. Zwaenepoel: TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems, Proc. of the Winter 94 Usenix Conf., pp.115-131, 1994.

[4] Weiwu Hu, Weisong Shi, Zhimin Tang : JIA-JIA: An SVM System Based on A New Cache Coherence Protocol, Proc. of the High Performance Computing and Networking (HPCN'99), LNCS 1593, pp.463-472, 1999.

[5] W. Carlson, J. Draper, D. Culler, K. Yelick, E. Brooks, and K. Warren: Introduction to UPC and Language Specification, CCS-TR-99-157, IDA Center for Computing Sciences, 1999.

[6] http://www.gwu.edu/ upc/software/gnu-upc.html