B-016

並列言語 MpC の高機能化
The Design and Implementation of Highly Parallel Descriptive Functions for Parallel Programming Language MpC

小山浩生† 緑川博子† 甲斐宗徳†
Hiroki Koyama Hiroko Midorikawa Munenori Kai

1. はじめに

現在，分散メモリ並列システムにおける並列プログラミングで，最も広く用いられている移植性のあるソフトウエアインタフェースといえば，MPI であろう．高性能並列コンピュータシステムのほとんどにおいて，各ベンダーにより高性能な MPI が実装され，並列プログラミングの事実上の標準 API となっている．このように，主に性能上の理由から長く続いている MPI 寡占の並列プログラミング環境において，最近，MPI 以外のプログラミングモデルや言語にも少し目が向けられるようになってきた．その一つが，GAS 言語とよばれるもので，UPC(Unified Parallel C)[4], CAF(co-Array FORTRAN)などがある．これらは，従来のデータをフラットに扱う純粋な共有メモリプログラミングモデルとは異なり，分散共有メモリモデル，区分化大域アドレス空間モデル(PGAS:Partitioned Global Address Space)に基づいている．いずれも，データのローカリティに着目し，データの分散マッピング機構を取り入れ，明示的な並列処理記述を基本としている．

筆者らは，これ以前からソフトウエア DSM の研究開発[1]をもとに，ローカリティに着目したメタプロセスモデルという階層型共有メモリプログラミングモデルを提案し，これに基づくポータブルな API として MpC 言語を開発してきた．すでに，MpC がコモディティクラスタや共有メモリ型 SMP マシンにおいて，UPC や OpenMP と比較し，性能上の優位性があることを示した．さらに，NPB ベンチマークセットを MpC と他の言語で記述したプログラムの有効行数の比較では，逐次コードに対し，MPI が平均 1.74 倍に増大するのに比べ，MpC は平均 1.07 倍にとどまっており，プログラムの可読性，生産性の上で優れていることを明らかにした[2][3].

MpC は並列プログラム記述上の柔軟性を高めるために，明示的な並列処理記述を基本とし，OpenMP のような常時固定的に全スレッド参加の処理を行う SPMD 型スタイルに限定していない．このため全スレッド参加のコレクティブ処理や並列ループなどの処理の多くを並列構文として取り込んでいる OpenMP などに比べ，若干プログラム行数が増大する傾向にあった．そこで，さらに生産性を向上させるため，このような典型的な処理に対する関数と構文を MpC に組み込んだので，報告する．

2. メタプロセスモデルと MpC 言語

メタプロセスモデルとは，従来の共有メモリモデルに，NUMA マシンなどにも対応できるような各プロセスへの共有分散データの階層構造（スコープ）を取り入れたプログラミングモデルである．メタプロセスとは，1 つの応用を協力して並列処理する複数のプロセス全体を指し，ユーザーにとっての実行単位である．MpC 言語は，このモデルが利用できるようにプロセス共有データのための shared(記憶クラス指定子)と共有データ分散マッピング指定子を新しく導入して C 言語を拡張したものである．UPC も MpC と同様に共有変数に shared(型修飾子)を使用しており，C 言語の拡張であるが，共有データの分散マッピング方式は全スレッドへの均一サイズのマッピングに固定されており，これを前提に CPU アーキテクチャ毎のコンパイラによる最適化を行うことを指向した言語である．

これに対し，MpC は柔軟性の高い様々なマッピングが可能で，コンパイル時よりも実行時に処理を行うことで柔軟性を高めている．また特定のコンパイラや実装系を前提とせず，ポータビリティが高いのが特徴である．

†成蹊大学大学院工学研究科　情報処理専攻

3. 実装

今回，プログラム記述の煩雑さの軽減を目的として，表 1 のような 5 つの新しい機能を付け加えた．(1)分散マップ付き動的共有データ割付関数，(2)broadcast，gather，scatter 関数，(3)計算 reduce 関数，(4)連続領域コピーmemcpy 関数，(5)並列 for 文　mpc_for 文，mpc_forall 文 である．

表 1　実装関数(1)-(4)と構文(5)

(1)	void * mpc_mapalloc(char *declare)
(2)	int mpc_broadcast(Type *dst, Type *src, int src_PID, int nbytes, int sync) int mpc_gather(Type *dst, Type *src, int src_PID, int nbytes, int sync) int mpc_scatter(Type *dst, Type *src, int src_PID, int nbytes, int sync)
(3)	int mpc_reduceT(Type *dst, Type *src, int op, int totalnum, int bolocknum)
(4)	int mpc_memcpy(Type *dst, Type *src, int nbytes, int sync)
(5)	mpc_for(int s, int e, int n, int i) mpc_forall(**初期化式; 条件式; 繰り返し毎の変更**; affinity)

3.1 分散マップ付き動的共有データ割付関数

MpC 言語では，共有データの分散マッピングを以下のような静的な宣言で記述できる．

shared int a[M][N]::[4][ ](0,4); （図 1(a)の例）

特に配列データに関しては，柔軟性のある割り付けが可能で図 1 に示すようにバンド，タイル，ライン，キューブ，などの様々な割り付けが容易に行える．しかし動的なメモリ割付は，以下のような任意の単一プロセスへの集中割り付けしか実装しておらず，プログラム実行時の柔軟性のある割り付けができなかった．

p = (int *)mpc_alloc(num* sizeof(int));

このため今回，共有データを任意のプロセスへ動的に割り付けることができる以下のような分散機能付きの動的共有データ割付関数を実装した．

p = (int (*)[N])mpc_mapalloc(“int p[M][N]::[4][ ](0,4)”);

引数には，ユーザーの複雑なパラメタ計算を不要にするため，静的宣言と同様の記述ができるようにした．MpC コンパイラはこの文字列引数から次元数，各次元のサイズ，各次元の分割数，割付開始プロセス番号，割付プロセス数を抜き出し，実装系で用いる分散マッピング関数に変換する．引数は，静的宣言と同様にパラメタの省略も可能になっている．

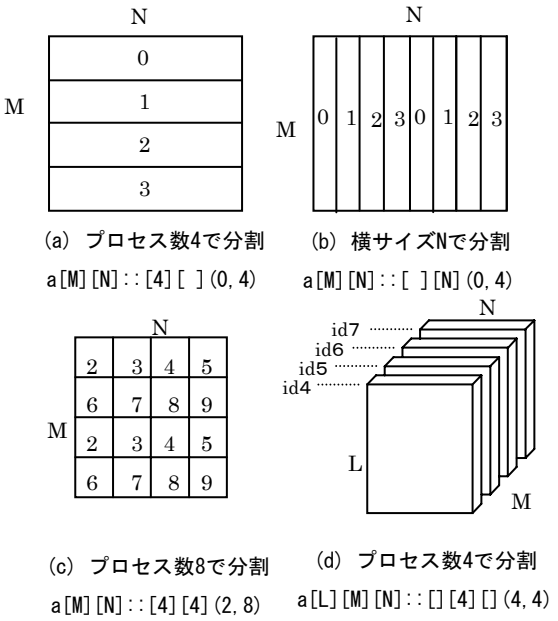

図１　共有データの分散マッピング例

3.2 collective 関数

表 1 (2), (3)の 4 つの関数では MpC ソースコードライブラリからユーザープログラムで使用されている関数のみを抽出してユーザープログラムに加える．この目的で従来 1 パスであった MpC コンパイラを図 2 のように 2 パスに変更し，1 パス目で当該関数使用の有無と種類(reduce 関数の場合，データ型と演算の種類)を同定し，該当 MpC コードをユーザープログラムに挿入する．2 パス目で全コードを C プログラムに変換する．実装方式としてはバイナリライブラリとして用意する方法もあるが下位の実装システム毎(SMS, TreadMarks, JIAJIA)にライブラリを用意しなくてもいいように，現在は MpC ソースコードをユーザープログラムソースに挿入する実装としている．

3.2.1 broadcast, scatter, gather 関数

表 1 (2)の関数は各プロセスにデータを分配，分散，集約を行う関数である．

int mpc_broadcast(Type *dst, Type *src, int src_PID, int nbytes, int sync)

UPC にもこのような機能はあるが，共有データ間(shared－shared)のみに限定されている．MpC では，図 3 のように shared－local，local－shared，shared－shared 間の共有データとプロセス局所データ間のやり取りがすべてできるようにしている．第 3 引数の src_PID は gather 関数では回収先，broadcast, scatter 関数では送信元のプロセス ID を表す．sync は全プロセスが関数の入口と出口でそれぞれ同期を取るかどうかを指定するフラグである．

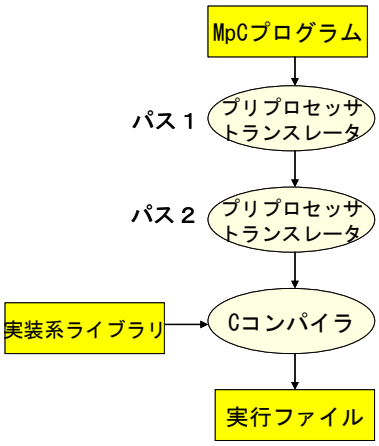

図２　MpC コンパイラ

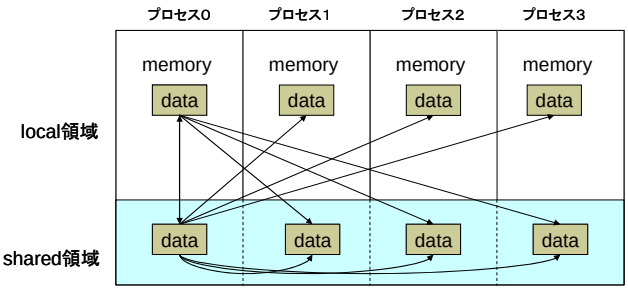

図３　broadcast の例

3.2.2 計算 reduce 関数

表 1 (3)の reduce 関数は各プロセスのデータを集めて collective な計算(SUM，AVE，MAX，MIN など)を行う関数である．計算種別を指定する第 3 引数 op には enum によるニーモニックが指定可能で，SUM(合計)，AVE(平均)，MAX(最大値)，MIN(最小値)が現在実装されている．第 1 引数は reduce 計算結果を格納する場所のアドレスで，第 2 引数は計算の対象となるデータで，通常，データ配列の先頭番地である．第 4 引数は，第 2 引数で指定したデータ配列の総データ(要素)数，第 5 引数はプロセスへの割り付け単位の要素数である．図 1 (a)のようなデータ配列なら第 5 引数は，総データ数/NPROCS(全プロセス)であるが，サイクリック型でも指定でき，第 2 引数の配列へのマッピングに合わせて，第 5 引数を指定する．この関数では内部計算に使用する共有データをとる必要があり，そのデータ型には int 型と double 型が用意されている．Type 型引数にもこの 2 種類のみが可能である．実際には関数名の最後の T に，I か D のいずれかの型名指定を指定する．ただし reduce 関数では，第 1，第 2 引数の dst と src は shared データを前提とする．

double の例

int mpc_reduceD(double *dst, double *src, int op, int totalnum, int blocknum)

3.3 memcpy 関数

表 1 (4)はデータコピー関数で，データのコピーは local－local 間を含む shared と local の組み合わせ 4 パターンで使用できるようになっている．

int mpc_memcpy(Type *dst, Type *src, int nbytes, int sync)

UPC にも同様な連続領域コピー関数はあるが shared と local の組み合わせ 4 パターンによって表 2 のように異なる関数名を用いている．MpC では引数 src と dst にデータ型に拠らない指定が可能で，同一関数名で統一した．引数 sync は(2)の関数と同様に，関数出入り口での同期方式を指定する． MpC の memcpy は，UPC と異なり noncollective 関数である．

表２　データコピー関数 UPC と MpC の比較

src- dst	UPC	MpC
local-local	-	mpc_memcpy
local-shared	upc_memput	
shared-local	upc_memget	
shared-shared	upc_memcpy	

3.4 並列 for 文

並列実行に用いている全プロセス(NPROCS)で for 文のループインデックスを分担して並列処理する場合の記述を容易にするために，2 つの並列 for 文を実装した．一つは典型的な単純ループ用の高速な mpc_for 文で，他方は一般 for 文と同様な文法形式で，for 文条件に実行時に決定する条件式などを書くこともできる柔軟性のある mpc_forall 文である．これらを用いると，プログラマが各プロセスでのループインデックスの開始値と終了値を計算する必要がなくなる．MpC では，全プロセス以外にも，任意数のプロセスグループや指定プロセスなどでループを分担させることも可能で，この場合には，ユーザーが MYPID(プロセス識別番号)などを使用し，自由に記述することができる．

3.4.1 単純並列 for 文　mpc_for 文

mpc_for 文では分担するインデックス変数名と，そのインデックス変数の開始値，終了値，インデックス変数のループ毎の加算値を指定する固定的な構文である．全プロセスでループ回数を等分に分担するような単純なコードをマクロ展開で生成する．後述の汎用 for 文と異なり，実行時に一度だけインデックス範囲を計算するだけなので高速である．第 4 フィールドのループ変数名は，プロセスごとのループインデックスとして使用されるので，あらかじめ int 型でユーザーが指定していることを前提とする．図 4 に展開例を示す．

単純並列 for 文
　mpc_for(**開始値，終了値，加算値，ループ変数名**)
例　mpc_for(s, e, d, i) ｛ ..... ｝ **のマクロ展開は，**
　for(i=s+MYPID*((e)-(s))/NPROCS;
　　i<s+(MYPID+1)*((e)-(s))/NPROCS; i += d) ｛ ..... ｝

図４　単純 for 文の展開例

3.4.2 汎用並列 for 文　mpc_forall 文

mpc_forall 文は、C 言語の for 文に第 4 フィールドとして affinity を追加した文で，単純並列 for 文と異なり，イテレーション毎にどのプロセスに割り付けるかを評価するため柔軟性が高い．affinity とは，ループインデックスのどれをどのプロセスに割り付けるかを示すもので，通常はループインデックス変数やそれを用いた数式を指定する．affinity の実行時値を全プロセス数(NPROCS)で割った剰余が自分のプロセス ID(MYPID)に等しい場合にそのループインデックスを担当する．図 5 に C の for 文への変換方式を示す．

汎用並列 for 文
　mpc_forall(**式 1；式 2；式 3；式 4**)　{....}　**は，**
　for(**式 1；式 2；式 3**)
　　if((**(式 4**)%NPROCS) == MYPID)){....} **に展開**
例 1　mpc_forall (i=0; i<N; i++; i) {....} **1 ループ毎に分担**
例 2　mpc_forall (i=s; i<e; i++; i/block) {....} **block 毎に分担**
例 3　mpc_forall (i=z+y*2; i<limit; i=i+delta; i) {....}**不規則分担**

図５ 汎用並列 for 文の展開例と使用例

4. おわりに

これらの関数や構文を MpC に導入することにより，MpC の特徴である非定型，不規則な並列処理の記述が可能であるという特性を損なわずに，さらに全プロセスで並列に処理を行う規則的あるいは定型的なループ処理や各種 collective 処理の記述が容易になった．たとえば，reduce 関数を用いると，その部分に該当するコードを明示的に記述した場合に比べ，1/20 程度にコードを短縮できる．

今後，これらの構文を用いてベンチマークプログラムなどを記述すると，全体としてどの程度の効果があるのか評価する予定である．

参考文献

[1] 緑川，飯塚 : "ユーザーレベル・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)

[2]　緑川,飯塚:"メタプロセスモデルに基づくポータブルな並列プログラミングインターフェース MpC",情処論文誌：コンピューテイングシステム,Vol.46 No.SIG4(ACS9),pp.69-85,（2005)

[3] Midorikawa,H : "The Performance Analysis of Portable Parallel Programming Interface MpC for SDSM and pthread", IEEE/ACM CCGrid2005 , Fifth International Workshop on Distributed Shared Memory (DSM2005), 2005
http://perso.ens-lyon.fr/laurent.lefevre/dsm2005/slides/DSM2005_Midorikawa.pdf

[4] William Carlson, Thomas Sterling, Katherine Yelick, Tarek El-Ghazawi UPC: Distributed Shared Memory Programming, Wiley Series on Parallel and Distributed Computing, 2005