分散メモリ向けデータ並列言語OpenMPDの設計と実装

システム情報工学研究科 1 年　200720928　李 珍泌
指導教員　佐藤 三久

平成 19 年 11 月 8 日

1　はじめに

今日では，高性能並列計算のプラットフォームとして PC クラスタを始めとする分散メモリ型のシステムが広く普及している．そのプログラミングには主に MPI（Message Passing Interface）が用いられる．MPI は並列化のための多様な機能を提供する反面，データの送受信をプログラマが明示的に指示しなければならないため，プログラミングが複雑になる．共有メモリの OpenMP のような簡単なプログラミングモデルを使うために Omni/SCASH[2] などが提案されているが，性能面で問題がある．本稿では PC クラスタのような分散メモリ型並列計算機のための簡単なプログラミングモデルとして OpenMPD を提案する．OpenMPD は OpenMP のような指示文による並列化を分散メモリ上で実現したものである．データの分散や同期，ループの並列実行によるワークシェアリングなどデータ並列プログラミングで頻繁に用いられる手法を指示文の記述によって簡単に行うことができる．CAF（Co-Array Fortran）や UPC（Unified Parallel C）のような既存の並列プログラミング言語は多様な並列化手法を提供する反面，そのモデルが抽象的で使いこなすことは容易ではない．そのため，OpenMPD の設計に際しては直観的で簡便なプログラミングモデルを提供することを心がけた．

2　データ並列化

分散メモリにおけるデータ並列化の典型的な手法を図 1 に示す．まず，データを各ノードに分散させ，割り当てる．各ノードは割り当てられたデータに対して並列に計算を行う．計算の結果として得られるデータは計算されたノードでしか直接参照することができない．他のノードでそのデータを参照するためにはノード間通信を用いる必要がある（同期を行う）．

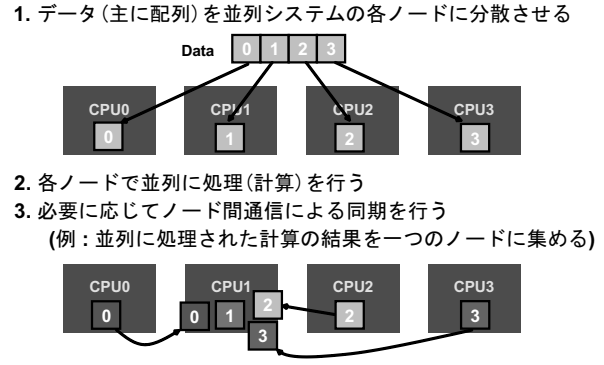

図 1: データ並列化の基本的手法

これはデータ並列化の基本的な手法であり，多くのアプリケーションの並列化に適用することができる．

3　OpenMPDの概要

OpenMPD は前章で述べたようなデータ並列化の典型的な手法を指示文で記述することができる．これは逐次コードにプログラマが明示的な指示文の記述で並列化を行うというモデルで，共有メモリの OpenMP に類似する．OpenMPD は C 言語の指示文による拡張として設計されており，実装した処理系は OpenMPD コードを MPI コードに変換するため，実行モデルは MPI 同様の SPMD(Single Program Multiple Data) である．図 2 にその記述例を示す．#pragma で始まる行は並列化のためにプログラマが挿入したものである．#pragma ompd distvar の記述で配列のデータを各ノードに割り当てる．for ループに指示文#pragma ompd for を記述し，ループの並列実行によるワークシェアリングを記述している．共有メモリの OpenMP と同様，正しい並列化のためにはプログラマが責任を持って適切な指示文を記述しなければならない．

MPI を用いた並列プログラミングでは並列化のため，元のコードに大量の修正を施す必要がある．その結果，プログラムが複雑になり生産性が低下する．指示文に

```
int array[10][10];

#pragma ompd distvar(var = array;dim = 2)

main(){
  int i,j;

#pragma ompd for affinity(array)
  for(i = 0; i < 10; i++)
    for(j = 0; j < 10; j++)
      array[i][j] = func(i, j);

#pragma ompd gather(var = array)
}
```

図 2: OpenMPD のプログラム例

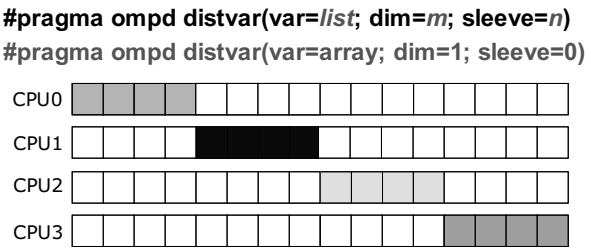

図 3: 配列の並列割り当て

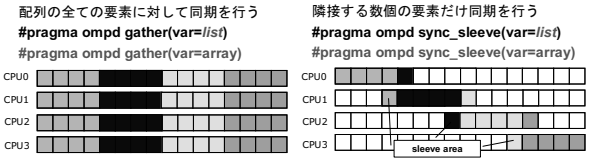

図 4: 配列の同期

よる並列化は元のコードに指示文を追加するだけで済むため，MPI のように並列化によってプログラムが複雑になることを避けることができる．その結果，MPI を使うより分散メモリ上の並列プログラミングが簡単になる．

OpenMPD の指示文はデータ並列プログラミングを支援する．それらは多くの科学計算アプリケーションのデータ並列化に用いられるものである．以下に OpenMPD の並列化手法とそれに対応する指示文の説明を行う．

4 OpenMPD による並列化の記述

4.1 配列の並列割り当てと同期

指示文 distvar によって配列の割り当てを行うことができる．図 3 のように配列 array を 4 ノードで並列に処理する時には各ノードで配列の全体領域を確保した後，異なる領域を処理するように制限することで（図 3 の灰色の部分のように）割り当てを行う．

以後，各ノードは自分に割り当てられた領域だけに対して計算を行い，正しい値を持つことが保証される．

各ノードが自分に割り当てられていない領域にアクセスするときには他のノードとの通信による同期が必要である．OpenMPD は配列の全ての領域の同期を取る gather と一部だけの同期を取る sync_sleeve 指示文を提供する．図 4 に同期の動作を示す．多くのアプリケーションでは自分に割り当てられた領域以外にその領域に隣接した数個の要素を参照している（ガウス・ザイデル法など）．その場合，4 のように自分に割り当てられた領域に隣接する数個分の要素の同期を取るだけで良い．sync_sleeve によってそのような同期を記述し，少ない通信で効率的な同期を行うことができる．gather による同期は配列の全ての要素に対して同期を行うので通信のコストが高いが，配列に対するどのようなアクセスパターンにも対応することができる．

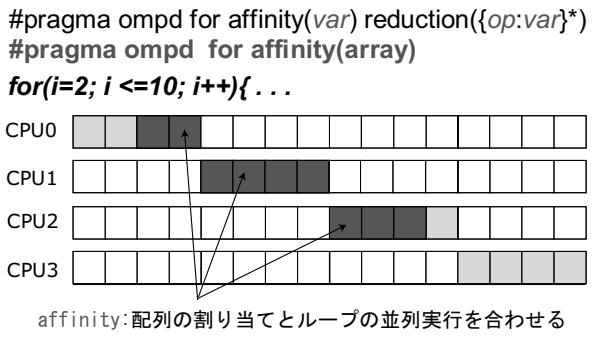

図 5: ワークシェアリング

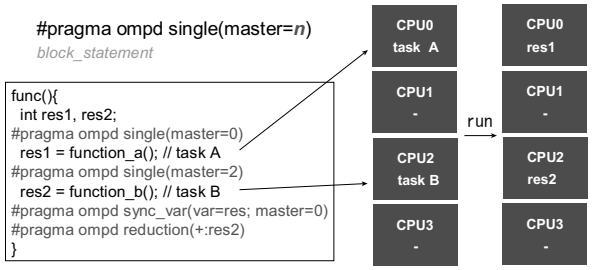

図 6: タスク並列化

4.2 ループの並列実行による ワークシェアリング

distvar によって割り当てた配列をループで並列に処理するために指示文 for を用いることができる．図 2.a にその動作を示す．配列の割り当てに合わせて（affinity(配列名) の記述による）for 文の作業を各ノードに割り当てている．このように for 文の作業を各ノードに分散させることで実行時間を短縮させることができる．

4.3 タスク並列化

プログラムを幾つかの独立した機能に分けられる場合，指示文 single によって並列化することができる．single は指定された機能（C 言語でいうとブロック文）を一つのノードだけで実行するようにする．各機能を異なるノードで実行させ，処理の高速化を図ることができる．但し，処理の結果は実行したノードのみで有効なので並列処理の後に変数の同期を行う必要がある．

4.4 変数の同期

配列の各要素の総和を求める for 文を並列化したとする．その場合，結果として得られるのは自分に割り当てられた配列要素の総和だけになってしまう．また，タスク並列化を行った場合，その結果を収納する変数は各ノード毎に独自の計算結果を持つようになる．SPMD モデルでは配列と同様，変数（配列以外のベーシックな変数）が各ノードで異なる値を持つ可能性があるため，通信による同期が必要である．OpenMPD は変数の同期に sync_var と reduction を提供する．sync_var は変数の値をあるノードの値で一致させるものである．reduction は総和のようなリダクション演算を記述するためのもので，for 文のオプションとして使うこともで

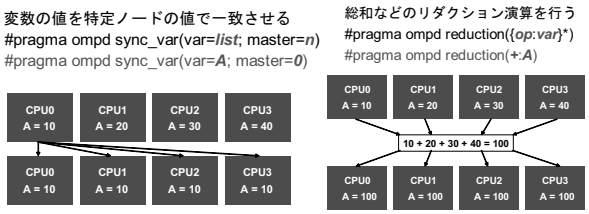

図 7: 変数の同期

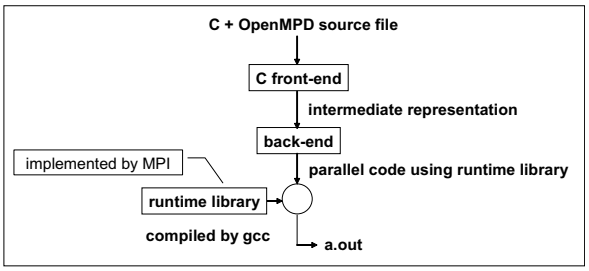

図 8: 処理系の概要

きる．

4.5 OpenMPD のライブラリ関数

OpenMPD は MPI と同様，SPMD 実行モデルで実行される．そのため，プログラマが全てのプロセッサの数と自分のプロセッサ番号を必要とする場合がある．実行環境に関する情報を取得するため，OpenMPD は以下の関数をプログラマに提供する．

- int ompd_get_num_nodes(void);
  ompd_get_num_nodes は全体のプロセッサ数を返す関数である．戻り値は 1 以上の整数である．
- int ompd_get_node_num(void);
  ompd_get_node_num は自分のプロセッサ番号を返す関数である．戻り値は 0 から (全プロセッサ数-1) の間の整数である．

5 処理系の実装

OpenMPD の処理系は C 言語の拡張として実装されている．Omni OpenMP Compiler をベースにして指示文の実装を行った．処理系は指示文によって得られた並列化に関する情報を用いて OpenMPD ソースを並列コードに変換する．

並列コードは通信ライブラリを用いてノード間通信による並列処理を行う．現在の実装では通信に MPI を用いるため，MPI コードが生成される．従ってユーザは性能の最適化のため，OpenMPD のコードの中で MPI 関数を使うことができる．

図 9 に図 2 から処理系によって変換された並列コードを示す．__ompd_で始まる変数は並列化に関する情報を収納するために処理系が宣言したものである．__ompd_で始まる関数は OpenMPD のランタイムライブラリ関数である．処理系は指示文から得られる情報を元に，プロセッサ間の通信や並列化に関する情報を計算する．

その後，得られた並列化情報を用いて図 3 のように配列の割り当てを行う（ブロック分割）．各プロセッサで割

```
int __ompd_idx_array_lower;
int __ompd_idx_array_upper;
void *__ompd_arg_array_arr_left;
extern void __ompd_init();
extern void __ompd_record_var_without_sync();
extern void __ompd_finalize();
int __ompd_myid;
int __ompd_nproc;
extern int __ompd_get_llimit_eq();
extern int __ompd_get_ulimit_neq();
extern void __ompd_allgather_n_n();
int array[10][10];

int main(argc,argv)
int argc;
char **argv;
{
 auto int __ompd_for_i_lower;
 auto int __ompd_for_i_upper;
 auto int i; auto int j;

 __ompd_init(&argc,&argv);
 __ompd_record_var_without_sync(
   &__ompd_idx_array_lower, &__ompd_idx_array_upper,
   10,array,10,4, &__ompd_arg_array_arr_left);

 __ompd_for_i_lower=
 __ompd_get_llimit_eq(0,__ompd_idx_array_lower);
 __ompd_for_i_upper=
 __ompd_get_ulimit_neq(10,__ompd_idx_array_upper);
 for(i=__ompd_for_i_lower;i<__ompd_for_i_upper;i+=1)
  for(j=0;j<10;j++){
   (*((*((array)+(i)))+(j)))=func(i,j);
  }

 __ompd_allgather_n_n(__ompd_arg_array_arr_left,
   (void *)(array),4,
   __ompd_idx_array_upper-__ompd_idx_array_lower,
   10,10);

 __ompd_finalize();
}
```

図 9: 処理系によって変換されたコード

り当てられる範囲の上限と下限を計算し，__ompd_idx_配列の名前_upper と __ompd_idx_配列の名前_lower に収納する．

for 文の並列実行はループ変数が持つ値を制限するようにして実現する．affinity が記述された場合は配列の割り当てに整合して，ループ変数の上限と下限を計算する．

配列や変数の同期に関する指示文が記述された場合はそのデータの構造（サイズ，型など）を解析し，同期関数を挿入する．

6 性能評価

ここでは実際のアプリケーションを OpenMPD で並列化し，その性能を評価する．性能評価には NAS Parallel Benchmark の The Embarrassingly Parallel Benchmark(以下 EP) と Conjugate Gradient Benchmark(以下 CG)，姫野ベンチマークを用いた．

図 10 のような構成の PC Cluster を 8 ノードを用いて性能評価を行った．

図 10 にコードのサイズを示す．並列化に大量のコード修正が必要な MPI と比べ，OpenMPD はごく僅かな変更だけで並列化することができる．これは並列化に必要なプログラマの作業量が減り，分散メモリの並列プログラミングを簡単に行えることを証明する．

図 11 に MPI 版と比べた OpenMPD のパフォーマン

■ ノードの構成

Item	Name
CPU	Intel Xeon Processor 2.4Ghz
Memory	1GB
Network	1000BASE-T
MPI	LAM 7.0.6
OS	Linux 2.6.9

■ 最大8ノードを使用

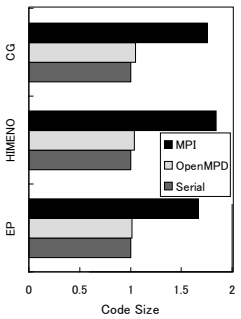

図 10: 評価環境・コードサイズ

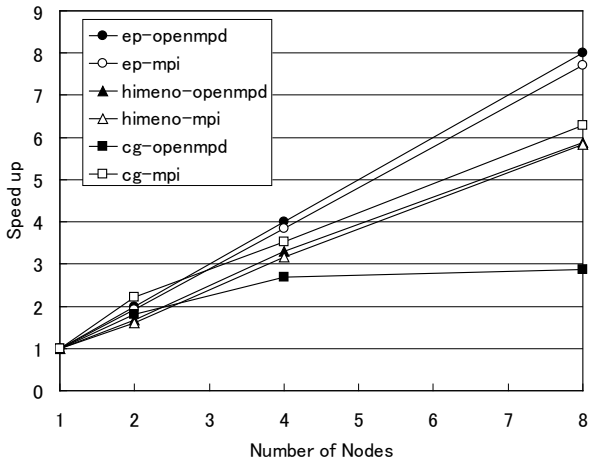

図 11: 評価結果

スを示す.

EP は乗算合同法による一様乱数，正規乱数の生成を行うベンチマークである．ループの各反復の実行を並列に処理することができる．通信がほとんど発生しないため，MPI 版と OpenMPD 版両方が理想的な性能を示す.

姫野ベンチマークはポアッソン方程式解法をヤコビの反復法で解くものである．OpenMPD 版は sync_sleeve による同期がアプリケーションと上手くマッチして効率的に通信を行うため，MPI 版とほぼ同じ性能を示す.

CG は大規模で正値対称な疎行列の，最小固有値の近似値を共役勾配法を用いて解くものである．OpenMPD 版の性能が MPI 版と比べ著しく悪いのは現在の OpenMPD の実装が配列の多次元分割をサポートしていないため，上手く並列化できないループが存在するからである．そのため，今後実装を見直す必要がある.

実行結果は OpenMPD が多くのアプリケーションの並列化を行うに十分な機能を提供することを実証している．このように，OpenMPD はデータ並列に関する典型的な並列化を支援するため，その指示文で多くのアプリケーションの並列化を記述することができる.

7 OpenMPD の問題点と課題

OpenMPD が持つ問題点を以下に示す.

- OpenMPD で配列を多次元で分割ことができない．プロセッサを有効に利用するためには多次元分割により，複数の次元の並列度を引き出すことが望ましい.
- OpenMPD で配列を並列処理するとき，全プロセッサに配列の全ての領域が宣言される．しかし，プロセッサが処理する領域はその中の一部であるため (図 3 参照)，無駄にメモリを消費することになる．分散メモリ型並列システムのメリットの一つは膨大なデータを各プロセッサに割り当て処理することができることである．現在の OpenMPD の仕様ではこのようなアーキテクチャの特徴を生かすことができない．各配列が割り当てられる部分だけを持つようにして，メモリを効率的に利用するようにしなければならない.
- 並列実行する for 文の中で異なる範囲で割り当てられた配列が参照される場合，affinity の指定が困難である．異なる範囲で割り当てられた配列を一緒の for 文で処理することができるように，データの整合性に関する規則とその実装が必要である.

8 まとめ

OpenMPD は MPI プログラミングを簡単に行う手段として，分散メモリ型並列システムに対する簡単なプログラミングモデルを提供する．多くの科学技術計算アプリケーションで用いられるデータ並列の典型的な手法を指示文を用いて記述することができる.

今後，多次元分割をサポートするようにして OpenMPD の機能を拡張していくと共に，ノード内並列化を OpenMP で記述した場合のスレッドセーフティを確保してハイブリッドなプログラミングモデルを提供することを目指す.

参考文献

[1] 太田 寛ほか, "HPF 処理系による NAS Parallel Benchmarks の並列化", 情報処理学会研究報告 (第 62 回 HPC 研究会), pp. 57-62, 1996

[2] 小島 好紀ほか, "MPI 上のソフトウェア分散共有メモリシステム", 情報処理学会研究報告 (第 98 回 HPC 研究会), pp. 43-48, 2004

[3] Mitsuhisa Sato et al., "Design of OpenMP Compiler for an SMP Cluster", Proc. of 1st European Workshop on OpenMP EWOMP'99, pp.32-39, 1999.

[4] Taisuke Boku et al., "OpenMPI - OpenMP like tool for easy programming in MPI", Proc. of 6th European Workshop on OpenMP (EWOMP'04), pp.83-88, 2004.

[5] Jinpil Lee et al., "Design and Implementation of OpenMPD", International Workshop on OpenMP 2007, pp. 132–135, 2007