B-026

ローカルメモリを越える大容量データを扱う逐次処理のための C コンパイラ
C Compiler for Large Data Sequential Processing over Computer Cluster

吉村 礎† 緑川 博子† 甲斐 宗徳†
Shiyo Yoshimura Hiroko Midorikawa Munenori Kai

1. はじめに

DLM(Distributed Large Memory)[1]とは，逐次プログラムの使用メモリ量が使用しているコンピュータの物理メモリ量を越えた時に，遠隔コンピュータ(クラスタなど)のメモリを通信によって利用し，大容量メモリとして逐次処理用に提供するシステムのことである．

逐次プログラムを利用する際，遠隔メモリを使用する研究には，カーネルレベル実装とユーザーレベル実装がある．カーネルレベル実装では，OS の書き換えなどが必要なため，性能を十分にあげることができなく，可搬性・可用性は低いが，ユーザに完全な透過性を与えてくれる．ユーザーレベル実装では，OS の書き換えをせずに，MPI や socket 通信などの汎用プロトコルを用いるだけで使用でき，そのため可搬性・可用性は高く，性能も高いことがわかっている[1]．しかし，一般的にはユーザのプログラム書き換えなどが必要になり，ユーザ透過性が低くなる．

ユーザーレベル実装の 1 つの，JumboMem[2]では，ユーザ透過性を実現するため，OS のメモリ関連関数コール(malloc など)を，動的ライブラリのロードパスを変更することにより，JumboMem 独自の共有ライブラリ関数コールに交換する方式をとっている．これにより，ユーザによるプログラムの書き換えは不要になるが，欠点が 2 点ある．1 点目は，動的メモリ確保関数にしか適用されないため，数値計算プログラムなどで用いられる静的な大規模配列宣言データには適用できない．2 点目は，ローカルメモリに確保されるべきバッファ領域なども遠隔メモリに取られてしまう可能性があり実用上問題がある．

DLM システムもユーザレベル実装であるが，DLM ではユーザによるプログラム書き換えを最小限にするため，動的メモリ確保と静的配列宣言の両方に対応できる API を提供する．本研究では，ユーザ透過性を高めるため，この API を実現する DLM コンパイラを作成した．

2. DLM システム

DLM システムは，図 1 に示すように，計算ホストノード(Cal Host node)でユーザプログラムを動かし，遠隔メモリが必要なときにのみ，メモリサーバノード(MemServer node)のメモリを借りてくる．データの通信は DLM ページサイズ(OS ページサイズの倍数の大きさ)という単位で行っている．通信は，TCP/IP または MPI を用いており，それらで動かすことのできる高速通信媒体(10Gbps Ethernet，Infiniband，Myri10G など)で利用できる．ユーザにとっては逐次プログラムの実行として見えるが，実際には，計算プロセス(Cal Process)とメモリサーバプロセス(Mem Server Process)の複数の並列プロセスを実行している．

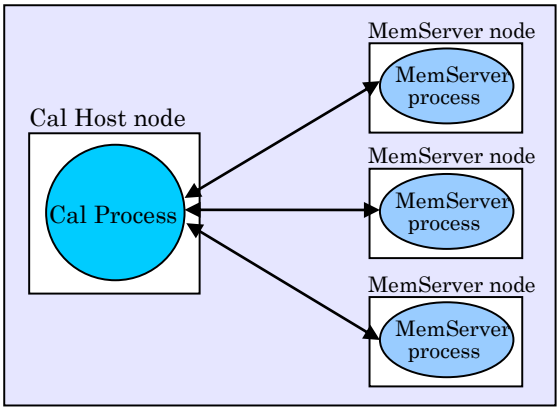

図 1　DLM システム

3. DLM におけるプログラムインターフェース

DLM の API の特徴は以下の 3 点である．

- ローカルメモリのみを使う通常変数とローカルメモリ不足時に遠隔メモリにも展開可能な変数(DLM データと呼ぶ)を，ユーザが「dlm」を付加するか否かで指定可能である．図 2 に例を示す．
- 静的データ宣言に「dlm」と付加するだけで DLM データとして使用可能になる．(図 2-①，②)
- 動的データ割付の malloc 関数を dlm_alloc 関数に書き換えるだけで DLM データとして使用可能になる．(図 2-③)

```
#include <dlm.h>
#define M 1000
#define N 100000

dlm int a[M][N]; ①

int main ( int argc,  char *argv[]){
   int i, k;
   dlm int c[M];  ②
   int * b;
   b = (int(*))dlm_alloc( M * sizeof(int));  ③
   for ( i = 0; i < M; i++)  {
      b[i] = i*2;
      c[i] = 0;
      for ( k = 0; k < N; k++)
          a[i][k] = i;
   }
   for ( i = 0; i < M; i++)  {
      printf("b = %d,  c = %d \n",  b[i],  c[i]);
      for ( k = 0; k < N; k++)
          printf("a = %d \n",  a[i][k]);
   }
   return 0;
}
```

図 2　DLM の API の使用例

† 成蹊大学 理工学研究科 理工学専攻　Graduate School of Science and Technology, Seikei University

4. DLM コンパイラの構造

図 3 に示すように，DLM コンパイラ(dlmc)では，DLM データが宣言されているプログラムをトランスレータに通して，通常の C 言語プログラムに変換する．次に，gcc コンパイラにより，dlm ライブラリをリンクし，実行ファイルを作成する．以下にプログラムコンパイルコマンドの例を示す．

```
dlmc “dlm プログラム名.c” -o “実行プログラム名” -ldlm
```

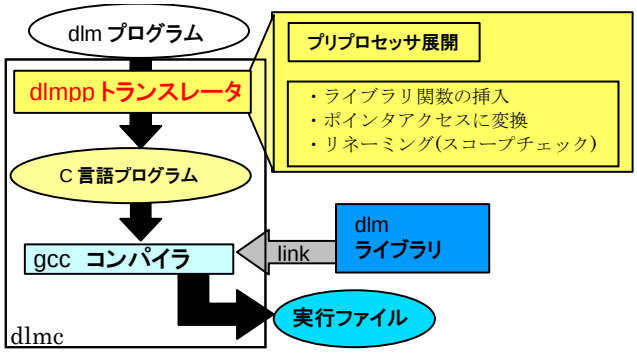

図 3　DLM コンパイラの構造

5. dlmpp トランスレータ

dlm プログラムは，通常の c 言語プログラムの文法に「dlm」を記憶指定子(extern，static など)として組み込んだ文法を利用して書かれている．dlmpp トランスレータにより図 2 のプログラムは図 4 のプログラムに変換される．

DLM プログラムをこのトランスレータに通すと，次のように挿入・変更が行われる．

- main 関数の変数宣言部分の後に dlm_startup 関数(DLM システムの起動関数)を挿入する．(図 4－①)
- プログラムを終了する前に dlm_shutdown 関数(DLM システムの終了関数)を挿入する．(図 4－②)
- DLM データの静的宣言の記述があれば，その変数をポインタアクセスの変数に変換し(図 4－③－1)，以降その変数名を「__dlm_変数名_ブロック番号」という変数名にリネーミングする(図 4－③－3)．DLM システムの dlm_alloc 関数を使い(図 4－③－2)動的にメモリを確保する．
- 関数やブロック内部の局所変数に dlm が宣言されている場合には，ブロックや関数の最後に dlm_free 関数も挿入する．

```
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・
int (*__dlm_a_0[100000];        ③－1

int main ( int argc,  char *argv[]){
   int i, k;
   int (*__dlm_c_1);
   int * b;
    dlm_startup(&argc,  &argv);    ①
   __dlm_a_0=(int(*)[100000])dlm_alloc(1000*100000*siz
eof(int));              ③－2
   __dlm_c_1 = (int(*))dlm_allloc(1000*sizeof(int));
   b = (int(*))dlm_alloc( M * sizeof(int));
   for ( i = 0; i <1000; i++)  {
      b[i] = i*2;
      __dlm_c_1[i] = 0;
      for ( k = 0; k < 100000; k++)
           __dlm_a_0[i][k] = i;      ③－3
   }
   for ( i = 0; i < 1000; i++)  {
      printf("b = %d,  c = %d \n",  b[i],  __dlm_c_1[i]);
       for ( k = 0; k < 100000; k++)
           printf("a = %d \n",  __dlm_a_0[i][k]);
   }
   dlm_shutdown();     ②
   return 0;
}
```

図 4　変換後の C 言語プログラム

6. DLM プログラムの実例

既存の数値計算のプログラムでは，静的な配列が使用されていることが多い．ここでは，姫野ベンチマークソースプログラム[3]を例にあげる．

静的領域の限度数を超える領域を宣言するときは，動的に malloc でメモリを確保することもできるが，多次元配列となってくるとポインタベースアクセスに変更するのは面倒である．しかしこの DLM コンパイラを使用すれば，「dlm.h」のヘッダを挿入し，図 5 の姫野ベンチマークプログラムのように，配列変数宣言の前に「dlm」と付加するだけで，それ以外の部分を変更せずに，既存プログラムが扱う問題の大規模化を図れる．

```
・・・・・・・・・・
dlm float p[MIMAX][MJMAX][MKMAX];
dlm float a[MIMAX][MJMAX][MKMAX];
      b[MIMAX][MJMAX][MKMAX];
      c[MIMAX][MJMAX][MKMAX];
dlm float bnd[MIMAX][MJMAX][MKMAX];
dlm float wrk1 [MIMAX][MJMAX][MKMAX];
      wrk2[MIMAX][MJMAX][MKMAX];
・・・・・・・・・・
```

図 5　姫野 DLM プログラム変更箇所

7. おわりに

DLM データは，dlm_alloc 関数により動的にメモリを確保する．dlm 宣言は，関数外部内部を問わず宣言することができる．従来のプログラムでは，大域変数(関数外部で宣言)は静的データ領域，関数内部変数はスタック領域で確保されるが，ローカルメモリサイズに収まる範囲であっても，コンパイラコード生成の制限やスタック領域サイズの制限などを受けて，大規模な配列データなどを宣言することができないことも多い．しかし，DLM データ宣言を利用することで，関数内外で宣言された配列データも，ローカルメモリサイズに制限されず(ローカルメモリが不足するときは遠隔メモリに展開し)，大規模化が可能である．PC などで小さいサイズでモデルやアルゴリズムを設計し，その後，並列プログラム化することなく，クラスタを利用して，逐次プログラムのまま問題の大規模化が可能であることが，大きな利点となる．

参考文献

[1] 緑川 博子，斉藤 和広，佐藤 三久，朴 泰祐 “クラスタをメモリ資源として利用するための MPI による高速大容量メモリ”，情報処理学会論文誌コンピューティングシステム，Vol.2，No.4 (2009).

[2] Scott Pakin，Greg Johnson，“Performance Analysis of a User-level Memory Server”，2007 IEEE International Conference on Cluster Computing (2007)

[3] Himeno Benchmark website[Online] (2009).available from http://accc.riken.jp/HPC/HimenoBMT.html (accessed 2010-06-23)