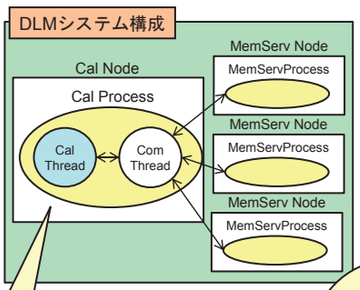
遠隔メモリ利用で大容量データ処理を可能にする逐次プログラムのためのCコンパイラ

吉村 礎，緑川 博子（成蹊大）

分散大容量メモリシステム (Distributed Large Memory System)

ネットワークで結ばれたコンピュータの物理メモリを通信によって利用し，逐次処理用に**仮想的に大容量のメモリ空間**を提供するシステム．

DLMシステム構成

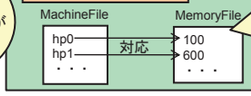


Cal Processで逐次プログラムが走る

DLMの遠隔Swap

計算ノード(Cal Node)で利用可能なメモリサイズを超えるメモリが必要な場合
→遠隔メモリ(MemServ)にデータを割り付ける
メモリサーバ上(MemServ Node)に割り付けたデータへアクセスする場合
→欲しいデータを持つメモリサーバ(MemServNode)からそのデータを含むDLMページを計算ノード(Cal Node)へ持ってきて(スワップイン)、不必要と思われるDLMページをメモリサーバ(MemServNode)へ送る(スワップアウト)。

DLM設定ファイル



MachineFile 使用ホストを指定
MemoryFile それぞれのサーバで利用可能メモリ量を指定 (単位はMB)

プログラム実行コマンド例

```
mpirun –np 2 –machinefile machinefile01 ./ sample -- -f memfile01
```

プロセッサ2個使用 / MachineFile指定 / MemoryFile指定

DLM コンパイラ

DLMシステムをユーザに使用しやすくするため開発．

定義

通常データ：ローカルメモリのみ使用する変数
DLMデータ：遠隔メモリにも展開可能な変数
DLMプログラム：DLMデータが宣言されているプログラム

DLMのAPIの例

静的データ宣言に適用
`dlm int a[2000];`
動的データ宣言に適用
`a = (int*)dlm_alloc(2000*sizeof(int));`

通常の変数宣言に「dlm」を付加するだけ。
malloc関数をdlm_alloc関数に書き換えるだけ。

プログラム例

通常のCプログラムに、DLMシステムを使用したい変数に「dlm」と付加するだけで、実行可能．
DLMプログラムをDLMコンパイラにかけるとC言語プログラムに変換される．

通常のCプログラム

```
#define MAX 1000
int a[MAX][MAX];

int main ( int argc, char *argv[])
{
    int i, j;
    for ( i = 0; i < MAX; i++)
        for ( j = 0; j < MAX; j++)
            a[i][j] = func();
    for ( i = 0; i < MAX; i++)
        for ( j = 0; j < MAX; j++)
            printf("%d", a[i][j]);
    return 0;
}

int func() {
    double b[MAX] = {1.0};
    int ans, i;
    for ( i = 0; i < MAX; i++) ans += b[i];
    return ans;
}
```

DLMプログラム

```
#define MAX 1000
dlm int a[MAX][MAX];

int main ( int argc, char *argv[])
{
    int i, j;
    for ( i = 0; i < MAX; i++)
        for ( j = 0; j < MAX; j++)
            a[i][j] = func();
    for ( i = 0; i < MAX; i++)
        for ( j = 0; j < MAX; j++)
            printf("%d", a[i][j]);
    return 0;
}

int func() {
    dlm double b[MAX] = {1.0};
    int ans, i;
    for ( i = 0; i < MAX; i++) ans += b[i];
    return ans;
}
```

グローバル変数指定可能
ローカル変数指定可能

変換後のC言語プログラム

```
int (*__dlm_a_0)[1000];

int main ( int argc, char *argv[])
{
    int i, j;
    dlm_startup(&argc, &argv);
__dlm_a_0 = ( int (*)[1000])dlm_alloc(1000*1000*sizeof( int ));
for ( i = 0; i < 1000; i++)
        for ( j = 0; j < 1000; j++)
            __dlm_a_0[i][j] = func();
    for ( i = 0; i < 1000; i++)
        for ( j = 0; j < 1000; j++)
            printf("%d", __dlm_a_0[i][j]);
    dlm_shutdown();
    return 0;
}

int func() {
    double (*__dlm_b_1);
    int ans, i;
    __dlm_b_1 = ( double (*))dlm_alloc(1000*sizeof( double ));
for ( i = 0; i < 1000; i++) ans += __dlm_b_1[i];
    dlm_free(__dlm_b_1);
return ans;
}
```

DLMコンパイラを使用しない場合はこのプログラムをユーザが書くことでDLMシステムを使用できる。

従来との特徴比較

変数	従来	DLM
静的変数	静的データ領域	ヒープ領域（遠隔メモリにも動的に展開）
関数内部自動変数	スタック領域	
動的メモリ割り当て	ヒープ領域	

既存のプログラムでローカルメモリサイズに影響をされず、データの大規模化が可能！ 利点

稼働実験

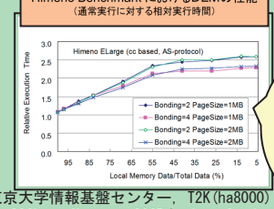
HimenoBenchmarkとNPBBenchmarkで変換・実行を確認．

HimenoBenchmark

```
dlm float p[MIMAX][MJMAX][MKMAX];
dlm float a[MIMAX][MJMAX][MKMAX],
    b[3][MIMAX][MJMAX][MKMAX],
    c[3][MIMAX][MJMAX][MKMAX];
dlm float bnd[MIMAX][MJMAX][MKMAX];
dlm float wrk1[MIMAX][MJMAX][MKMAX],
    wrk2[MIMAX][MJMAX][MKMAX];
```

たった4か所変更するだけで使用可！

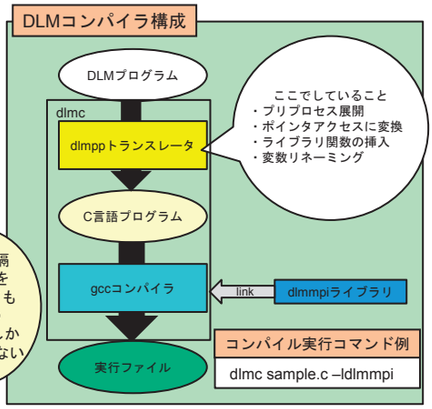
Himeno Benchmark におけるDLMの性能（通常実行に対する相対実行時間）



95%遠隔メモリを使用しても通常の約2.5倍しか遅くならない

実験環境：東京大学情報基盤センター，T2K(ha8000)
Network：40Gbps，20Gbps，Memory：20GB/node x 2nodes

DLMコンパイラ構成



ここでしていること
・プリプロセス展開
・ポインタアクセスに変換
・ライブラリ関数の挿入
・変数リネーミング

コンパイル実行コマンド例

```
dlmc sample.c –ldlmmpi
```