MPIBNCpack

幸谷智紀
tkouya@na-net.ornl.gov

Version 0.1: September 7, 2003

1 初めに

私 (幸谷) は，IEEE754 単精度・倍精度，及び GMP/MPFR の多倍長浮動小数点数を利用した 1CPU 用数値計算ライブラリ BNCpack[2] を作成し，リリースしました。幸い利用者は私以外の何方もいらっしゃらないようで，ほっと胸をなで下ろしております。その後は心おきなく，自分の好きなように機能拡張をして遊んでおりました。しかし，ある日，PC cluster と出会ってからジワジワと人生が狂い始め (大げさ？) たのです。GMP/MPFR を MPI と組み合わせて簡単に使えるライブラリ [1] をリリースしてからは，寝ても覚めても「多倍長」「並列分散」をブツブツと呟いて辺りを徘徊する怪しいオヤジと成り果てて，とうとうこの MPI 版 BNCpack(MPIBNCpack … 何の捻りもないネーミング) を公開するに至りました。

未だに非同期通信も扱えない MPI の新参者が作ったものであり，とりあえず動いているようなのでできている分だけまとめた，というだけの代物です。が，並列分散環境で多倍長数値計算を現に動かしている人はそれほどいらっしゃらないようなので，比較検討の材料ぐらいにはなるでしょう。

もし，真面目にこれを使ってシビアなシミュレーションを行いたい，という方がいらっしゃれば (いないと思うけど)，タイトルのメールアドレスまでご連絡下さい。

なお，このライブラリは，LGPL[5] にて基づいて公開されています。これを逸脱するような利用は慎んで下さい。また，本ライブラリは自己責任で使用して下さい。本ライブラリを利用した結果，被った損害について，著者一切関知いたしません。bug 報告は歓迎いたします。

2 MPIBNCpack の内部構造

2.1 GMP について

GMP は 1991 年から開発が進められてきた ANSI C とアセンブラで記述された多倍長計算ライブラリです。Version 4 になってからは，試験的ではありますが，以前より要望の強かった C++クラスインターフェースが追加されています。また，GMP とは別の Project で開発の進められてきた MPFR パッケージ [6] も含まれるようになっています。これは，GMP の mpf_t 型をベースにした mpfr_t 型を用いて，IEEE754 standard と互換性を持つように改良された 2 進浮動小数点演算ライブラリです。丸めモードの変更が可能で，高速な初等関数も提供されており，現時点での BNCpack 及び MPIBNCpack はこれをデフォルトに用いています。つーか，もう MPFR 以外のデータ型は相

手にしないことにしています。面倒ですから。MPFR の機能については，Web インターフェースが公開されているので，それで試してみて下さい [8]。

図 1 にこれら二つのデータ型の構成を示しますが，これは現行の Ver.4.1.2 に基づくもので，将来変更される可能性があることはご承知おき下さい。多分ないと思いますけどねぇ。

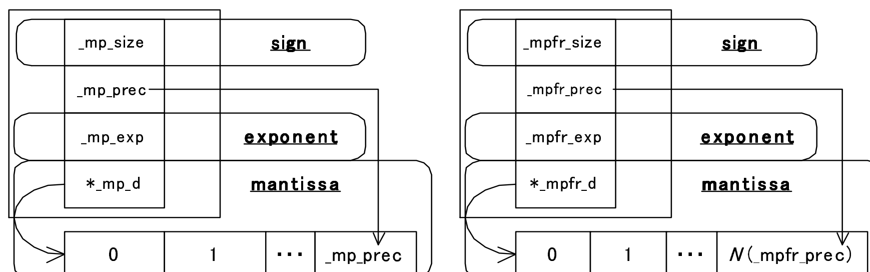

図 1: ”mpf_t”型 (in GMP) と”mpfr_t”型 (in MPFR) の構造

どちらも拡張された符号部 (sign)・指数部 (exponent)・仮数部 (mantissa) を持った構造体として定義されており，精度の増減は構造体内部のポインタ (_mp_d/_mpfr_d) が指定しているアドレス位置に存在する仮数部の長さを調整することで行っています。

2.2 MPI で MPFR を使用する方法

昔のスーパーコンピュータは，今以上に Job の制限が厳しく，時間あたり何円という課金体制をとっていました。従って，めったやたらに時間のかかる，成功するかどうかもわからない計算を湯水のように実行することは，よほどのお金持ち研究室でなければ不可能でした。その欲求不満からか，性能は悪くても安価で占有できる並列分散環境が望まれ，現在では PC を多数つなげた環境が普及しています。地球シミュレータのような例外的な存在もありますが，我々一般ビンボー研究者にとっては，あーゆー雲の上の存在は指銜えて奉っておくべきものでしかありません。日々，Linus 様 Stallman 様 OpenSourceCommumity の Hackers 様に感謝の念を捧げつつ，やれ電源が逝かれたの，HDD がすっ飛んだの，Hub の特定ポートが通信不能になったのとトラブルが続く PC Cluster と格闘するのが，正しいビンボー研究者のあり方です。

では，多倍長浮動小数点数をどのように PC Cluster で並列分散処理したらいいのでしょうか。

実際に数値計算に必要とされる桁数はそれほど多くはありません。IEEE754 倍精度の 2 倍の精度，4 倍の精度, 8 倍, 16 倍 ··· といった程度で十分だというのが巷の声です。また，PC cluster で用いる Ethernet では，1 フレームのサイズ制限が 1500bytes となっていますが，これに収まる 2 進浮動小数点数の有効桁数は 10 進数換算で 3000 桁を超えてしまいます。これだけの桁数を必要とするケースはあまりありません。また通信量は少ないに越したことはありませんから，1 フレームに収まるサイズの浮動小数点数は分割せずに処理した方が好都合です。何よりも作るのが簡単，ラクショーです。

というわけで，実装する並列分散プログラムでの多倍長浮動小数点数は，GMP の mpf_t 型もしくは mpft_t 型をそのまま使用し，各 PE(Processor Element) 間のやりとりの際にもこれらのデータ型をそのままバッファに pack して使うようにしました。この流れを図 2 に示します。例えば PE0 から mpfr_t 型のデータを送信し，PE1 でそれを受信する場合は，PE0 は送信前にデータを void 型ポ

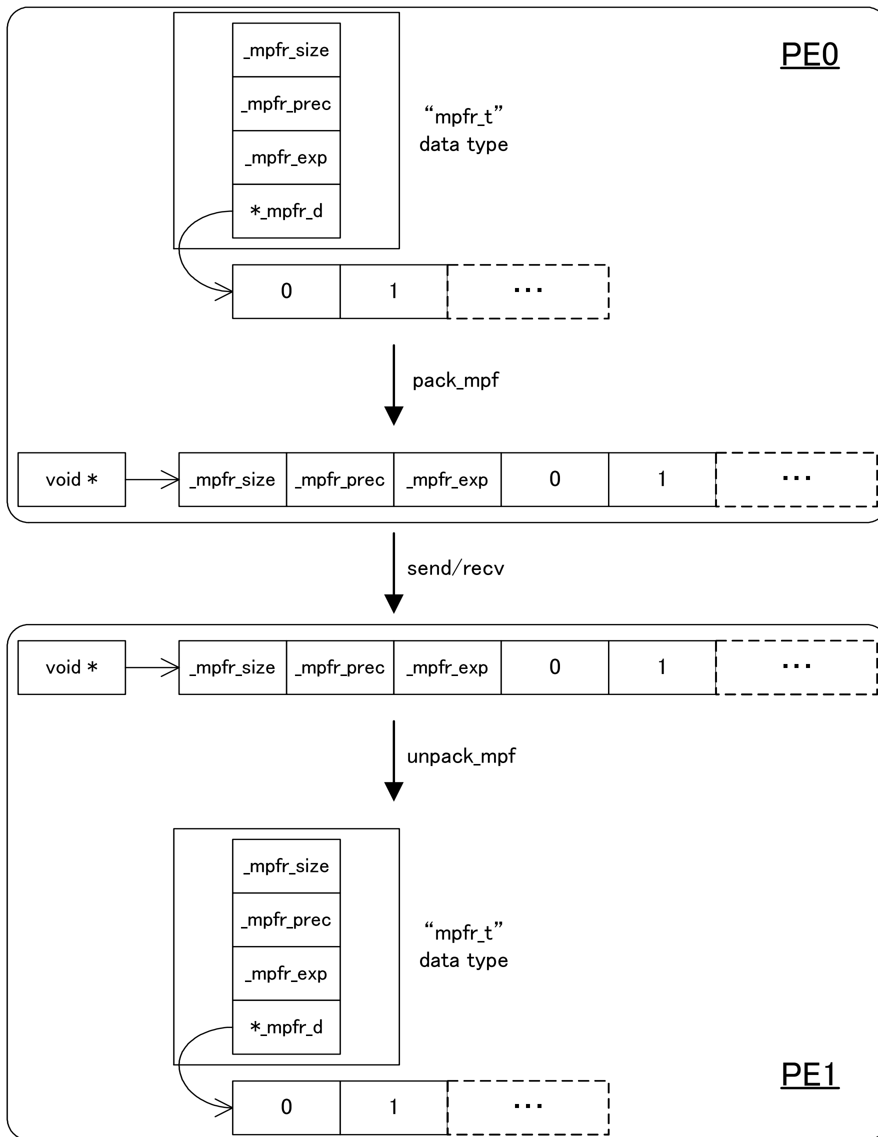

図 2: 一個分の mpfr_t データを送受信する処理

インタで指定されたバッファにパック (pack_mpf 関数を使用) し，PE1 は受信したバッファからデータをアンパック (unpack_mpf 関数を使用) します。この一連の操作を行うための関数群は MPIGMP Library[1] として公開済みですので，詳細はそちらを参照して下さい。また，このライブラリは本 MPIBNCpack にも同胞されています。

開発環境は，以下に示す PC Cluster，9PEs の cs-pccluster とソフトウェアを用いました。度重なる設置場所変更と，途中，学生実験にも利用される等，極めて乱暴な扱いをされたにもかかわらず，せいぜい電源と HDD をすっ飛ばしたぐらいで何とか動いてくれているこの cs-pccluster に感謝いたします。

ハードウェア

CPU Intel Pentium III 1GHz/Celeron 1GHz 合計 9 台

Ethernet 100BASE-TX + 24port Switch(NFS/mpich 共用)

ソフトウェア

OS Vine Linux 2.6r1

MPI mpich 1.2.5-1 (ch_p4, rsh 使用)

C compiler gcc 3.2.2

GMP GMP 1.4.2 (mpfr_t 型を使用)

cs-pccluster の構築方法については，私が書いた文書 [3]，及びその巻末に挙げた参考文献を参照して下さい。大して難しいことはしていません。

2.3 MPIBNCpack の構造

ということで，MPI を使って並列分散処理を行う環境は整いました。あとは，BNCpack に実装したアルゴリズムのうち，並列化に向いたものを pickup して，実装するだけです。これは案外楽な作業でした。何せ，データは単純に PE に分割して割り振っていくだけで，アルゴリズムもそれに対応すればいいだけなのですから，難しいことは何もないわけです。並列分散化を目指して 2ヶ月程 (2003 年 3 月〜4 月) で，大枠は完成しました。

早く出来上がったのは，自分で作った部分がごく少ないということも影響しています。図 3 に，MPIBNCpack の構造を示します。

お分かりのように，私がコードを書いたのは BNCpack と MPIBNCpack の部分だけです。後はぜーんぶ他の方が作った代物ですから，早く出来ても大して自慢にはなりゃしません。ここでもう一度，土台部分を作り上げた皆様方に感謝申し上げます。

以下，現時点で「使ってもらっても，まあ大丈夫じゃないの？(ドキドキ)」と思い切った関数だけのリファレンスを示します。ソースを見ていただければ分かるとおり，自分だけ使えればいーや的な，未公開部分やコメントがワンサと残っています。利用するのは勝手ですが，公開部分も含めて何が起こっても著者は保証しません。繰り返しになりますが，自己責任でお願い致します。

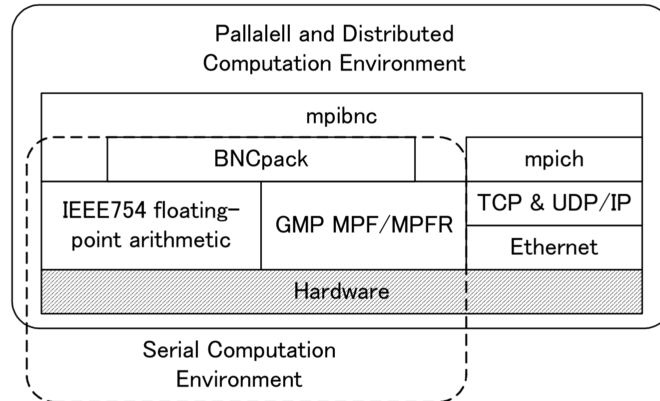

図 3: MPIBNCpack の構造

2.4 利用に当たっての基本事項

MPIBNCpack，特に多倍長浮動小数点数を使う場合は，MPIGMP Library が必須です。必ず以下のオマジナイを実行して下さい。MPI そのものについての解説は，専門の書籍 [7] 等を参考にして下さい。

1. 必ず次のヘッダファイルを include して下さい。また，それぞれに対応するライブラリもあらかじめ作成しておき，リンクして下さい。

```
#include <stdio.h>
#include <math.h> // -lm
#include "mpi.h" // MPI用
#include "bnc.h" // BNCpack用 -> libbnc.a
#include "mpi_gmp.h" // MPIGMP Library用 -> libmpfr.a, libgmp.a
#include "mpi_bnc.h" // MPIBNCpack用 -> libmpibnc.a
```

2. まず，MPI の初期化を行って下さい。その際，コミュニケータ (MPI_COMM_WORLD)，プロセス数 (numprocs)，自プロセスのランク (myid) を必ず取得して下さい。

```
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    MPI_Get_processor_name(processor_name, &namelen);
```

3. 次に，そのコミュニケータ内で有効な多倍長型を宣言するために，精度を 2 進数桁 (bit 数) で指定します。ここでは 256 桁になります。ついでに，reduce 関数で使用する演算を宣言します。現時点では加法以外の演算はサポートしていません。

```
#define MPF_PREC 256
    mpf_set_default_prec(MPF_PREC);
```

```
commit_mpf(&(MPI_MPF), MPF_PREC, MPI_COMM_WORLD);
create_mpf_op(&(MPI_MPF_SUM), _mpi_mpf_add, MPI_COMM_WORLD);
```

4. 各プロセスで実行する計算を記述した後，通信を行います。その際には

   (a) 多倍長型を pack するためのバッファを allocbuf_mpf 関数で確保する。

   (b) 多倍長型を pack 関数で確保したバッファに pack する。

   (c) バッファに格納されたデータを使って送受信処理を行う。

   (d) 受信したバッファのデータを unpack 関数で多倍長型に戻す。

   という手順で行って下さい。送受信されるデータは，あくまでバッファ内 (`void *`型) に pack されたものだけです。ここでは多倍長型変数 `mpf_mypi` と `mpf_pi` をそれぞれバッファ `packed_mpf_mypi` と `packed_mpf_pi` に格納して MPI_Reduce 関数を呼び出して `packed_mpf_pi` に加算していき，それを元の `mpf_pi` に unpack しています。

```
    packed_mpf_mypi = allocbuf_mpf(mpf_get_prec(mpf_mypi), 1);
    packed_mpf_pi = allocbuf_mpf(mpf_get_prec(mpf_pi), 1);
    pack_mpf(mpf_mypi, 1, packed_mpf_mypi);
    pack_mpf(mpf_pi, 1, packed_mpf_pi);
    MPI_Reduce(packed_mpf_mypi, packed_mpf_pi, 1, MPI_MPF, MPI_MPF_SUM, 0, M
PI_COMM_WORLD);
    unpack_mpf(packed_mpf_pi, mpf_pi, 1);
```

5. MPI 終了処理を行います。宣言した多倍長型と加算処理を解放しています。

```
free_mpf(&(MPI_MPF));
free_mpf_op(&(MPI_MPF_SUM));
MPI_Finalize();
```

以上の処理を前提として，MPIBNCpack の関数は動作するようになっています。

3 MPIGMP Library mpi_gmp.h, mpi_gmp.c

これらの関数群を利用したサンプルプログラムは cpi-gmp.c です。

void *allocbuf_mpz(mpz_t `zdata`**)**

多倍長整数型 `zdata` を格納できるバッファを確保し，そのポインタを返します。

void *allocbuf_mpq(mpq_t `qdata`**)**

多倍長有理数型 `qdata` を格納できるバッファを確保し，そのポインタを返します。

void *allocbuf_mpf(unsigned long `prec`**, int** `incount`**)**

`prec` ビットの多倍長型を `incount` 個分繋げたデータ型を格納できるバッファを確保し，そのポインタを返します。解放するときには標準の free 関数を使います。

long int _mpi_divide_dim(long int d_dim[]**, long int** dim**, int** num_procs**)**

配列，ベクトル，行列を num_procs プロセスに分割する際，トータルの次元数 dim から各プロセスに割り当てる次元数を計算し，配列 d_dim[] に格納します。集団通信の都合上，各プロセスは全て返値の次元数の割り当てを受けますが，実際には d_dim[] に格納された次元数だけが使用されることになります。

size_t get_bufsize_mpf(mpf_t fdata**, int** incount**)**

incount 個の多倍長型 fdata を格納するためのバッファサイズを計算して返します。

size_t get_bufsize_mpz(mpz_t zdata**)**

多倍長整数型 zadata を格納するためのバッファサイズを計算して返します。

size_t get_bufsize_mpq(mpq_t qdata**)**

多倍長有理数型 qadata を格納するためのバッファサイズを計算して返します。

void pack_mpf(mpf_t a**, int** incount**, void** *buf**)**

incount 個分の多倍長型変数 a をバッファbuf にパックします。

void pack_mpz(mpz_t a**, void** *buf**)**

多倍長整数型変数 a をバッファbuf にパックします。

void pack_mpq(mpq_t a**, void** *buf**)**

多倍長有理数型変数 a をバッファbuf にパックします。

void unpack_mpf(void *buf**, mpf_t** ret**, int** count**)**

バッファbuf のデータを incount 個分の多倍長型変数 a にアンパックします。

void unpack_mpz(void *buf**, mpz_t** ret**)**

バッファbuf のデータを多倍長整数型変数 a にアンパックします。

void unpack_mpq(void *buf**, mpq_t** ret**)**

バッファbuf のデータを多倍長有理数型変数 a にアンパックします。

void commit_mpf(MPI_Datatype *mpi_mpf_t**, unsigned long** prec**, MPI_Comm** comm**)**

コミュニケータ comm 内で，精度 prec ビットの MPI 用多倍長型 mpi_mpf_t を宣言します。

void create_mpf_op(MPI_Op *mpi_mpf_op**, void (***func**)(void *, void *, int *, MPI_Datatype *), MPI_Comm** comm**)**

コミュニケータ comm 内で，MPI の集団通信用演算関数 mpi_mpf_op を宣言します。

void free_mpf(MPI_Datatype *mpi_mpf_t**)**

MPI 用多倍長型 mpi_mpf_t を解放します。

void free_mpf_op(MPI_Op *mpi_mpf_op**)**

MPI の集団通信用演算関数 mpi_mpf_op を解放します。

void _mpi_mpf_add(void *in**, void** *ret**, int** *len**, MPI_Datatype** *datatype**)**

MPI の集団通信用加算関数です。

4 複素数 mpi_complex.c

これらの関数群を使用したプログラム例は test_mpidka.c です。

MPI_BNC_MPFCMPLX

MPI 用の多倍長複素数型の宣言です。

void commit_mpi_mpfcmplx(MPI_Datatype *`mpfcmplx_t`, unsigned long `prec`, MPI_Comm `comm`)

コミュニケータ comm 内で有効な，prec ビットの MPI 用多倍長複素数型 mpfcmplx_t を宣言します。

void free_mpi_mpfcmplx(MPI_Datatype *`mpfcmplx_t`)

MPI 用多倍長複素数型 mpfcmplx_t を解放します。

size_t get_bufsize_mpfcmplx(MPFCmplx `a`, int `incount`)

incount 個分の多倍長複素数型変数 a を格納するのに必要なバッファサイズを計算して返します。

void *allocbuf_mpfcmplx(unsigned long `prec`, int `incount`)

prec ビットの多倍長複素数型を incount 個格納するのに必要なバッファを確保し，そこへのポインタを返します。

int pack_cmpfarray(CMPFArray `array`, void *`buf`)

多倍長複素数配列型 array をバッファbuf にパックします。

void unpack_cmpfarray(void *`buf`, CMPFArray `array`, long int `size`)

バッファbuf のデータを，次元数 size の多倍長複素数型変数 array にアンパックします。

void *allocbuf_dcmplx(int `incount`)

incount 個の IEEE754 倍精度複素数を格納するのに必要なバッファを確保し，そこへのポインタを返します。

int pack_cdarray(CDArray `array`, void *`buf`)

IEEE754 倍精度複素数配列 array をバッファbuf にパックします。

void unpack_cdarray(void *`buf`, CDArray `array`, long int `size`)

バッファbuf のデータを，次元数 size の IEEE754 倍精度複素数配列 array にアンパックします。

5 基本線型計算 mpi_linear.c

これらの関数群を使用したプログラム例は test_mpilinear.c です。

DVector _mpi_init_dvector(long d_dim[]**, long int** dimension**, MPI_Comm** comm**)**

dimension 次元の IEEE754 倍精度ベクトルをコミュニケータ comm 内に確保し，各プロセスに d_dim[] 次元分割り当てます。

void _mpi_free_dvector(DVector vec**)**

コミュニケータ comm 内で確保された IEEE754 倍精度ベクトル vec を解放します。

void _mpi_divide_dvector(DVector d_vec**, long int** d_dim[]**, DVector src_vec, MPI_Comm** comm**)**

IEEE754 倍精度ベクトル src_vec を，コミュニケータ comm 内のプロセスの d_vec に d_dim[] 次元分ずつ分割して割り当てます。

void _mpi_collect_dvector(DVector src_vec**, long int** d_dim[]**, DVector** d_vec**, MPI_Comm** comm**)**

分割された IEEE754 倍精度ベクトル d_vec を，ランク 0 のベクトル src_vec にまとめます。

void _mpi_init_dmatrix(DMatrix ret[]**, long** d_dim[]**, long int** dimension**, MPI_Comm** comm**)**

dimension 次元の IEEE754 倍精度行列をコミュニケータ comm 内に確保し，各プロセスに d_dim[] 次元分，行ごとに割り当てます。

void _mpi_free_dmatrix(DMatrix mat[]**, MPI_Comm** comm**)**

コミュニケータ comm 内で確保された IEEE754 倍精度行列 mat[] を解放します。

void _mpi_divide_dmatrix(DMatrix d_mat[]**, long int** d_dim[]**, DMatrix** src_mat**, MPI_Comm** comm**)**

IEEE754 倍精度行列 src_mat を，コミュニケータ comm 内のプロセスの d_mat[] に d_dim[] 次元分ずつ分割して割り当てます。

void _mpi_collect_dmatrix(DMatrix src_mat**, long int** d_dim[]**, DMatrix** d_mat[]**, MPI_Comm** comm**)**

分割された IEEE754 倍精度行列 d_mat[] を，ランク 0 の行列 src_mat にまとめます。

double _mpi_ip_dvector(DVector in_a**, DVector** in_b**, MPI_Comm** comm**)**

IEEE754 ベクトル in_a と in_b の内積を並列計算します。

void _mpi_mul_dmatrix_dvec(DVector ret**, DMatrix** a[]**, DVector** x**, DVector** x_all**, MPI_Comm** comm**)**

IEEE754 倍精度行列 a とベクトル x の積を並列計算します。

MPFVector _mpi_init_mpfvector(long d_dim[]**, long int** dimension**, MPI_Comm** comm**)**

dimension 次元の多倍長ベクトルをコミュニケータ comm 内に確保し，各プロセスに d_dim[] 次元分割り当てます。

void _mpi_free_mpfvector(MPFVector vec**)**

コミュニケータ comm 内で確保された多倍長ベクトル vec を解放します。

void _mpi_divide_mpfvector(MPFVector d_vec**, long int** d_dim[]**, MPFVector** src_vec**, MPI_Comm** comm**)**

多倍長ベクトル src_vec を，コミュニケータ comm 内のプロセスの d_vec に d_dim[] 次元分ずつ分割して割り当てます。

void _mpi_collect_mpfvector(MPFVector src_vec**, long int** d_dim[]**, MPFVector** d_vec**, MPI_Comm** comm**)**

分割された多倍長ベクトル d_vec を，ランク 0 のベクトル src_vec にまとめます。

void _mpi_init_mpfmatrix(MPFMatrix ret[]**, long** d_dim[]**, long int** dimension**, MPI_Comm** comm**)**

dimension 次元の多倍長行列をコミュニケータ comm 内に確保し，各プロセスに d_dim[] 次元分，行ごとに割り当てます。

void _mpi_free_mpfmatrix(MPFMatrix mat[]**, MPI_Comm** comm**)**

コミュニケータ comm 内で確保された多倍長行列 mat[] を解放します。

void _mpi_divide_mpfmatrix(MPFMatrix d_mat[]**, long int** d_dim[]**, MPFMatrix** src_mat**, MPI_Comm** comm**)**

多倍長行列 src_mat を，コミュニケータ comm 内のプロセスの d_mat[] に d_dim[] 次元分ずつ分割して割り当てます。

void _mpi_collect_mpfmatrix(MPFMatrix src_mat**, long int** d_dim[]**, MPFMatrix** d_mat[]**, MPI_Comm** comm**)**

分割された多倍長行列 d_mat[] を，ランク 0 の行列 src_mat にまとめます。

void _mpi_ip_mpfvector(mpf_t ret**, MPFVector** in_a**, MPFVector** in_b**, MPI_Comm** comm**)**

多倍長 4 ベクトル in_a と in_b の内積を並列計算します。

void _mpi_mul_mpfmatrix_mpfvec(MPFVector ret**, MPFMatrix** a[]**, MPFVector** x**, MPFVector** x_all**, MPI_Comm** comm**)**

多倍長行列 a とベクトル x の積を並列計算します。

6 Krylov 部分空間法 mpi_cg.c

これらの関数群を使用したプログラム例は test_mpicg.c です。

long int _mpi_DCG(DVector local_answer**, DMatrix** local_a[]**, DVector** local_b**, double** reps**, double** aeps**, long int** maxtimes**, long int** dim**, MPI_Comm** comm**)**

IEEE754 倍精度で Conjugate-Gradient 法を並列に実行します。

long int _mpi_MPFCG(MPFVector local_answer**, MPFMatrix** local_a[]**, MPFVector** local_b**, mpf_t** reps**, mpf_t** aeps**, long int** maxtimes**, long int** dim**, MPI_Comm** comm**)**

多倍長で Conjugate-Gradient 法を並列に実行します。

7 DKA 法 mpi_dka.c

これらの関数群を使用したプログラム例は test_mpidka.c です。

void _mpi_ddka_init(CDArray `local_x_init`**, DPoly** `func`**, MPI_Comm** `comm`**)**

Aberth の初期値を IEEE754 倍精度で計算し，コミュニケータ comm 内のプロセスに配置します。

void _mpi_ddka(long int `*lasttimes`**, CDArray** `ans`**, CDArray** `local_ans`**, CDArray** `x_init`**, CDArray** `local_x_init`**, DPoly** `func`**, long int** `maxtimes`**, double** `abs_eps`**, double** `rel_eps`**, MPI_Comm** `comm`**)**

コミュニケータ comm 内で，Durand-Kerner 法を IEEE754 倍精度で並列に実行します。

void _mpi_mpf_dka_init(CMPFArray `local_x_init`**, MPFPoly** `func`**, MPI_Comm** `comm`**)**

Aberth の初期値を多倍長で計算し，コミュニケータ comm 内のプロセスに配置します。

void _mpi_mpf_dka(long int `*lasttimes`**, CMPFArray** `ans`**, CMPFArray** `local_ans`**, CMPFArray** `x_init`**, CMPFArray** `local_x_init`**, MPFPoly** `func`**, long int** `maxtimes`**, mpf_t** `abs_eps`**, mpf_t** `rel_eps`**, MPI_Comm** `comm`**)**

コミュニケータ comm 内で，Durand-Kerner 法を多倍長で並列に実行します。

8 数値積分 mpi_integral.c

これらの関数群を使用したプログラム例は test_mpiint.c です。

void _mpi_dtrapezoidal_fs(double `*ptr_ret`**, double** `x_start`**, double** `x_end`**, double (***`func`**)(double x), long int** `num_div`**, MPI_Comm** `comm`**)**

コミュニケータ comm 内で，数値積分を台形則に基づいて IEEE754 倍精度で並列に実行します。

void _mpi_mpf_trapezoidal_fs(mpf_t `ret`**, mpf_t** `x_start`**, mpf_t** `x_end`**, void (***`func`**)(mpf_t, mpf_t), long int** `num_div`**, MPI_Comm** `comm`**, MPI_Datatype** `mpi_mpf_type`**)**

コミュニケータ comm 内で，数値積分を台形則に基づいて多倍長で並列に実行します。

9 データ送受信 mpi_bcastbnc.c

これらの関数群を使用したプログラム例は test_mpidka.c です。

void _mpi_bcast_dpoly(DPoly `poly`**, MPI_Comm** `comm`**)**

コミュニケータ comm 内のプロセスに IEEE754 実係数多項式 poly をブロードキャストします。

void _mpi_bcast_mpfpoly(MPFPoly `poly`**, MPI_Comm** `comm`**)**

コミュニケータ comm 内のプロセスに多倍長実係数多項式 poly をブロードキャストします。

謝辞

本ライブラリは静岡理工科大学の援助を受けて作成されました。そのことに感謝いたします。また，作成するに当たり使用した hardware, software の開発者の方々にも感謝いたします。そして，いつも七面倒な議論に付き合っていただいている永坂秀子先生と,「計算ソフトとその周辺ワークショップ」参加者の方々にも合わせて感謝いたします。でも一番偉いのは，何の業績にもならないのにこんな文書をボランティアで書いている私です (誰も言ってくれないから自分で褒めてやる)。

参考文献

[1] MPIGMP Library, http://na-inet.jp/na/bnc/mpigmp.tar.gz

[2] BNCpack, http://na-inet.jp/na/bnc/

[3] 幸谷智紀, Vine Linux による PC Cluster の構築, http://na-inet.jp/na/mpipc.pdf

[4] GNU MP, http://swox.com/gmp/

[5] Lesser GNU Public License, http://www.gnu.org/copyleft/lesser.html

[6] MPFR Project, http://www.mpfr.org/

[7] P.Pacheco/秋葉博 訳, MPI 並列プログラミング, 培風館, 2001.

[8] Try MPFR!, http://www.jpsearch.net/try_mpfr.html