G5 ユーザガイド
version 2.0

株式会社 K & F Computing Research
E-mail: support@kfcr.jp

Abstract

この文書では G5 の詳細について説明します。G5 とは GRAPE-7 アドインカードに書き込む内部回路の一種です。

目 次

1 GRAPE-7 の概要

1.1 アーキテクチャ

GRAPE-7 は FPGA を搭載した、内部回路の書き換えが可能なアドインカードです。カードをホスト計算機 (PC) の PCI-X スロットに挿すことにより、ホスト計算機の計算能力が強化されます。

本節では GRAPE-7 のアーキテクチャを概観します。GRAPE-7 には model 100, model 300, model 600 の 3 種類のモデルがあります (図 1 参照)。model 100 のアーキテクチャは model 300 や 600 とは大幅に異なるため、両者を別個に説明します。

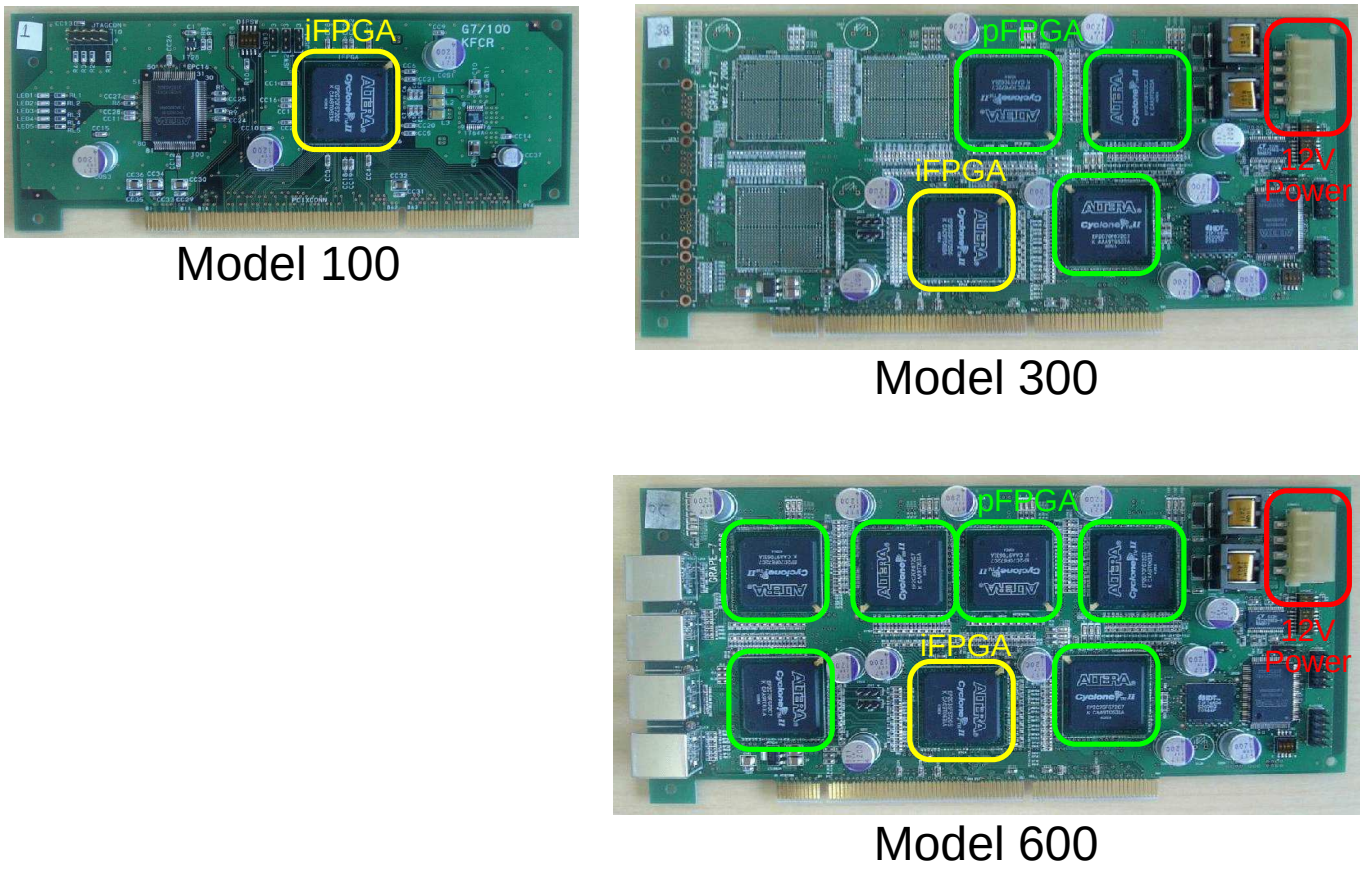

図 1: GRAPE-7 model 100、および 300、600 の写真。

1.1.1 Model 100

図 2 に GRAPE-7 model 100 のブロック図を示します。アドインカードは FPGA をひとつだけ搭載しています。これを **iFPGA** と呼ぶことにします。iFPGA には任意の計算資源と、ホスト計算機へのインタフェース回路 (XHIB) が書き込まれています。iFPGA の内部回路は当社によりあらかじめ書き込み済みであり、ユーザはこれを上書きすることは出来ません。

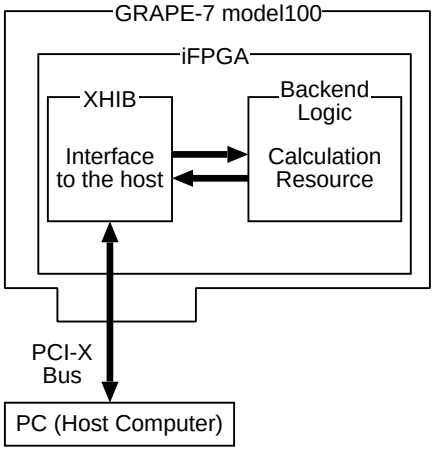

図 2: GRAPE-7 model 100 の内部構成。

1.1.2 Model 300 および model 600

図 3 に GRAPE-7 model 300 および model 600 のブロック図を示します。アドインカードは iFPGA をひとつと、**pFPGA** を複数搭載しています。iFPGA はホスト計算機とのデータ転送を制御します。内部回路は当社によりあらかじめ書き込み済みであり、ユーザはこれを上書きすることは出来ません。pFPGA は任意の計算資源として機能します。内部回路はユーザによって書き換えられます。model 300 は 3 個、model600 は 6 個の pFPGA を搭載しています。

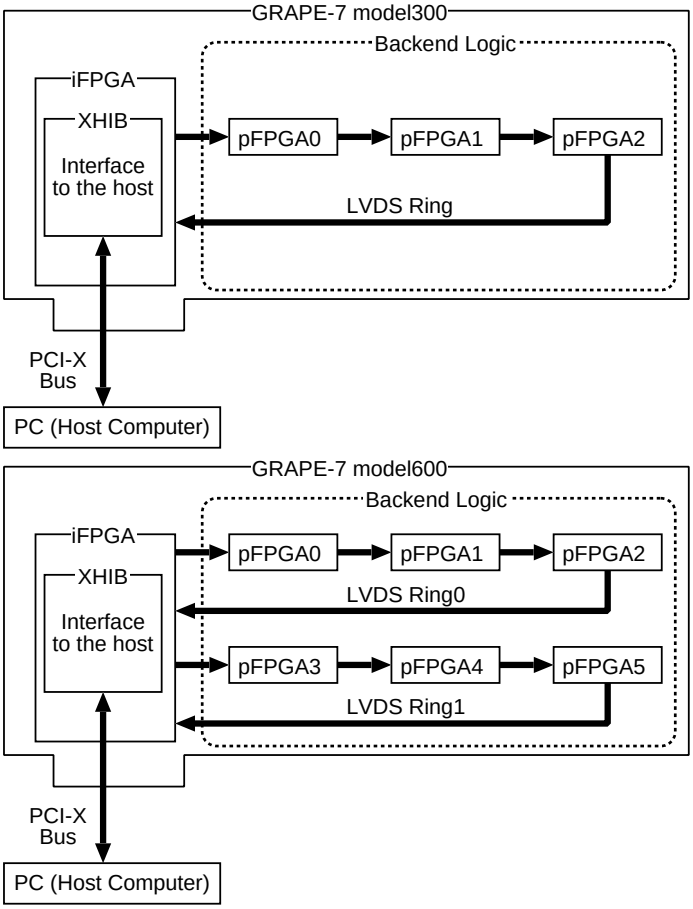

図 3: GRAPE-7 model 300 および model 600 の内部構成。

1.2 pFPGA の 内部回路

pFPGA に書き込むための内部回路として、現在のところ当社は **G5** という回路を配布しています。model 100 にはあらかじめ G5 が内部回路として書き込まれています。model 300 および model 600 については、G5 の回路データが .ttf ファイルとしてソフトウェアパッケージに含まれています。カードの電源を入れ直すたびに、ユーザはこれらのファイルを用いて pFPGA に内部回路を書き込む必要があります。

1.2.1 G5

内部回路 G5 は、従来の GRAPE と同様に、重力多体シミュレーションを加速することを目的として設計されました。G5 はパイプライン回路を用いて粒子からの重力を計算します (図 4)。それ以外の計算、例えば粒子軌道の時間積分などは、ホスト計算機が行います。

G5 のパイプラインは GRAPE-5[1] のそれと基本的には同じものです。GRAPE-7 ソフトウェアパッケージには、G5 を操作するための関数ライブラリ `/usr/g7pkg/lib/libg75.a` が含まれています。使用法は「G5 ユーザガイド」(`/usr/g7pkg/doc/g5user-j.pdf`) で説明されています。このライブラリには GRAPE-5 ライブラリとの後方互換性があります。

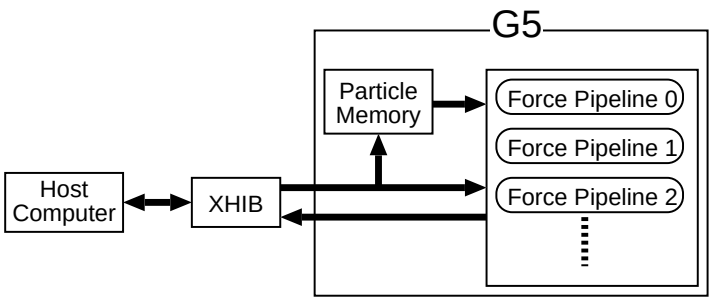

図 4: バックエンド回路 G5 の内部構成

1.3 XHIB

PCI-X Host Interface Bridge (XHIB) は GRAPE-7 とホスト計算機との間のデータ転送を制御する回路です。GRAPE-7 とホスト計算機とは PCI-X 規格の I/O バスで接続されています。

GRAPE-7 ソフトウェアパッケージには XHIB を直接に操作するための関数ライブラリ `/usr/g7pkg/lib/libhib.a` が含まれています。使用法は「HIB Library Functions Reference Manual」(`/usr/g7pkg/doc/hibref.pdf`) で説明されています。しかしながら G5 のユーザはこのライブラリに関する情報を一切必要としません。XHIB ライブラリ関数は G5 ライブラリ関数によって完全に隠蔽されています。

2 G5 の基本的な使用法

2.1 コンパイルとリンクの方法

G5 の制御には G5 ライブラリ関数を用います。ライブラリをユーザ自身のアプリケーションプログラムから使用するには、コード中に `g5util.h` をインクルードします。また G5 ライブラリ (`libg75.a`)、 HIB ライブラリ (`libhib.a`)、C の標準数学ライブラリ (`libm.a`) をリンクします。以下にコンパイルを行う際のコマンドの例を示します。

```
cc -o foo foo.c -L/usr/g7pkg/lib -I/usr/g7pkg/include -lg75 -lhib -lm
```

G5 ライブラリの完全な説明については、第 3 節 「G5 ライブラリ関数リファレンス」を参照してください。従来の GRAPE-5 ライブラリ (`libg5a.a`) や GRAPE-6A ライブラリ (`libg65.a`) を使用していたユーザは、第 3 節を精読する必要はないかもしれません。`g5_` で始まるほとんどの関数は、新しい G5 ライブラリでも従来通り利用できます。ただし新機能や変更のあった機能、削除された機能が第 2.4 節にまとめられていますので、これらについては注意してください。

2.2 環境変数

計算資源の割り当て (複数枚のカードを使う場合のみ): システムに複数枚のカードがインストールされている場合、G5 の標準関数はすべてのカードを使用して計算を行おうとします。この挙動を変えるには、環境変数 `G5_CARDS` に使用したいカードのデバイス ID を列挙します。例えば以下の設定:

```
setenv G5_CARDS "0 2 3"
```

により、G5 の標準関数はデバイス ID 0 番、2 番、3 番 のカードを使用するようになります。この環境変数はひとつのシステムを他人と共同利用する際に役立ちます。

警告メッセージの制御: 環境変数 `G5_WARNLEVEL` によって、警告メッセージの出力を制御できます。この変数は 0, 1, 2, 3 のいずれかの値を取り得ます。大きな値を設定するほど、より詳細なメッセージが出力されます。通常利用の際には 1 か 2 を設定することをお勧めします。3 はデバッグ時に便利かもしれません。0 を設定すると致命的なエラーメッセージ以外の一切のメッセージが出力されなくなります。動作テスト用プログラム (`/usr/g7pkg/scripts/check.csh`) 実行時には、この変数の値を 0 には設定しないでください。0 に設定するとテストが正常に行われません。

データ転送モードの設定: G5 ライブラリはデータをカードへ転送する際に、通常は PCI-X バス上の Programmed I/O Write (PIOW) 転送モードを使用します。PIO とはホスト計算機側が起動するデータ転送モードのことを言います。環境変数 G5_SENDFUNC の値を DMAR に設定すると、代わりに Direct Memory Access Read (DMAR) 転送モードが使用されるようになります。DMA とはカード側が起動するデータ転送モードのことを言います。PIOW と DMAR の性能はホスト計算機に依ります。速い方の転送モードを用いることで、より高い計算性能を得られます。最近のほとんどの PC では、/usr/g7pkg/hibutil/setmtrr.csh によって MTRR を設定してあれば、PIOW の方が高速に動作します。何らかの理由で MTRR がうまく設定できない場合には、DMAR を用いると良いでしょう。MTRR については「GRAPE-7 インストールガイド」の第 3.1 節を参照してください。

2.3 サンプルプログラムの実行

GRAPE-7 ソフトウェアパッケージには以下のサンプルプログラムが含まれています。

```
/usr/g7pkg/direct/direct
/usr/g7pkg/direct/directa
/usr/g7pkg/direct/directmc
/usr/g7pkg/vtc/vtc
/usr/g7pkg/directf77/direct
```

ただしここで、パッケージは /usr/g7pkg にインストールされているものとします。ディレクトリ /usr/g7pkg/direct には direct、directa、directmc が置かれています。direct は直接法の最も単純なプログラム例であり、C 言語で書かれています。directa は direct に簡単なアニメーション機能をつけたものです。directmc は direct とほぼ同じですが、「標準関数」の代わりに「基本関数」を使用しています。基本関数を用いると、ユーザはどのカードを使って計算を行うかを明示的に指定できます。「標準関数」と「基本関数」の詳細については第 3 節を参照してください。 /usr/g7pkg/directf77/direct は /usr/g7pkg/direct/direct の Fortran 版です。/usr/g7pkg/vtc/vtc は Barnes-Hut ツリーコード [3] の C 言語による実装です。使用法は /usr/g7pkg/vtc/00README を参照してください。

以降ではディレクトリ /usr/g7pkg/direct/ 内のプログラムについて、もう少し詳しく説明します。direct と directa はコマンドラインから 2 つの引数をとります。一つ目は入力ファイル名、二つ目は出力ファイル名です。入力ファイルは計算開始時の粒子分布を与えます。出力ファイルには計算終了時の粒子分布が保存されます。両者ともフォーマットは NEMO[2] の 'stoa' 形式です。これは NEMO スナップショットファイルを ASCII フォーマットに変換するコマンド stoa の吐くファイル形式です。このフォーマットを以下に示します。

```
NOBJ                    粒子数 [int]
```

```
NDIM                    空間次元 [int] (常に 3 を指定すること)
TIME                    シミュレーション内の時刻 [double]
MASS(i)                 粒子の質量。i = 1...NOBJ [double]
.....
X(i) Y(i) Z(i)          粒子の位置。i = 1...NOBJ [double]
.....
U(i) V(i) W(i)          粒子の速度。i = 1...NOBJ [double]
.....
```

/usr/g7pkg/direct/pl2k は粒子数 2048 の入力ファイルの一例です。

プログラム directmc は コマンドラインから 3 つの引数をとります。はじめの 2 つは direct の引数と同じです。3 つ目には計算に使用するカードのデバイス ID を与えます。

プログラムの動作を見るには、ディレクトリ /usr/g7pkg/direct へ移動し、コマンド ./direct pl2k outfile を実行します。以下の出力が得られるはずです。

```
nj: 2048
use g5_cards[0]
GRAPE-7 model100  g5_nchip:1 g5_npipes:20 g5_jmemsize:4096 g5_eps2format:floating-point
Warning: g5_get_forceMC() does not calculate potential.
The value returned is just a dummy.
ke: 0.246207
step: 10 time: 1.000000e-01
e: 2.462658e-01 de: 5.902496e-05
ke: 2.462658e-01 pe: 0.000000e+00
ke/pe: inf
.................

step: 90 time: 9.000000e-01
e: 2.554286e-01 de: 9.221770e-03
ke: 2.554286e-01 pe: 0.000000e+00
ke/pe: inf
```

ただし model 100 以外のカードを使っている場合や複数のカードを使っている場合には、出力の第 2 行目と 3 行目は上記の出力とは異なるかもしれません。計算終了時の粒子分布は outfile に保存されます。

ソースコード direct.c を見てみます。このファイル内には calc_gravity() という関数があります:

```
void
calc_gravity(double *mj, double (*xj)[3], double (*vj)[3],
      double eps, double (*a)[3], double *p, int n)
```

```
{
        g5_set_xmj(0, n, xj, mj);
        g5_set_eps_to_all(eps);
        g5_set_n(n);
        g5_calculate_force_on_x(xj, a, p, n);
}
```

この関数は位置 xj に置かれた粒子間の力を計算します。g5_set_xmj() は粒子の質量と位置を G5 のメモリユニットへ保存します (図 4 参照)。g5_set_eps_to_all() はソフトニングパラメタを設定します。g5_set_n() は全粒子数を設定します。g5_calculate_force_on_x() は G5 パイプラインを稼働します。この関数は粒子の位置 xj をパイプラインレジスタへ設定し、パイプラインを稼働します。すると、メモリユニットに保存されている粒子からの力が計算され、順次積算されます。積算された結果はホスト計算機上の配列 a へ送り返されます。

関数 calc_gravity() のすぐ下に、もうひとつ別の関数 calc_gravity2() があります。この関数は calc_gravity() よりも若干複雑ですが、より応用が効きます。ファイル directmc.c にはさらに別の関数 calc_gravity3() があります。これは「基本関数」の使用例です。デバイス ID (1 番目の引数 id) によって、関数 calc_gravity3() がどのカードを使って力の計算を行うべきかを指定できます。

2.4 GRAPE-5/GRAPE-6A との違い

従来の GRAPE-5 ライブラリ (libg5a.a) や GRAPE-6A ライブラリ (libg65.a) にあった関数のほとんどは、新しい G5 ライブラリでも使用できます。ただし変更があった関数や削除された関数、追加された機能などもあります。従来のライブラリを使用していたユーザは以下の変更点に注意してください。

- ライブラリ名が変更されました。従来の libg5.a と libphibdma.a の代わりに libg75.a と libhib.a を使用してください。
- ヘッダファイル名が g5util.h に変更されました。従来の gp5util.h も g5util.h へのシンボリックリンクとして用意されていますが、g5util.h を使用することをお勧めします。
- g5_set_xj() と g5_set_mj() は削除されました。代わりに g5_set_jp() を使用してください。
- G5 は重力のみを計算します。重力ポテンシャルは計算できません。
- 重力カットオフの関数形は P^3M 法で使用されるものに固定されており、プログラマブルではありません。g5_set_cutoff_table() は後方互換性のためだけに存在し、呼び出しても実際には何も行いません。

- G5 の粒子メモリユニットは GRAPE-5 のものに比べて小さいため、GRAPE-5 用のコードに 'j- ループ' を付け足す必要が生じるかも知れません。これによってメモリサイズを超える数の粒子を扱えるようになります。メモリサイズは `g5_get_jmemsize()` によって得られます。得られる値は model 100、300、600 でそれぞれ 4k、6k、12k ですが、これらの値は変更される可能性があるので、値に依存したコードを書くことは勧められません。'j- ループ' を付け足したコードの例は、第 4.3 節にあります。

- G5 の仮想パイプラインの本数は GRAPE-5 のそれよりも大きくなっています。この本数は `g5_get_number_of_pipelines()` によって得られます。得られる値は現在のところ 256 ですが、この値は変更される可能性があるので、値に依存したコードを書くことは勧められません。

- G5 の実パイプラインの本数は `g5_get_number_of_real_pipelines()` によって得られます。この本数はシステムのピーク性能を求める際に便利かも知れません (`/usr/g7pkg/direct/directtest.c` に実例があります)。' 仮想' パイプラインと' 実' パイプラインの違いについては第 3 節を参照してください。

- いくつかの環境変数が新たに追加され、いくつかは削除されました。G5 ライブラリで利用できる環境変数は、第 2.2 節に列挙されています。

2.5 マルチスレッド環境下での使用法

G5 をマルチスレッドプロセスから使用する際の注意点を以下に示します。

- 「標準関数」(第 3.1.1 節) は使わないこと。「基本関数」のみを使用すること。

- 1 枚のカードを複数のスレッドから使用しないこと。つまり、「基本関数」を呼ぶ際に、ある特定のデバイス ID を複数のスレッドから使用しないこと。

3 G5 ライブラリ関数リファレンス

3.1 書式

3.1.1 標準関数 (C 言語)

以下の関数は G5 を制御する際に用いる高レベルのプログラミングインタフェースです。ここで説明する関数は計算に使用するカードの枚数をユーザから隠蔽します。これによってユーザのプログラムが単純になります。ここで説明する関数を用いて G5 を制御している限り、ユーザは複数のカードを一つの実体として扱えます。ユーザはホスト計算機に何枚のカードが挿さっているかを気にすることなくプログラムを記述できます。

```
void g5_open(void);
void g5_close(void);
double g5_get_pcibus_freq(void);
void g5_set_range(double xmin, double xmax, double mmin);
void g5_get_range(double *xmin, double *xmax, double *mmin);
void g5_set_xmj(int adr, int nj, double (*xj)[3], double *mj);
void g5_set_jp(int adr, int nj, double *m, double (*x)[3]);
void g5_calculate_force_on_x(double (*x)[3], double (*a)[3], double *p, int ni);
void g5_set_xi(int ni, double (*x)[3]);
void g5_run(void);
void g5_set_n(int nj);
void g5_set_eps(int ni, double *eps);
void g5_set_eps2(int ni, double *eps2);
void g5_set_eps_to_all(double eps);
void g5_set_eps2_to_all(double eps2);
void g5_get_force(int ni, double (*a)[3], double *pot);
void g5_set_cards(int *c);
void g5_get_cards(int *c);
int g5_get_number_of_cards(void);
int g5_get_number_of_pipelines(void);
int g5_get_number_of_real_pipelines(void);
int g5_get_jmemsize(void);
void g5_set_eta(double eta);
void g5_set_cutoff_table(double (*ffunc)(double), double fcut, double fcor,
                         double (*pfunc)(double), double pcut, double pcor);
```

3.1.2 基本関数 (C 言語)

以下の関数は G5 を制御する際に用いる低レベルのプログラミングインタフェースであり、これらを用いると各 GRAPE-7 カード内に書き込まれた G5 回路を個別に操作できます。各関数はデバイス ID (devid) を引数にとります。各 GRAPE-7 カードのデバイス ID は /usr/g7pkg/hibutil/lsgrape ユーティリティを用いて取得できます。このユーティリティの使用法は「GRAPE-7 インストールガイド」を参照してください。

各関数の devid 以外の引数の意味は、対応する標準関数 (前節参照) の引数と同じです。

```
void g5_openMC(int devid);
void g5_closeMC(int devid);
double g5_get_pcibus_freqMC(int devid);
void g5_set_rangeMC(int devid, double xmin, double xmax, double mmin);
void g5_get_rangeMC(int devid, double *xmin, double *xmax, double *mmin);
void g5_set_xmjMC(int devid, int adr, int nj, double (*xj)[3], double *mj);
void g5_set_jpMC(int devid, int adr, int nj, double *m, double (*x)[3]);
void g5_set_xiMC(int devid, int ni, double (*x)[3]);
void g5_runMC(int devid);
void g5_set_nMC(int devid, int n);
void g5_set_epsMC(int devid, int ni, double *eps);
void g5_set_eps2MC(int devid, int ni, double *eps2);
void g5_set_eps_to_allMC(int devid, double eps);
void g5_set_eps2_to_allMC(int devid, double eps2);
void g5_get_forceMC(int devid, int ni, double (*a)[3], double *pot);
int g5_get_number_of_pipelinesMC(int devid);
int g5_get_number_of_real_pipelinesMC(int devid);
int g5_get_jmemsizeMC(int devid);
void g5_set_etaMC(int devid, double eta);
void g5_set_cutoff_tableMC(int devid,
                           double (*ffunc)(double), double fcut, double fcor,
                           double (*pfunc)(double), double pcut, double pcor);
```

3.1.3 標準関数 (Fortran 言語)

以下のサブルーチンと関数は Fortran 言語から G5 を制御する際に用いるプログラミングインタフェースです。C 言語版と同じ機能を有します (ダミー関数の g5_set_cutoff_table() は除く)。

```
subroutine g5_open

subroutine g5_close

function g5_get_pcibus_freq
real*8 g5_get_pcibus_freq

subroutine g5_set_range(xmin, xmax, mmin)
real*8 xmin
real*8 xmax
real*8 mmin

subroutine g5_get_range(xmin, xmax, mmin)
real*8 xmin
real*8 xmax
real*8 mmin

subroutine g5_set_xmj(adr, nj, xj, mj)
integer adr
integer nj
real*8 xj(3, *)
real*8 mj(*)

subroutine g5_set_jp(adr, nj, mj, xj)
integer adr
integer nj
real*8 mj(*)
real*8 xj(3, *)

subroutine g5_calculate_force_on_x(x, a, p, ni)
real*8 x(3, *)
real*8 a(3, *)
real*8 p(*)
integer ni
```

```
subroutine g5_set_xi(ni, xi)
integer ni
real*8 xi(3, *)

subroutine g5_run

subroutine g5_set_n(n)
integer n

subroutine g5_set_eps(ni, eps)
integer ni
real*8 eps(*)

subroutine g5_set_eps_to_all(eps)
real*8 eps

subroutine g5_set_eps2(ni, eps2)
integer ni
real*8 eps2(*)

subroutine g5_set_eps2_to_all(eps2)
real*8 eps2

subroutine g5_get_force(ni, a, p)
integer ni
real*8 a(3, *)
real*8 p(*)

subroutine g5_set_cards(c)
real*8 c(*)

subroutine g5_get_cards(c)
real*8 c(*)

function g5_get_number_of_cards
integer g5_get_number_of_cards

function g5_get_number_of_pipelines
integer g5_get_number_of_pipelines
```

```
function g5_get_number_of_real_pipelines
integer g5_get_number_of_real_pipelines

function g5_get_jmemsize
integer g5_get_jmemsize

subroutine g5_set_eta(eta)
real*8 eta
```

3.1.4 基本関数 (Fortran 言語)

以下のサブルーチンと関数は Fortran 言語から G5 を制御する際に用いるプログラミングインタフェースです。C 言語版と同じ機能を有します (ダミー関数の g5_set_cutoff_tableMC() は除く)。

```
subroutine g5_openMC(devid)
integer devid

subroutine g5_closeMC(devid)
integer devid

function g5_get_pcibus_freqMC(devid)
integer devid
real*8 g5_get_pcibus_freq

subroutine g5_set_rangeMC(devid, xmin, xmax, mmin)
integer devid
real*8 xmin
real*8 xmax
real*8 mmin

subroutine g5_get_rangeMC(devid, xmin, xmax, mmin)
integer devid
real*8 xmin
real*8 xmax
real*8 mmin

subroutine g5_set_xmjMC(devid, adr, nj, xj, mj)
integer devid
integer adr
integer nj
real*8 xj(3, *)
real*8 mj(*)

subroutine g5_set_jpMC(devid, adr, nj, mj, xj)
integer devid
integer adr
integer nj
real*8 mj(*)
```

```
real*8 xj(3, *)

subroutine g5_set_xiMC(devid, ni, xi)
integer devid
integer ni
real*8 xi(3, *)

subroutine g5_runMC(devid)
integer devid

subroutine g5_set_nMC(devid, n)
integer devid
integer n

subroutine g5_set_epsMC(devid, ni, eps)
integer devid
integer ni
real*8 eps(*)

subroutine g5_set_eps_to_allMC(devid, eps)
integer devid
real*8 eps

subroutine g5_set_eps2MC(devid, ni, eps2)
integer devid
integer ni
real*8 eps2(*)

subroutine g5_set_eps2_to_allMC(devid, eps2)
integer devid
real*8 eps2

subroutine g5_get_forceMC(devid, ni, a, p)
integer devid
integer ni
real*8 a(3, *)
real*8 p(*)

function g5_get_number_of_pipelinesMC(devid)
integer devid
```

```
integer g5_get_number_of_pipelines

function g5_get_number_of_real_pipelinesMC(devid)
integer devid
integer g5_get_number_of_real_pipelines

function g5_get_jmemsizeMC(devid)
integer devid
integer g5_get_jmemsize

subroutine g5_set_etaMC(devid, eta)
integer devid
real*8 eta
```

3.2 説明

3.2.1 標準関数 (C 言語)

void g5_open(void) は GRAPE-7 の利用権限を取得する。g5_set_cards() を除くすべてのライブラリ関数は、この関数を呼んだ後にしか使用できない。g5_open(void) が他のプログラムから既に呼ばれていた場合には、Someone is using hib[n]. Sleep... というメッセージを出力し、他のプログラムが g5_close() によって利用権限を破棄するまで待ちつづける。

g5_open() はホスト計算機に挿さっているすべての GRAPE-7 カードを取得しようとする。この挙動を変えるには、環境変数 G5_CARDS に取得したいカードのデバイス ID を列挙する。例えば以下の設定:

```
setenv G5_CARDS "0 2 3"
```

により、G5 の標準関数はデバイス ID 0 番、2 番、3 番 のカードを取得するようになる。取得するカードを指定するためには、環境変数による方法の他に、関数 g5_set_cards() を用いる方法もある。

void g5_close(void) GRAPE-7 の利用権限を破棄する。一つの GRAPE-7 システムを他のユーザと共同で利用する場合には、この関数を一定時間ごと (例えば 1 分ごと) に呼ぶべきである。こうすることによって、自分以外のユーザのプログラムが GRAPE-7 を取得できるようになる。なお、いったん GRAPE-7 の利用権限を破棄したら、再度利用するには g5_open() を呼び直さなくてはならない。

double g5_get_pcibus_freq(void) は PCI-X バスの動作周波数を MHz 単位で返す。返り値は 133.0、100.0、66.0 のいずれかである。この関数は、システムのピーク性能をプログラムの実行時に動的に求める際に便利かも知れない (/usr/g7pkg/direct/directtest.c に実例がある)。

void g5_set_range(double *xmin*, **double** *xmax*, **double** *mmin*) は空間スケールと質量スケールを指定する。

引数 *xmin* と *xmax* は座標系の下限と上限を決定する。すべての粒子の位置座標の各成分は (*xmin/2*, *xmax/2*) の範囲に収まっていなくてはならない。座標系の解像度、つまり表現できる長さの最小値は $\Delta x = (xmax - xmin)/2^{32}$ である。これは G5 パイプラインが位置座標の表現に 32 ビット固定小数フォーマットを用いていることに起因する。例えば $xmin = -64$、$xmax = 64$ と設定すると、解像度は $\Delta x = (64 - (-64))/2^{32} = 1/2^{25} \approx 3 \times 10^{-8}$ となる。この時すべての粒子は辺の長さ 64 で原点を中心とした立方体内に収まっていなくてはならない。そうでない場合には正しい計算結果が得られない。

周期境界下で計算を行う場合 (例えば P^3M 法において Particle-Mesh 相互作用の計算 [4] を行う場合) には、単位セルを表現するように引数 *xmin* と *xmax* を設定する。例え

ば辺の長さが 128 で原点を中心とした立方体形状の単位セルを表すには、$xmin = -64$、$xmax = 64$ と設定する。このように設定すると、力の計算には最近傍の粒子レプリカが自動的に用いられる。カットオフ関数のスケール長 η が適切に設定されていれば、他のレプリカからの力は無視される (*cf.* `g5_set_eta()`)。

引数 *mmin* は粒子の質量の下限を決定する。関数 `g5_set_range()` は、質量 *mmin* の粒子からの力が、たとえその粒子との距離が最大値 $\sqrt{3}(xmax/2 - xmin/2)$ をとった場合でもアンダーフローしないように質量をスケーリングする。*mmin* よりも小さな質量を使用した場合には力が非常に小さくなり、下位ビットの一部が失われてしまう可能性がある。質量の解像度は *mmin* である。つまり、[*mmin*, 2×*mmin*) の範囲内のすべての質量は *mmin* と見なされる。表現できる質量の最大値は $511 \times 2^{31} \times mmin \approx 10^{12} \times mmin$ である。しかしシミュレーション中は質量が以下の関係を常に満たさねばならない。そうしないと力がオーバーフローしてしまう可能性がある。

$$\frac{m}{mmin} \leq \left(\frac{r}{rmin}\right)^2 \tag{1}$$

ここで m は粒子の質量、r は粒子とその粒子からの力を評価する位置との (ソフトニング込みの) 距離であり、また $\mathrm{rmin} \approx 10^{-7} \times (xmax - xmin)$ である。この制限は G5 パイプラインが力の表現に 57-ビット固定小数フォーマットを用いていることに起因する。

例えばすべての粒子が同じ質量を持つ系のシミュレーションの場合には、単に粒子の質量を *mmin* に設定すればよい。ソフトニングは式 (1) が成り立つように *rmin* よりも大きく設定する。各粒子の質量が異なる系のシミュレーションの場合には、各粒子の質量を表現できる程度に充分小さな *mmin* を設定する。i 番目の粒子のソフトニング ϵ_i は、$\epsilon_i \geq rmin\sqrt{m_i/mmin}$ を満たすように充分大きく設定する。ここで m_i は i 番目の粒子の質量である。

void g5_get_range(double **xmin*, **double** **xmax*, **double** **mmin*) は空間スケールと質量スケールを返す。

void g5_set_jp(int *adr*, **int** *nj*, **double ***mj, **double (***xj**)[3])** はメモリユニットへ粒子の質量 *mj[0]...mj[nj−1]* と位置 *xj[0]...xj[nj−1]* を書き込む。*nj* 個の粒子の質量と位置が、メモリユニット内の *adr* 番目から *(adr+nj−1)* 番目の粒子情報として保存される。*adr + nj* はメモリの大きさを超えてはならない。メモリの大きさは関数 `g5_get_jmemsize()` によって得られる。

void g5_set_xmj(int *adr*, **int** *nj*, **double (***xj**)[3], double ***mj**)** は `g5_set_jp()` と同じように動作するが、3 番目と 4 番目の引数の順序が入れ替わっている。この関数は後方互換性のためだけに存在する。

void g5_set_xi(int *ni*, **double (***xi**)[3])** は位置 *xi[0]...xi[ni−1]* をパイプラインユニットに設定する。関数 `g5_run()` によって計算を開始すると、パイプラインはこれらの位置

における力を評価する。*ni* は関数 g5_get_number_of_pipelines() の返す値を超えてはならない。

void g5_set_n(int *n***)** はメモリユニット内に保存されている粒子の数 *n* を設定する。これら *n* 個の粒子からの力は、関数 g5_run() を呼ぶと、関数 g5_set_xi() によって設定された位置において評価され、積算される。*n* はメモリの大きさを超えてはならない。メモリの大きさは関数 g5_get_jmemsize() によって得られる。

void g5_run(void) は力の計算を開始する。メモリユニット内に保存されている粒子は、関数 g5_set_xi() によって設定された位置において評価され、積算される。

void g5_set_eps(int *ni*, **double ****eps***) は *ni* 個のパイプラインユニットのソフトニングパラメタ *eps[0]...eps[ni−1]* を設定する。*ni* は関数 g5_get_number_of_pipelines() の返す値を超えてはならない。

void g5_set_eps2(int *ni*, **double ****eps2***) は g5_set_eps() と同じように動作するが、2 番目の引数としてソフトニングパラメタの 2 乗をとる。

void g5_set_eps_to_all(double *eps***)** すべてのパイプラインユニットへ同一のソフトニングパラメタ *eps* を設定する。

void g5_set_eps2_to_all(double *eps2***)** は g5_set_eps_to_all() と同じように動作するが、引数としてソフトニングパラメタの 2 乗をとる。

void g5_get_force(int *ni*, **double (****a***)[3], double ****p***)** は力の計算が終了するまで待ち、*ni* 個の力をパイプラインユニットから回収する。回収した力とポテンシャルはそれぞれ *a[0]...a[ni−1]* と *p[0]...p[ni−1]* とに返される。*ni* は関数 g5_get_number_of_pipelines() の返す値を超えてはならない。通常の使用方法では、この関数と対応する g5_set_xi() とに同じ *ni* を与える。

G5 は現在のところ重力ポテンシャルを計算しないことに注意。*p* **に返されるポテンシャルの値には意味が無い** (実際には常に 0 が返される)。

void g5_calculate_force_on_x(double (**x***)[3], double (****a***)[3], double ****p***, int** *ni***)** はユーザのコード中によく現れる、典型的な重力計算のくりかえし処理を、関数としてひとつにまとめたものである。

この関数は配列 *x* で与えられる *ni* 個の位置において力を計算する。関数内部では基本的な g5_ 関数を呼ぶことによって計算を行う。関数 g5_set_xi() によってパイプラインユニットへ位置座標を設定し、g5_run() によって計算を開始し、終了まで待ち、g5_get_force() によって結果を回収して配列 *a* へ返す。

G5 は現在のところ重力ポテンシャルを計算しないことに注意。p **に返されるポテンシャルの値には意味が無い** (実際には常に 0 が返される)。

ni が関数 g5_get_number_of_pipelines() の返す値を超えた場合、関数 g5_calculate_force_on_x() は自動的に位置座標を複数のグループに分割し、それぞれのグループに対して力の計算を行う。

void g5_set_cards(int *c) 計算に使用するカードを指定する。デバイス ID が i のカードを使用するには、配列 c の i 番目の要素を 1 に設定する。使用しないカードに対応する要素には 0 を設定する。例えば 5 枚のカードがインストールされているシステムで、デバイス ID 0 番、2 番、3 番のカードを使用するには、c の要素を

```
c = {1, 0, 1, 1, 0};
```

と設定する。配列 c の要素数は、ホスト計算機に挿さっているカードの総数分必要である。

void g5_get_cards(int *c) は配列 c に現在使用中のカードを返す。配列要素の意味は関数 g5_set_cards() の配列引数 c のそれと同じである。

int g5_get_number_of_cards(void) は現在使用中のカードの枚数を返す。特に指定がなければホスト計算機に挿さっているカードの枚数を返す。環境変数 G5_CARDS が設定されている場合や、関数 g5_set_cards() が呼ばれた場合には、返り値は自動的に計算される。

int g5_get_number_of_pipelines(void) は仮想パイプラインの本数を返す。返される値は、関数 g5_set_xi()、g5_run()、g5_get_force() の一連の呼び出し 1 回で処理できる粒子数の最大値である。

より正確には、この関数の返す値は G5 バックエンド回路内部の FIFO (以降では FIFO1 と呼ぶ) に保存し得る粒子の最大数である。関数 g5_set_xi() により、粒子が FIFO1 に保存される。関数 g5_run() を呼ぶと、FIFO1 内から $n_{\rm pipe}$ 個の粒子が取り出され、G5 パイプラインレジスタへ設定される。ここで $n_{\rm pipe}$ は FPGA 内に実在する G5 パイプラインの本数である。そしてメモリユニットに保存されている粒子からパイプラインレジスタへ設定されている粒子に対して及ぼされる力が計算され、積算される。積算が終了すると、その結果は別の FIFO (以降では FIFO2 と呼ぶ) に保存される。それが終わると、新たに別の $n_{\rm pipe}$ 個の粒子が FIFO1 から取り出され、先と同様にパイプラインレジスタへ設定される。この操作は FIFO1 が空になるまで繰り返される。その後、FIFO2 の内容はホスト計算機へ送り返され、関数 g5_get_force() によって取り出される。

int g5_get_number_of_real_pipelines(void) は G5 パイプラインの本数を返す。複数枚のカードを使用している場合には、使用中の全てのカードが持つパイプラインの総数を

返す。この関数は、システムのピーク性能をプログラムの実行時に動的に求める際に便利かも知れない (`/usr/g7pkg/direct/directtest.c` に実例がある)。

通常 GRAPE-7 model 100、300、600 には、それぞれ 20、48、96 本のパイプラインが搭載されている。

int g5_get_jmemsize(void) はメモリユニットに保存できる粒子の最大数を返す。複数枚のカードを使用している場合には、使用中の全てのカードを使って保存できる粒子の総数を返す。

通常 GRAPE-7 model 100、300、600 は、それぞれ 4k, 6k, 12k 個の粒子を保存できる。

void g5_set_cutoff_table(double (*ffunc)(double), double fcut, double fcor, double (*pfunc)(double), double pcut, double pcor) は後方互換性のためだけに存在する。呼び出しても実際には何も行わない。

void g5_set_eta(double eta) はカットオフ関数のスケール長 η を設定する。関数形は P^{3}M 法において Particle-Mesh 相互作用の計算 [4] に使用されるものに固定されており、プログラマブルではない。η を設定すると、力にはカットオフ関数 $g_{\rm P3M}(R)$ の値が乗じられる。関数 $g_{\rm P3M}(R)$ は以下の式で表される。

$$g_{\rm P3M}(R) = \begin{cases} 1 - \dfrac{1}{140}(224R^3 - 224R^5 + 70R^6 + 48R^7 - 21R^8) & \text{for} \quad 0 \le R < 1 \\ 1 - \dfrac{1}{140}(12 - 224R^2 + 896R^3 - 840R^4 + 224R^5 + 70R^6 - 48R^7 + 7R^8) & \text{for} \quad 1 \le R < 2 \\ 0 & \text{for} \quad R \ge 2, \end{cases} \tag{2}$$

ここで R は、η でスケーリングした (ソフトニングパラメタ込みの) 粒子間距離である。GRAPE の P^{3}M 法への応用については、[5] を参照のこと。

3.2.2 基本関数 (C 言語)、標準/基本関数 (Fortran 言語)

基本関数の動作は標準関数とほぼ同じです。唯一の違いは、基本関数は各カードを個別に制御できることです。

Fortran 言語版のすべての関数の動作は、C 言語版と同じです。

4 ライブラリ関数使用例

4.1 C 言語による例

以下のコードは N 個の粒子の加速度を直接法によって求める方法を示しています。**このコードでは g5_get_jmemsize() が返す値以下の数の粒子しか扱うことが出来ません。**このような制限の無いコードの例については第 4.3 節を参照してください。

```
#include <stdio.h>
#include "g5util.h"
#define NJMAX (10000)

void main(int argc, char **argv)
{
    int i, nj, step, final_step = 100;
    double eps, size;
    static double mj[NJMAX], xj[NJMAX][3], a[NJMAX][3], p[NJMAX];

    readnbody(&nj, mj, xj, vj, argv[1]); // 粒子の初期分布を設定。
    g5_open();
    size = 10.0;
    g5_set_range(-size/2.0, size/2.0, 1.0);

    for (step = 0; step < final_step; step++) {
        g5_set_jp(0, nj, mj, xj); // 力を及ぼす側の粒子を設定。
        g5_set_eps_to_all(eps);   // ソフトニングパラメタを設定。
        g5_set_n(nj);             // 力を及ぼす側の粒子の数を設定。
        g5_calculate_force_on_x(xj, a, p, nj); // 力を受ける側の粒子を設定。
                                               // G5 がこれらの粒子への力を
                                               // 計算、結果を'a' に返す。
                                               // 'p' は現在未使用。

        integrate(xj, vj, a, dt, nj);  // 粒子の位置を更新。
    }
    g5_close();
    writenbody(nj, mj, xj, vj, argv[2]);
}
```

4.2 Fortran 言語による例

以下のコードは N 個の粒子の加速度を直接法によって求める方法を示しています。**このコードでは** g5_get_jmemsize() **が返す値以下の数の粒子しか扱うことが出来ません。** このような制限の無いコードの例については第 4.3 節を参照してください。

```
#include <g5util.h>
#define REAL real*8
#define NJMAX (10000)

      program direct_summation

      REAL mj(NJMAX), xj(3, NJMAX), vj(3, NJMAX)
      REAL a(3, NJMAX), p(NJMAX)
      REAL xmax, xmin, mmin
      REAL eps, epsinv, dt
      integer nj
      integer step, final_step
      integer i, j, k

C     set initial distribution of particles.
      call readnbody(nj, mj, xj, vj)

      final_step = 100
      eps = 0.02
      dt = 0.01
      xmax = 10.0
      xmin = -10.0
      mmin = mj(1)
      call g5_open()
      call g5_set_range(xmin, xmax, mmin)

      do step=1,final_step
C         send particles which exert forces.
          call g5_set_jp(0, nj, mj, xj)

C         send a softening parameter.
          call g5_set_eps_to_all(eps)

C         set number of particles which exert forces.
```

```
          call g5_set_n(nj)

C         send particles which ‘‘feel’’ the force, then
C         G5 calculate the force on particles, and send
C         them back to 'a'.
C         'p' is not used for now.
          call g5_calculate_force_on_x(xj, a, p, nj)

C         update particle positions.
          call integrate(xj, vj, a, dt, nj)
      enddo

      call g5_close()
      call writenbody(nj, mj, xj, vj)

      end
```

4.3 C 言語によるもうひとつの例

以下のコードは N 個の粒子の加速度を直接法によって求める方法を示しています。第 4.1 節の例とは異なり、このコードが扱える粒子の数は、g5_get_jmemsize() による制限を受けません。粒子はメモリユニットの大きさを超えないグループに分割されます。各グループからの力は別個に計算され、それぞれの計算結果はホスト計算機上で足し 合わされます。

```
#include <stdio.h>
#include "g5util.h"
#define NJMAX (10000)

void main(int argc, char **argv)
{
    int i, j, nj, njtmp, step, final_step = 100, jmemsize;
    double eps, size;
    static double mj[NJMAX], xj[NJMAX][3], a[NJMAX][3], p[NJMAX];
    static double atmp[NJMAX][3], ptmp[NJMAX][3];

    readnbody(&nj, mj, xj, vj, argv[1]);
    g5_open();
    size = 10.0;
    g5_set_range(-size/2.0, size/2.0, 1.0);
    jmemsize = g5_get_jmemsize();
    for (step = 0; step < final_step; step++) {
        for (i = 0; i < nj; i++) {  // 全nj 個の粒子の力をゼロクリア。
            for (k = 0; k < 3; k++) {
                a[i][k] = 0.0;
            }
        }
        for (j = 0; j < nj; j += jmemsize) { // 力を及ぼす側の粒子を
            njtmp = jmemsize;                // jmemsize 個以下のグループに
            if (j + jmemsize > nj) {         // 分割して処理する。
                njtmp = nj - j;
            }
            g5_set_jp(0, njtmp, mj + j, xj + j); // 力を及ぼす側の粒子のうち
                                                 // njtmp 個をメモリユニット
                                                 // へ送る。
            g5_set_eps_to_all(eps);
            g5_set_n(njtmp);                          // njtmp 個の粒子から
            g5_calculate_force_on_x(xj, atmp, ptmp, nj); // 全nj 個の粒子へ
```

```
                                                    // の力を計算。

            for (i = 0; i < nj; i++) {    // njtmp 個の粒子からの力を積算。
                for (k = 0; k < 3; k++) {
                    a[i][k] += atmp[i][k];
                }
            }
        }
        integrate(xj, vj, a, dt, nj);
    }
    g5_close();
    writenbody(nj, mj, xj, vj, argv[2]);
}
```

参考文献

[1] Kawai A., Fukushige T., Makino J., and Taiji M.,
GRAPE-5: A Special-Purpose Computer for N-Body Simulations,
Publ. Astron. Soc. Japan (2000), Vol. 52, p. 659,
`http://xxx.lanl.gov/abs/astro-ph/9909116`.

[2] Teuben P. J.,
NEMO - A Stellar Dynamics Toolbox,
`http://bima.astro.umd.edu/nemo/`.

[3] Barnes J. E. and Hut P.,
A Hierarchical $O(N \log N)$ Force Calculation Algorithm,
Nature (1986), Vol. 324, p. 446.

[4] Hockney R. W. and Eastwood J. W.,
Computer Simulation Using Particles,
(McGraw-Hill, New York, 1981) ch5.

[5] Yoshikawa K. and Fukushige T.,
PPPM and TreePM Methods on GRAPE Systems for Cosmological N-Body Simulations,
Publ. Astron. Soc. Japan (2005), Vol. 57, p. 849.

謝辞

以下の皆様にはバグの報告や、貴重なコメントを頂きました。ここに感謝の意を表します: 藤田 裕二 (情報通信研究機構)、斎藤 貴之 (国立天文台)、曽田 康秀 (お茶の水女子大学)、井口 修 (お茶の水女子大学)、立川 崇之 (工学院大学)、Peter Englmaier (University of Zurich)。

更新履歴

version	date	description	author(s)
2.0	23-Apr-2007	カットオフ関数に関する記述を追加。	AK
1.3	11-Mar-2007	第 2.4 節を追加。	AK
1.2	03-Mar-2007	謝辞を追加。	AK
		`G5_SENDFUNC` の記述を追加。	
		サンプルコードを追加。	
1.1	19-Feb-2007	`g5_set_range` の記述を訂正。	AK
1.0	13-Feb-2007	初版作成	AK、TF