

# G5PIPE ユーザガイド

## for GRAPE-7 software package version 2.1

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

### 概要

この文書では G5PIPE の詳細について説明します。G5PIPE とは GRAPE-7 アドインカード向けバックエンド回路 (FPGA に書き込んで使用する演算回路) の一種であり、重力相互作用の計算を行います。

## 目次

<b>1</b>	<b>G5PIPE の概要</b>	<b>2</b>
<b>2</b>	<b>G5PIPE の使用法</b>	<b>3</b>
2.1	コンパイルとリンクの方法 . . . . .	3
2.2	環境変数 . . . . .	4
2.3	サンプルプログラムの実行 . . . . .	6
2.4	GRAPE-5/GRAPE-6A との違い . . . . .	8
2.5	マルチスレッド環境下での使用法 . . . . .	9
<b>3</b>	<b>G5PIPE ライブラリ関数リファレンス</b>	<b>10</b>
3.1	書式 . . . . .	10
3.1.1	標準関数 (C 言語) . . . . .	10
3.1.2	基本関数 (C 言語) . . . . .	12
3.1.3	標準関数 (Fortran 言語) . . . . .	13
3.1.4	基本関数 (Fortran 言語) . . . . .	16
3.2	説明 . . . . .	19
3.2.1	標準関数 (C 言語) . . . . .	19
3.2.2	基本関数 (C 言語)、標準/基本関数 (Fortran 言語) . . . . .	23
<b>4</b>	<b>ライブラリ関数使用例</b>	<b>24</b>
4.1	C 言語による例 . . . . .	24
4.2	Fortran 言語による例 . . . . .	25
4.3	C 言語によるもうひとつの例 . . . . .	27

# 1 G5PIPE の概要

G5PIPE はパイプラインを用いて粒子間の重力を計算します (図 1)。重力以外の計算、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。G5PIPE 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。ただし、高速化のために、重力ポテンシャルの計算機能や近傍粒子リストの作成機能などの一部の機能を省いてあります。

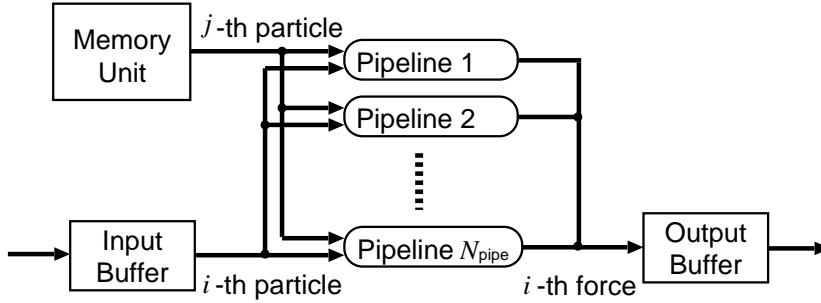


図 1: G5PIPE の内部構成

G5PIPE を用いた重力計算は以下の手順で行われます。なお以下の説明では、重力を受ける側の粒子を  $i$  粒子、重力を及ぼす側の粒子を  $j$  粒子と呼ぶことにします。

- 手順 (1) ホスト計算機が G5PIPE のメモリユニットへ  $j$  粒子を送ります。送る  $j$  粒子の個数はメモリユニットの大きさを超えてはなりません。
- 手順 (2) ホスト計算機が G5PIPE の入力バッファへ  $i$  粒子を送ります。送る  $i$  粒子の個数は入力バッファの大きさを超えてはなりません。
- 手順 (3) ホスト計算機が計算開始コマンドを G5PIPE へ送ります。
- 手順 (4) パイプライン本数  $N_{\text{pipe}}$  ぶんの  $i$  粒子が入力バッファから取り出され、各パイプラインへ 1 粒子ずつ配布されます。
- 手順 (5) パイプラインが稼働し、メモリユニット内の全ての  $j$  粒子から、各パイプラインに配布された  $i$  粒子への重力を計算します。計算中は 1 クロックサイクルにつき 1 個の  $j$  粒子がメモリユニットから全  $N_{\text{pipe}}$  本のパイプラインへ放送されます。各パイプラインは  $j$  粒子が自身の  $i$  粒子へ及ぼす重力を計算し、結果を内部のレジスタに積算します。
- 手順 (6) 各パイプラインは、メモリユニット内のすべての  $j$  粒子から自身の  $i$  粒子への重力を積算し終わると、その結果を出力バッファへ送信します。出力バッファの内容は随時ホスト計算機に回収されます (Model 300, Model 600 の場合には、全ての pFPGA の出力が加算されてからホスト計算機へ回収されます)。
- 手順 (7) 入力バッファが空になるまで、手順 (4)~(6) が繰り返されます。

手順 (8) ここまでで、手順 (1) においてメモリユニットへ送信されたすべての  $j$  粒子から、手順 (2) において入力バッファへ送信されたすべての  $i$  粒子への重力が求められました。ホスト計算機はシミュレーションで扱うすべての  $i$  粒子に対して重力を求め終えるまで、手順 (2)~(7) を繰り返します。

手順 (9) ここまでで、手順 (1) においてメモリユニットへ送信されたすべての  $j$  粒子から、シミュレーションで扱うすべての  $i$  粒子への重力が求められました。ホスト計算機はこれらの重力をメインメモリに保存します。そして次に、ここまでとは別の  $j$  粒子に対して、手順 (1)~(8) を実行し、得られた力を先にメインメモリに保存した力へ加算します。ホスト計算機はシミュレーションで扱うすべての  $j$  粒子からの重力を求め終えるまで、この手順を繰り返します。

G5PIPE 回路の主な仕様を表 1 に示します。表中の値は GRAPE-7 アドインカードの各モデルに G5PIPE を書き込んだ場合の、カード 1 枚当たりのものです。例えばピーク性能は、FPGA チップ 1 個当たりのピーク性能に、カードに搭載されているチップの個数を乗じたものです。同様に、パイプライン本数は、前述の  $N_{\text{pipe}}$  にチップの個数を乗じたものです。なお仕様は変更される可能性があるため、表中の値に依存したコードを書くことは勧められません。

表 1: G5PIPE の主な仕様

	ピーク性能 (Gflops)	パイプライン 本数	動作周波数 (MHz)	メモリユニットに 格納可能な粒子数
Model 100	101	20	133	4095
Model 300	228	60	100	12285
Model 600	456	120	100	24570
Model 800	827	128	170	32764

## 2 G5PIPE の使用法

### 2.1 コンパイルとリンクの方法

G5PIPE の制御には G5PIPE ライブラリを用います。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 G5PIPE 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
```

G5PIPE ライブラリの完全な説明については、第 3 節「G5PIPE 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 G5PIPE 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、G5PIPE の標準関数はすべてのカードを使用して計算を行います。この挙動を変更するには、使用したいカードのデバイス ID を環境変数 G5\_CARDS に列挙します。例えば以下の設定:

```
ssh> setenv G5_CARDS "0 2 3"
ssh> export G5_CARDS="0 2 3"
```

により、G5PIPE 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のユーザと共同で利用する際に役立ちます。

なお Model 800 については、カード 1 枚に搭載された 4 個の iFPGA それぞれに別個のデバイス ID が割り当てられます。そのためそれぞれの iFPGA を独立した GRAPE-7 カードとみなし、別個の計算に使用することも可能です。例えばシステムに 1 枚の Model 800 のみがインストールされている場合に、以下の設定:

```
ssh> setenv G5_CARDS "0 2"
ssh> export G5_CARDS="0 2"
```

を行うと、G5PIPE の標準関数は Model 800 の 0 番および 2 番の iFPGA を使用して計算を行います。使用されていない 1 番および 3 番の iFPGA は、他の計算に使用することが可能です。

より複雑な例として、システムに 2 枚の Model 600 と 1 枚の Model 800 がインストールされている場合を考えます。この場合にはまずコマンド `lsgrape`<sup>1</sup>を用いて、各カードのデバイス ID を調べます。

```
localhost>/usr/g7pkg/scripts/lsgrape
devid grape(model)      backend-logic
```

---

<sup>1</sup>コマンド `lsgrape` の利用法については「GRAPE-7 インストールガイド」(`/usr/g7pkg/doc/g7install-j.pdf`) 第 5 節を参照してください。

```
0  GRAPE-7(model600)  G5
1  GRAPE-7(model800)  G5
2  GRAPE-7(model800)  G5
3  GRAPE-7(model800)  G5
4  GRAPE-7(model800)  G5
5  GRAPE-7(model600)  G5
```

この出力から、2 枚の Model 600 にはそれぞれデバイス ID 0 番および 5 番が割り当てられており、Model 800 の 4 個の iFPGA にはデバイス ID 1 ~ 4 番が割り当てられていることがわかります。このとき、以下の設定:

```
csh> setenv G5_CARDS "3 4 5"
sh> export G5_CARDS="3 4 5"
```

を行うと、G5PIPE の標準関数はデバイス ID 3, 4 番をもつ Model 800 上の 2 個の iFPGA と、デバイス ID 5 をもつ 1 枚の Model 600 を使用して計算を行います。使用されていない Model 800 上の 2 個の iFPGA および 1 枚の Model 600 は、他の計算に使用することが可能です。

**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:** G5PIPE ライブラリはデータをホスト計算機からカードへ転送する際に、転送方式として Programmed I/O Write (PIOW) を使用します。PIO とはホスト計算機が転送を開始するデータ転送方式のことを言います。環境変数 `G5_SENDFUNC` の値を `DMAR` に設定すると、転送方式として PIOW の代わりに Direct Memory Access Read (DMAR) が使用されるようになります:

```
csh> setenv G5_SENDFUNC "DMAR"
sh> export G5_SENDFUNC="DMAR"
```

DMA とはカード側が転送を開始するデータ転送方式のことを言います。PIOW と DMAR の性能はホスト計算機に依って異なります。速い方の転送方式を用いることで、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、コマンド `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。MTRR を設定しないと、多

くの場合に DMAR 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。コマンド `setmtrr.csh` および 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/direct/directttest
/usr/g7pkg/directf77/direct
/usr/g7pkg/vtc/vtc
/usr/g7pkg/direct/directnb
/usr/g7pkg/direct/directnba
```

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`、`directttest`、`directnb`、`directnba` 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。

`directttest` は基本的には `direct` と同じですが、G5PIPE の粒子メモリサイズよりも多くの粒子を扱うための拡張がなされています。この拡張についてのより詳しい説明は第 4.3 節にあります。

`/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/OOREADME` を参照してください。

`directnb`、`directnba` は近傍粒子リスト作成機能を持ったパイプライン回路、G5nbPIPE 向けのプログラムです。使用法は「G5nbPIPE ユーザガイド」(`/usr/g7pkg/doc/g5nbuser-j.pdf`) 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 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 (ただしハードウェアの構成や G5PIPE およびソフトウェアのバージョンによっては、出力される数値は完全には一致しないかも知れません)。

```

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

```

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_jp(0, n, mj, xj);
    g5_set_eps2_to_all(eps*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_jp()` は粒子の質量と位置を G5PIPE のメモリユニットへ格納します (第 1 節 手順 (1))。`g5_set_eps2_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()` は第 1 節の手順 (2)~(7) を繰り返し、 $n$  個すべての粒子に対するすべての粒子からの重力を求めます。

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。この関数の動作は `calc_gravity()` と同じですが、`calc_gravity()` の内部で行われている第 1 節 手順 (2)~(7) の繰り返し処理を、明示的に記述しています。そのため `calc_gravity()` よりも複雑に見えますが、より柔軟なコーディングが可能です。例えば必要に応じて各手順の合間に別の処理を行わせることができます。

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。GRAPE-7 のカードを複数使用している時、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。

ファイル `directtest.c` には関数 `calc_gravity4()` があります。この関数は `calc_gravity2()` を、G5PIPE の粒子メモリサイズよりも多くの粒子を扱えるように改良したものです (第 4.3 節を参照)。

## 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`) に含まれていた関数のほとんどは、新しい G5PIPE ライブラリでも使用できます。ただし仕様の変更され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。



- G5PIPE 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。近傍粒子リスト作成機能については、G5PIPE の機能拡張版パイプライン回路 G5nbPIPE を用いれば利用できます。通常版の G5PIPE 回路では行えません。G5nbPIPE の使用方法については「G5nbPIPE ユーザガイド」(</usr/g7pkg/doc/g5nbuser-j.pdf>) を参照してください。
- 重力カットオフの関数形は P<sup>3</sup>M 法で使用す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。
- G5PIPE 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。得られる値は表 1 に記載されていますが、これらの値は変更される可能性があるため、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。
- G5PIPE の仮想パイプラインの本数 (つまり図 1 中の入力バッファの大きさ) 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ため、値に依存したコードを書くことは勧められません。
- 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。G5PIPE 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

G5PIPE 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 G5PIPE ライブラリ関数リファレンス

G5PIPE ライブラリ関数はパイプライン回路 G5PIPE 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。以下ではそれらの関数の書式と使用方法について説明します。

### 3.1 書式

#### 3.1.1 標準関数 (C 言語)

ここで説明する関数は G5PIPE 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。これらの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 G5PIPE 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意識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);
void    g5_set_range(double xmin, double xmax, double mmin);
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_eps2(int ni, double *eps2);
void    g5_set_eps2_to_all(double eps2);
void    g5_get_force(int ni, double (*a)[3], double *pot);
int     g5_get_number_of_pipelines(void);
int     g5_get_jmемsize(void);
void    g5_set_eta(double eta);
void    g5_set_h_to_all(double h);
void    g5_set_h(int ni, double *h);
int     g5_read_neighbor_list(void);
int     g5_get_neighbor_list(int ip, int *list);
int     g5_get_nbmemsize(void);
int     g5_set_nbmemsize(int size);
```

以下の関数は廃止の予定です。次期バージョンではサポートしない可能性があります。

```
double g5_get_pcibus_freq(void);
```

```
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_eps(int ni, double *eps);
void    g5_set_eps_to_all(double eps);
void    g5_set_cards(int *c);
void    g5_get_cards(int *c);
int     g5_get_number_of_cards(void);
int     g5_get_number_of_real_pipelines(void);
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)

ここで説明する関数は G5PIPE 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す。これらを用いると各 GRAPE-7 カード内に書き込まれた G5PIPE 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);
void    g5_set_rangeMC(int devid, double xmin, double xmax, double mmin);
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_eps2MC(int devid, int ni, double *eps2);
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_jmемsizeMC(int devid);
void    g5_set_etaMC(int devid, double eta);
void    g5_set_h_to_allMC(int devid, double h);
void    g5_set_hMC(int devid, int ni, double *h);
int      g5_read_neighbor_listMC(int devid);
int      g5_get_neighbor_listMC(int devid, int ip, int *list);
int      g5_get_nbmemsizeMC(int devid);
int      g5_set_nbmemsizeMC(int devid, int size);
```

以下の関数は廃止の予定です。次期バージョンではサポートしない可能性があります。

```
double  g5_get_pcibus_freqMC(int devid);
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_epsMC(int devid, int ni, double *eps);
void    g5_set_eps_to_allMC(int devid, double eps);
int      g5_get_number_of_real_pipelinesMC(int devid);
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 G5PIPE 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
```

```
subroutine g5_set_range(xmin, xmax, mmin)
```

```
real*8 xmin
```

```
real*8 xmax
```

```
real*8 mmin
```

```
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_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(*)
```

```
function g5_get_number_of_pipelines
```

```
integer g5_get_number_of_pipelines
```

```
function g5_get_jmemsize
```

```
integer g5_get_jmemsize
```

```
subroutine g5_set_eta(eta)
```

```
real*8 eta
```

以下の関数は廃止の予定です。次期バージョンではサポートしない可能性があります。

```
function g5_get_pcibus_freq
```

```
real*8 g5_get_pcibus_freq
```

```
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_eps(ni, eps)
```

```
integer ni
```

```
real*8 eps(*)
```

```
subroutine g5_set_eps_to_all(eps)
```

```
real*8 eps
```

```
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_real_pipelines
integer g5_get_number_of_real_pipelines
```

### 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 G5PIPE 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
```

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

```
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_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_jmemsizeMC(devid)
integer devid
integer g5_get_jmemsize

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

```

以下の関数は廃止の予定です。次期バージョンではサポートしない可能性があります。

```

function g5_get_pcibus_freqMC(devid)
integer devid
real*8 g5_get_pcibus_freq

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_epsMC(devid, ni, eps)
integer devid
integer ni
real*8 eps(*)

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

function g5_get_number_of_real_pipelinesMC(devid)
integer devid
integer g5_get_number_of_real_pipelines

```

## 3.2 説明

### 3.2.1 標準関数 (C 言語)

**void g5\_open(void)** は G5PIPE 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、メッセージ:

```
Someone is using hib[n]. Sleep...
```

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 カード上の G5PIPE を取得しようとする。環境変数 G5\_CARDS を設定することで、この挙動を変更できる。環境変数 G5\_CARDS の詳細については第 2.2 を参照のこと。

**void g5\_close(void)** G5PIPE 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、他のユーザのプログラムにも G5PIPE を利用する機会が与えられる。なお、いったん G5PIPE の利用権限を破棄したのちに再度 G5PIPE を利用するには g5\_open() を呼び直さなくてはならない。

**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 G5PIPE 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<sup>3</sup>M 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  $\eta$  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ルリングする。 $mmin$  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、 $[mmin, 2 \times mmin)$  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。

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

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  $rmin \approx 10^{-7} \times (xmax - xmin)$  である。この制限は G5PIPE 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  $\epsilon_i$  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\_set\_jp(int adr, int nj, double \*mj, double (\*xj)[3])** はメモリユニットへ  $j$  粒子の質量  $mj[0] \dots mj[nj-1]$  と位置  $xj[0] \dots xj[nj-1]$  を書き込む (cf. 第 1 節 手順 (1))。  $nj$  個の  $j$  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格納される。  $adr + nj$  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\_xi(int ni, double (\*xi)[3])** は  $i$  粒子の位置  $xi[0] \dots xi[ni-1]$  をパイプラインへの入力バッファに設定する (cf. 第 1 節 手順 (2))。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  $i$  粒子の位置において  $j$  粒子からの力を評価し、積算する。  $ni$  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格納されている  $j$  粒子の数  $n$  を設定する。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  $n$  個の  $j$  粒子からの力を、関数 `g5_set_xi()` によって設定された  $i$  粒子の位置において評価し、積算する。  $n$  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。計算中、パイプラインは  $j$  粒子からの力を、関数 `g5_set_xi()` によって設定された  $i$  粒子の位置において評価し、積算する。

**void g5\_set\_eps2(int ni, double \*eps2)** は  $ni$  個の  $i$  粒子に対して、ソフトニングパラメタを設定する。引数  $ni$  は関数 `g5_get_number_of_pipelines()` の返す値を超えてはならない。引数  $eps2[0] \dots eps2[ni-1]$  には設定しようとするソフトニング値の 2 乗を与える。この関数は対応する `g5_set_xi()` よりも前に呼び出しておく必要がある。

**void g5\_set\_eps2\_to\_all(double eps2)** はすべての  $i$  粒子に対して同一のソフトニングパラメタを設定する。引数 *eps2* には設定しようとするソフトニング値の 2 乗を与える。この関数は対応する **g5\_set\_xi()** よりも前に呼び出しておく必要がある。

**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、それに対応する **g5\_set\_xi()** とには、同じ  $ni$  を与える。

G5PIPE 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 (第 1 節 手順 (2)~(7)) 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  $i$  粒子位置において、メモリユニットに格納されているすべての  $j$  粒子からの力を計算し、積算する。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っている。すなわち関数 **g5\_set\_xi()** によって  $i$  粒子の位置座標を設定し、**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。

G5PIPE 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 (つまり第 1 節 手順 (8) の繰り返しを自動的に行う)。

**int g5\_get\_number\_of\_pipelines(void)** は入力バッファ (図 1) のサイズを返す。つまり返される値は、関数 **g5\_set\_xi()**、**g5\_run()**、**g5\_get\_force()** の一連の呼び出し 1 回 (第 1 節 手順 (2)~(7) に対応) で処理できる  $i$  粒子の個数の最大値である。

**int g5\_get\_jmemsize(void)** はメモリユニットに格納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格納できる粒子の総数を返す。GRAPE-7 の各モデルが格納できる粒子の個数については表 1 を参照のこと。

**void g5\_set\_eta(double eta)** はカットオフ関数のスケール長  $\eta$  を設定する。関数形は  $P^3M$  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。 $\eta$  を設定すると、力にはカットオフ関数  $g_{P^3M}(R)$  の値が

乗じられる。関数  $g_{\text{P3M}}(R)$  は以下の式で表される。

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

ここで  $R$  は、 $\eta$  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<sup>3</sup>M 法への応用については、[5] を参照のこと。

**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)。この関数は廃止の予定。次期バージョンではサポートしない可能性がある。

**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。この関数は廃止の予定。次期バージョンではサポートしない可能性がある。

**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。この関数は廃止の予定。次期バージョンではサポートしない可能性がある。

**void g5\_set\_eps(int ni, double \*eps)** は g5\_set\_xi() によって設定した  $ni$  個の  $i$  粒子に対するソフトニングパラメタ  $eps[0] \dots eps[ni-1]$  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。この関数は対応する g5\_set\_xi() よりも前に呼び出しておく必要がある。この関数は廃止の予定。次期バージョンではサポートしない可能性がある。

**void g5\_set\_eps\_to\_all(double eps)** はすべての  $i$  粒子に対して同一のソフトニングパラメタ  $eps$  を設定する。この関数は対応する g5\_set\_xi() よりも前に呼び出しておく必要がある。この関数は廃止の予定。次期バージョンではサポートしない可能性がある。

**void g5\_set\_cards(int \*c)** は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。この関数は廃止の予定。次期バージョンではサポートしない可能性がある。

**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。この関数は廃止の予定。次期バージョンではサポートしない可能性がある。

**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。この関数は廃止の予定。次期バージョンではサポートしない可能性がある。

**int g5\_get\_number\_of\_real\_pipelines(void)** は **G5PIPE** 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)。GRAPE-7 の各モデルが搭載するパイプラインの本数については表 1 を参照のこと。この関数は廃止の予定。次期バージョンではサポートしない可能性がある。

**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。この関数は廃止の予定。次期バージョンではサポートしない可能性がある。

### 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  $n_j$  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()` が返す値以下の個数の  $j$  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_eps2_to_all(eps*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。
                                                // G5PIPE 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  $n_j$  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()` が返す値以下の個数の  $j$  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_eps2_to_all(eps*eps)

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

```

        call g5_set_n(nj)

C        send particles which ‘‘feel’’ the force, then
C        G5PIPE 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  $n_j$  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  $j$  粒子の数は、`g5_get_jmemsize()` による制限を受けません。 $j$  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_eps2_to_all(eps*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: 中里直人 (会津大学)、藤田 裕二 (情報通信研究機構)、斎藤 貴之 (国立天文台)、曾田康秀 (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)
2.1	19-Jan-2007	Model 800 に対応。	AK
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。 <code>G5_SENDFUNC</code> の記述を追加。 サンプルコードを追加。	AK
1.1	19-Feb-2007	<code>g5_set_range</code> の記述を訂正。	AK
1.0	13-Feb-2007	初版作成	AK、TF