G5nbPIPE 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 G5nbPIPE の詳細について説明します。G5nbPIPE 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) の一種です。重力相互作用の計算を行うバックエンド回路 G5PIPE に、近傍粒子リストの作成機能を付加したものです。この文書では G5PIPE に対して G5nbPIPE で新たに追加された機能を中心に説明を行います。従来の G5PIPE の機能については「G5PIPE ユーザガイド」を参照してください。

目 次

1 G5PIPE との相異点

G5nbPIPE は、G5PIPE に近傍粒子リストの作成機能と、リストを保存するためのメモリを付加した回路です (図 1)。G5nbPIPE のピーク演算速度は G5PIPE に比べて低速です。これは機能追加に伴いパイプライン 1 本の実装に必要な回路資源が増加し、FPGA 内に実装されるパイプラインの本数 $N_{\rm pipe}$ が減少したためです。

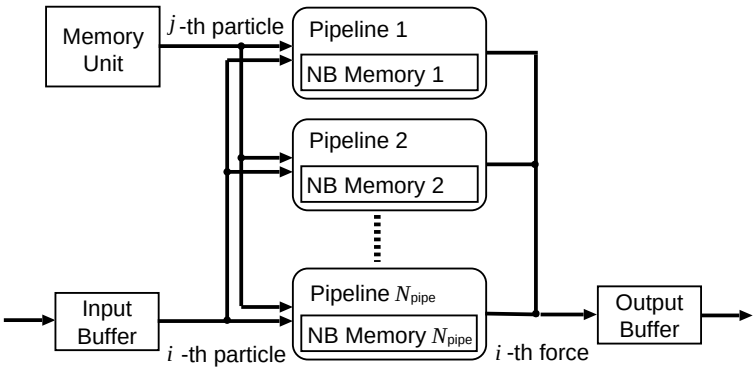

図 1: G5nbPIPE の内部構成

Gnb5PIPE 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 G5nbPIPE を書き込んだ場合の、カード 1 枚当たりのものです。例えば理論性能は、FPGA チップ 1 個当りの理論性能に、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 $N_{\rm pipe}$ 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: G5nbPIPE の主な仕様

	理論性能 (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	近傍粒子リストの最大長
Model 100	71	14	133	4095	24
Model 300	182	48	100	12285	72
Model 600	365	96	100	24570	144
Model 800	731	26	185	32764	96

以降では G5PIPE に対して G5nbPIPE で新たに追加された機能を中心に説明を行います。従来の G5PIPE の機能については「G5PIPE ユーザガイド」を参照してください。

2 G5nbPIPE の使用法

2.1 コンパイルとリンクの方法

G5nbPIPE の制御には G5nbPIPE ライブラリを用います。このライブラリにはすべての G5PIPE ライブラリ関数に加えて、近傍粒子リストの作成に必要な関数が含まれています。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 `g5nbutil.h` をインクルードします。また G5nbPIPE ライブラリ (`libg75nb.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 -lg75nb -lhib -lm
```

2.2 動作テスト用プログラム

G5nbPIPE の動作を確認するために、テスト用のコマンド `./scripts/checknb.csh` を使用できます。コマンド `checknb.csh` の使用法は従来の G5PIPE 向けコマンド `check.csh` と同じです。使用法の詳細については「GRAPE-7 インストールガイド」第 3.2 節を参照してください。

コマンド `checknb.csh` はコマンド `check.csh` と同様に多体計算の時間積分シミュレーションを実行します。シミュレーション終了時の最終的な粒子分布は一時ファイルに保存され、ディレクトリ `/usr/g7pkg/direct/snapshots` 内の対応するファイルと比較されます。これに加えて `checknb.csh` では、シミュレーション終了時のすべての粒子に対するすべての近傍粒子も一時ファイルに保存し、`/usr/g7pkg/direct/snapshots` 内の対応するファイルと比較します。

テストには数分から数十分の時間を要します。テスト中に何らかのエラーメッセージが出力された場合には `support@kfcr.jp` へ連絡して下さい。

2.3 環境変数

設定できる環境変数はすべて G5PIPE のものと同一です。

2.4 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/directtest
/usr/g7pkg/vtc/vtc
/usr/g7pkg/directf77/direct
/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、directtest、directnb、directnba が置かれています。

これらのうち directnb および directnba のみが、近傍粒子リスト作成機能を使用するプログラムです。その他のプログラムの使用法については「G5PIPE ユーザガイド」(/usr/g7pkg/doc/g5user-j.pdf) を参照してください。

directnb は directtest と同様の手順で直接法による重力計算を行い、それと同時に各粒子の近傍にある粒子をリストアップします。directnba は directnb に簡単なアニメーション機能をつけたものです。特定の粒子の近傍粒子のみを赤く、その他の粒子を黄色く表示します。

ファイル directnb.c には関数 calc_gravity5() があります。これは directtest.c 内で記述されている関数 calc_gravity4() に、近傍粒子リスト作成機能を追加したものです。このコードを解読する際には、事前に「G5PIPE ユーザガイド」(/usr/g7pkg/doc/g5user-j.pdf) 5.3 節のサンプルコードと、directtest.c 内の関数 calc_gravity4() を参照しておくことをお薦めします。

2.5 GRAPE-5/GRAPE-6A との違い

G5nbPIPE ライブラリにおいて近傍粒子リスト作成のために用意されている関数は、以下の点を除いて GRAPE-5 ライブラリのそれらと同一です。

- G5nbPIPE の近傍粒子メモリは GRAPE-5 のそれよりも小さくなっています。近傍粒子リストの最大長 (1 個の i 粒子に対して求めることのできる近傍粒子の最大数) は関数 g5_get_nbmemsize() 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。

- 近傍粒子リストがオーバーフローを起こした場合の、関数 g5_get_neighbor_list() の返り値が変更されました。GRAPE-5 ライブラリでは、オーバーフローの有無にかかわらず得られたリスト長が返されました。これに対し G5nbPIPE ライブラリでは、オーバーフローを起こした場合には得られたリスト長に -1 を乗じた値が返されます。オーバーフローを起こさない限りは、両ライブラリ関数の挙動は同一です。

- 近傍粒子リストの最大長を初期値よりも小さな値に設定するための関数 g5_set_nbmemsize() が追加されました。必要な近傍粒子の個数が近傍粒子メモリのサイズよりも小さいこ

とがあらかじめ分かっている場合には、この関数を用いることによって、GRAPE-7とホスト計算機との間の通信量を減らすことができます。

3 G5nbPIPE ライブラリ関数リファレンス (G5PIPE に対する差分のみ)

G5nbPIPE ライブラリ関数はバックエンド回路 G5nbPIPE 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。G5nbPIPE ライブラリには G5PIPE ライブラリに含まれるすべての関数と、近傍粒子リストの作成に必要な関数が含まれています。以下ではこれらの関数のうち、近傍粒子リストの作成に必要な関数のみについて説明します。G5PIPE ライブラリに含まれている関数については「G5PIPE ユーザガイド」(/usr/g7pkg/doc/g5user-j.pdf) を参照してください。

3.1 書式

3.1.1 標準関数 (C 言語)

ここで説明する関数は G5nbPIPE 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 G5nbPIPE 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_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);
```

3.1.2 基本関数 (C 言語)

ここで説明する関数は G5nbPIPE を制御する際に用いる低レベルの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 G5nbPIPE 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 はコマンド lsgrape を用いて取得できます。このコマンドの使用法は「GRAPE-7 インストールガイド」第 5 節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_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_set_nbmemsizeMC(int devid, int size);
int  g5_get_nbmemsizeMC(int devid);
```

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 G5nbPIPE 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。

```
subroutine g5_set_h_to_all(h)
real*8 h

subroutine g5_set_h(ni, *h)
integer ni
real*8 h(*)

function g5_read_neighbor_list()
integer g5_read_neighbor_list

function g5_get_neighbor_list(ip, *list)
integer ip
integer list(*)
integer function g5_get_neighbor_list

function g5_get_nbmemsize()
integer g5_get_nbmemsize

function g5_set_nbmemsize(size)
integer size
integer g5_set_nbmemsize
```

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 G5nbPIPE 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。

```
subroutine g5_set_h_to_allMC(devid, h)
integer devid
real*8 h

subroutine g5_set_hMC(devid, ni, *h)
integer devid
integer ni
real*8 h(*)

function g5_read_neighbor_listMC(devid)
integer devid
integer g5_read_neighbor_listMC

function g5_get_neighbor_listMC(devid, ip, *list)
integer devid
integer ip
integer list(*)
integer g5_get_neighbor_listMC

function g5_get_nbmemsizeMC(devid)
integer devid
integer g5_get_nbmemsizeMC

function g5_set_nbmemsizeMC(devid, size)
integer devid
integer size
integer g5_set_nbmemsizeMC
```

3.2 説明

3.2.1 標準関数 (C 言語)

void g5_set_h_to_all(double h) は近傍粒子の探索半径 h を設定する。設定後に重力計算を行うと、重量計算と同時に、各 i 粒子から半径 h 内に存在する粒子のリスト (近傍粒子リスト) が作成される。作成されたリストは、計算終了後に `g5_read_neighbor_list()` と `g5_get_neighbor_list()` を用いて取得できる。

void g5_set_h(int ni, double *h) は `g5_set_h_to_all()` と同様に近傍粒子の探索半径を設定するが、`g5_set_h_to_all()` とは異なり、ni 個のパイプラインユニットそれぞれに、別個の *h[0]...h[ni−1]* を設定できる。ni は関数 `g5_get_number_of_pipelines()` の返す値を超えてはならない。

int g5_read_neighbor_list(void) は、G5nbPIPE によって作成された近傍粒子リストを、ユーザライブラリ内のバッファへ格納する。この関数は 0 または 1 を返す。返り値 0 は、関数 `g5_set_xi(`ni`,` xi によって G5nbPIPE のパイプラインユニットへ設定した ni 個の i 粒子すべてに対する近傍粒子が、すべて正常に求められたことを意味する。返り値 1 は、いずれかの i 粒子に対する近傍粒子の一部が、G5nbPIPE の近傍粒子リスト用メモリに入りきらずに無視された (オーバーフローした) ことを意味する。オーバーフローを起こした i 粒子を特定するには、関数 `g5_get_neighbor_list()` を用いる。

int g5_get_neighbor_list(int ip, int *list) は関数 `g5_set_xi(`ni`,` xi によって G5nbPIPE のパイプラインユニットへ設定した ni 個の i 粒子のうち、ip 番目の粒子に対する近傍粒子の総数 $n_{\rm ip}$ を返す。ただし、近傍粒子リストの一部が G5nbPIPE の近傍粒子リスト用メモリに入りきらずに無視された (オーバーフローした) 場合には、総数に -1 を乗じた値、$-n_{\rm ip}$ を返す。つまりユーザは、返り値の符号によってリストがオーバーフローしたかどうかを知り、返り値の絶対値からリストの長さを知ることができる。

またこの関数は、近傍粒子の粒子インデクスを配列 *list[0]...list[*$n_{\rm ip}$*−1]* に返す。リストがオーバーフローした場合でも、配列 `list` に返される $n_{\rm ip}$ 個の粒子インデクスは有効である。

配列 `list` に返されるインデクスは、関数 `g5_set_jp(adr, nj, mj, xj)` によってメモリユニットへ設定された nj 個の j 粒子の位置インデクス、すなわち *adr*、*adr+1*、*adr+2*、、、*(adr+nj−1)* である。

一方 ip の値は、パイプラインユニットのインデクスを意味する。つまり関数 `g5_set_xi(ni, xi)` によってパイプラインユニットへ設定された ni 個の i 粒子の ip の値は、順に 0、1、2、、、*(ni−1)* である。

本関数を呼ぶ際には、事前に関数 `g5_read_neighbor_list()` を呼んでおく必要がある。

int g5_get_nbmemsize(void) は近傍粒子リストの最大長 (1 個の i 粒子に対して求めることのできる近傍粒子の最大数) を返す。GRAPE-7 の各モデルに対するこの値については表 1 を参照のこと。

複数の pFPGA チップを持つ Model300 および Model600 では、ひとつの粒子に対する近傍粒子が複数の pFPGA にまたがって保存される。pFPGA 1 個当り最大 24 個の近傍粒子を保存でき、従って pFPGA を 3 個持つ Model300 では最大 72 個 (24×3 個)、6 個持つ Model600 では最大 144 個 (24×6 個) を保存できる。近傍粒子がこれらの最大数に満たなくとも、複数の pFPGA のうちいずれかの近傍粒子メモリが一杯になると、関数 `g5_get_neighbor_list()` はオーバーフローを返すことに注意されたい。

例えば Model600 を用いた計算において、ある粒子の近傍粒子が 25 個しか存在しない場合であっても、それらすべてが 1 番目の pFPGA のメモリユニットに保存されていれば、リストはオーバーフローを起こす。このような事態を起きにくくするため、関数 `g5_set_jp()` はインデクス $6n+k$ を持つた粒子を第 k 番目の pFPGA のメモリユニットへ書き込む (n は非負整数、k は 0〜5 までの整数)。この機構により、互いに近傍にある粒子が連続したインデクスを持つように並べられた配列 xj および mj を関数 `g5_set_jp()` へ渡した場合には、互いに近傍にある粒子が 6 個の pFPGA へほぼ均等に分散される。

int g5_set_nbmemsize(int size) は近傍粒子リストの最大長を、近傍粒子メモリのサイズよりも小さな値に設定する。設定後は *size* で指定した値 (厳密には、*size* の値を FPGA チップ数の整数倍に切り下げた値) までしか粒子リストをホスト計算機に回収しなくなる。これにより GRAPE-7 とホスト計算機との間の通信量を減らすことができる。

近傍粒子メモリのサイズを超える値を *size* に与えた場合には、メモリサイズが設定される。負の値を与えた場合には、0 が設定される。実際に設定された値が、本関数の返り値として返される。

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 近傍粒子リスト 作成の例 (C 言語)

以下のコードは nj 個の粒子の加速度を直接法によって求めると同時に、各粒子から半径 h 内に存在する粒子のリストを作成します。

```
#include <stdio.h>
#include "g5util.h"
#define NJMAX (10000)   // 本サンプルでは簡単のため定数を使用しますが、
#define NPIPES (300)    // 実用コードではこれらの値は G5nbPIPE ライブラリ関数
#define NBMEMSIZE (200) // を用いて動的に取得することを推奨します。

void main(int argc, char **argv)
{
    int i, ii, nj, step, final_step = 100;
    double h, eps, size, dt;
    static double mj[NJMAX], xj[NJMAX][3], vj[NJMAX][3];
    static double a[NJMAX][3], p[NJMAX];
    int npipes, nbof;
    static int nnb[NPIPES], nblist[NPIPES][NBMEMSIZE];

    readnbody(&nj, mj, xj, vj, argv[1]);
    g5_open();
    size = 10.0;
    g5_set_range(-size/2.0, size/2.0, mj[0]);
    npipes = g5_get_number_of_pipelines();
    h = size*0.01;
    for (step = 0; step < final_step; step++) {
        g5_set_jp(0, nj, mj, xj);
        g5_set_eps2_to_all(eps*eps);
        g5_set_h_to_all(h);                 // 近傍粒子探索の半径を設定。
        g5_set_n(nj);
        for (i = 0; i < nj; i += npipes) { // このループ内で、ni 個の
            int ni = npipes;                // 粒子に対する全 nj 個の粒子
            if (i+ni > nj) {                // からの力と、近傍粒子リスト
                ni = nj-i;                  // を求める。
            }
            g5_set_xi(ni, (double (*)[3])xj[i]);
            g5_run();
```

```
            g5_get_force(ni, (double (*)[3])a[i], p+i);
            // 近傍粒子リストをライブラリ内部のバッファへ格納。
            nbof = g5_read_neighbor_list();
            if (nbof == 1) {
                fprintf(stderr, "some NB lists overflown\n");
            }
            for (ii = 0; ii < ni; ii++) {
                // ii 番目の粒子の近傍粒子リストを取得。
                nnb[ii] = g5_get_neighbor_list(ii, nblist[ii]);
                if (nnb[ii] < 0) {
                    fprintf(stderr, "NB list of particle %d overflown\n", ii);
                    nnb[ii] *= -1;
                }
                fprintf(stderr, "NB list length of particle %d is %d\n", ii, nnb[ii]);
            }
            /* --- みつかった nnb[ii] 個の近傍粒子を用いた作業をここで行う --- */
        }
        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`.

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: 斎藤 貴之 (国立天文台)、牧野 淳一郎 (国立天文台)

更新履歴

version	date	description	author(s)
2.1	19-Jan-2008	初版作成	AK