TR－A－0085

MASSCOMP用ベクトル演算装置を用いた
高速フーリエ変換（FFT）プログラム

緒形　昌美

1990. 7. 6

ATR視聴覚機構研究所

〒619-02 京都府相楽郡精華町乾谷　☎07749-5-1411

ATR Auditory and Visual Perception Research Laboratories

Inuidani, Seika-cho, Soraku-gun, Kyoto 619-02 Japan

Telephone: +81-7749-5-1411
Facsimile: +81-7749-5-1408
Telex: 5452-516 ATR J

目次

１．はじめに

FFT（Fast Fourie Transform）は、フーリエ変換をディジタル計算機上で高速に計算するために開発されたアルゴリズムであり、パターンの周波数分析や、フィルタリング処理には必要不可欠なツールになっている。視覚研究においても心理実験における呈示パターンの作成や、モデルのシミュレーションなど、FFTの利用範囲は広い。しかしこのような視覚研究で扱うデータは、通常、２次元画像や３次元（時空間）画像であり、その膨大なデータ量のためにFFTといえども多くの計算時間を必要とする。そこで我々は、現在運動視や立体視の研究に使用しているUNIXマシンMASSCOMP5600の上で、FFTをより高速に計算するために、同計算機に搭載されたベクトル演算装置”ベクトル・アクセレータ”を用いたFFT関数をC言語によって開発した。本報告書では、ベクトル・アクセレータの簡単な使用法と、これを用いて作成したFFTプログラムについて説明する。ただし、フーリエ変換やFFTに関する解説は他の教科書[1]に任せるとし、ここではプログラムの使用法のみを説明する。

なお、混乱を避けるために、本報告書では”ベクトル”と”配列”という２つの言葉を次のように区別して用いる。

ベクトル　・・・　計算対象となるデータ
配列　　・・・　ベクトルを収めるメモリ上の領域

２．ベクトル・アクセレータ

ベクトル・アクセレータ（以後ＶＡと記す）は、MASSCOMPに搭載されたベクトル演算装置であり、用意されたRTL（Run Time Library、Vector Accelerator Programmer's Manual[2]　第13章参照）の中の適当な関数を用いることによって所望の計算を高速に実行することができる。これらの関数はCまたはFORT-RANによって書かれたプログラムから呼び出される。

２．１　ＶＡの構成

ＶＡシステムの概念的構成図を図１に示す。ＶＡは、ベクトル・メモリと呼ばれるローカル・メモリを持っており、数値計算はすべてこのメモリ上のデータに対して行われる。従って、計算を施されるデータすなわちベクトルは、あらかじめホスト・メモリからベクトル・メモリへ転送（ロード）されていなければならない。また、計算終了後のベクトルはベクトル・メモリからホスト・メモリへ転送（ストア）される。ベクトル演算や、ロード、ストアは、RTLの適当な関数を呼び出すことによって実行される。

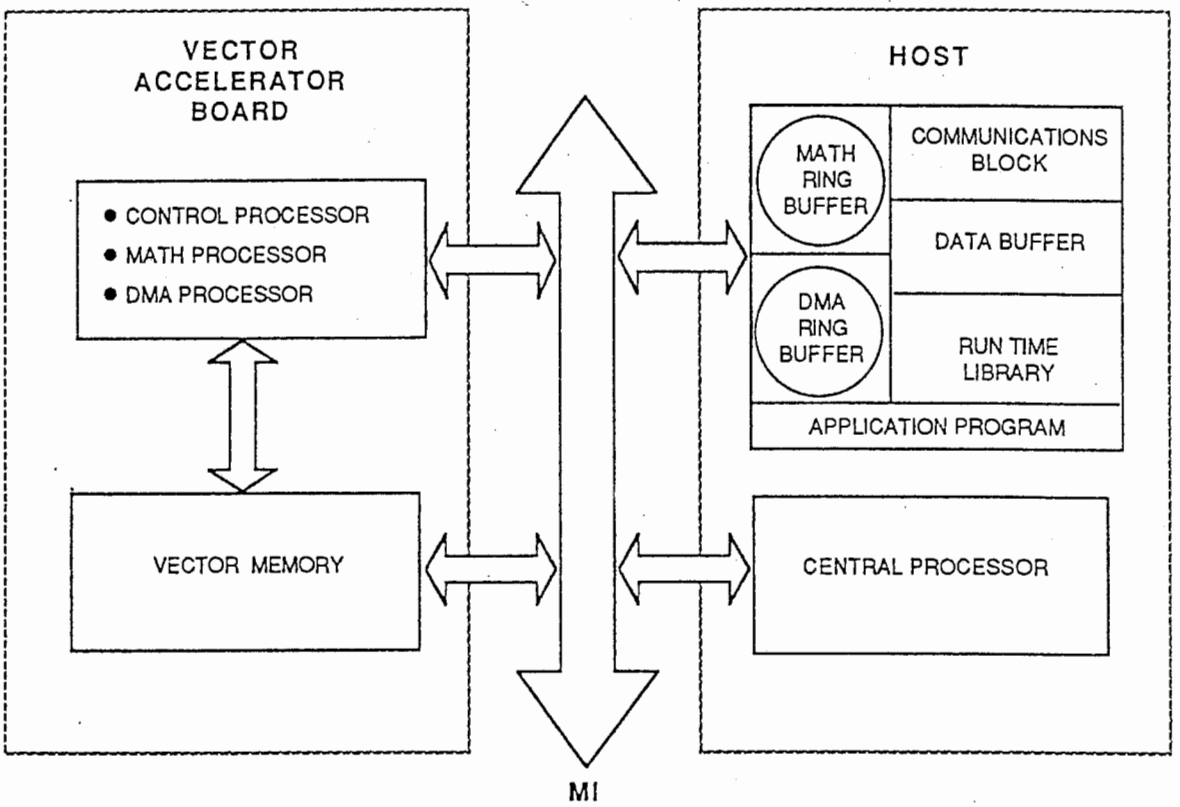

図１ＶＡの構成

表１ホスト・メモリとベクトル・メモリ
における各データ型の大きさ

Sym-bol	Type	Storage		Description
		Host bytes	VA locations	
A	Address	4	1	Vector Memory address
B	Byte	1	1	8-bit integer, sign or zero-ext to 32 bits
CF	Complex	8	2	Single-precision complex floating point
F	Float	4	1	Single-precision floating point
I	Integer	4	1	24-bit integer,sign-ext to 32 bits
W	Word	2	1	16-bit integer,sign or zero-ext to 32 bits

RTLの関数は、その種類、すなわちベクトル演算命令か転送命令（ロード、ストア）かによって、それぞれMATH RING BUFFER、またはDMA RING BUFFERにpacketとして蓄えられる。各RING BUFFERのpacketは、CONTROL PROCESSORによって解釈され、対応するプロセッサ、すなわちMATH PROCESSORまたはDMA PROCESSORに送られ、そこで実行される。

２．２　ベクトル・メモリ上におけるベクトルの表現

ホスト・メモリでは通常、データの単位はバイトであり、例えば整数型ベクトルの１つの要素は４バイトからなっている。一方、ベクトル・メモリ上では32ビットを１つのロケイションと考える。従って、整数型のベクトルの１要素はベクトル・メモリ上では１ロケイションを占める。また複素数型ベクトルの１要素は２ロケイションを占める。ホスト・メモリとベクトル・メモリにおける各データ型のベクトルの１要素の大きさを表１に示す。

２．３　ＶＡによる計算手順

ＶＡを用いて計算を行う場合、おおよそ次のようなステップが必要となる。

(1)ＶＡの確保
(2)使用するベクトル・メモリの領域確保
(3)ベクトルのロード
(4)ベクトル演算
(5)ベクトルのストア
(6)ベクトル・メモリの領域解放
(7)ＶＡの解放

以下、各ステップについて説明する（ただしC言語を用いることを前提とする）。

(1)ＶＡの確保

あるアプリケーション・プログラムからＶＡを用いるためには、まずそのプログラムにＶＡを割り付ける必要がある。このためには、次の関数を用いる。

```
mapinitva(dslock, vacount, fftlock)
      dslock  = 1 ・・・ ホスト・メモリのデータ領域をロックする
                0 ・・・ しない
      vacount     ・・・ 用いるＶＡの数
      fftlock = 1 ・・・ HOST MEMORY BUFFERの割付とロックをする
                0 ・・・ しない
```

引数はすべて整数型である。

データ領域のロックとは、プログラム実行中、常にそのデータ領域を物理的なメモリ上に固定しておくことである（そのメモリ領域ののページ・アウト等を禁止する）。ベクトル・メモリとホスト・メモリの間でデータ転送（ロード、ストア）を行う場合、ホスト側の対象となるデータ領域はロックされている必要があり、通常はこの値を１にする。ただし、この場合、全てのデータ領域がロックされるため、プログラムの中で利用できるデータ領域の大きさは、物理的に存在するメモリの大きさによって制限される。

vacountはそのプログラムの中で用いるＶＡの数である。１つのＶＡは、複数のプログラムから利用することは出来ないが、１つのプログラムの中で複数のＶＡを用いることは可能である。ただし、現在MASSCOMPに搭載されているＶＡは１つだけであるので、この値は常に１とする。

HOST MEMORY BUFFERは、31Kの実数型データ領域で、２次元FFTとベクトル・メモリ間のコピーを行う時に用いる。従って、これらの計算（あるいはコピー）を行う場合には１とする。

(2)使用するベクトル・メモリの領域確保

ベクトル・メモリ上に、計算の対象となるベクトルを格納するための領域を次の関数を用いて確保する。

```
base=mapmalloc(count, alignment, lun)

      base       ・・・  確保された領域の先頭オフセット
      count      ・・・  格納するベクトルの要素数
      alignment  ・・・  確保される領域のメモリ上における配置の仕方
      lun        ・・・  ＶＡのロジカルユニット番号
```

引数はすべて整数型である。この関数は、実数型のベクトルを格納するためのものであるが、実際には取られる大きさが同じであれば表１のどのデータ型のベクトルに対しても用いることができる。例えば、複素数型ベクトルの場合には、count=要素数*2とすればよい。

オフセットとは、ベクトル・メモリ上におけるベクトルの先頭アドレスのことである。

alignmentは、baseが取り得る値を指定するもので、baseの値はalignmentの倍数に限られる。例えばalignment=1024の場合、baseが取り得る値は、0, 1024,2048, 4096, ・・・となる。 通常は、0、1、または2（これらは全て２と見なされる）としておけばよい。

lunはＶＡを識別する番号であり、インクルードファイルaplib.hで定数VA*

(*=0, 1, 2, ・・・. or 7) に割り付けられている。ただし、現在は１つのＶＡしか用いることができないので、ここは常にVA0とすればよい。

(3)データのロード

転送するベクトルの型によっていくつかの関数が用意されている。例えば、

maplodfv(V1addr, V1inc, V0off, V0inc, count)

は実数型のベクトルをロードするための関数である。引数はすべて整数型で、それぞれ、

V1addr・・・　転送されるベクトルが格納されている配列のホスト・メモリ上における先頭アドレス
V1inc ・・・　転送されるベクトルのホスト・メモリにおけるインクリメント
V0off ・・・　ベクトル・メモリ上の転送先のオフセット
V0inc ・・・　ベクトル・メモリにおけるインクリメント
count ・・・　転送ベクトルの要素の数

を表す。ここで、インクリメントとは、ベクトルの隣接する要素の先頭アドレス、またはオフセットの差で、メモリ上に各要素が連続して並んでいる場合（通常の場合）には、１つの要素の大きさに相当する。実数を扱うこの関数では、V1inc=4、V0inc=1となる（表１参照）。また転送先のオフセットはmapmallocによって領域確保を行ったときにその関数値（base）として返された値を用いればよい。

他のロード用の関数としては、

maplodbrfv, maplodbrbv, maplodbv, maolodbrcfv, maplodcfv, maplodbriv, maplodiv, maplodbrwv, maplodbrzvb, maplodzbv, maplodbrzwv, maplodzwv, maplodfs

がある。

(1)に述べたように、ホスト側における転送データの領域は物理的なメモリ上にロックされている必要がある。もし、ロックされていない場合にはエラーとなる。また、ベクトル・メモリとの間でやり取りするデータは、ホスト側ではグローバルに定義された配列に格納されていなければならない。

(4)ベクトル演算

RTLにはベクトル演算を行うための多くの関数が用意されている。例えば

madaddfvv(V1off, V1inc, V2off, V2inc, V0off, V0inc, count)

は、２つの実数型ベクトルの足し算を行うもので、各引数はすべて整数型である。

V1off ・・・ 加算されるベクトルのオフセット
V1inc ・・・ 加算されるベクトルのインクリメント
V2off ・・・ 加算されるベクトルのオフセット
V2inc ・・・ 加算されるベクトルのインクリメント
V0off ・・・ 計算結果のベクトルを格納する領域のオフセット
V0inc ・・・ 計算結果のベクトルを格納するためのインクリメント
count ・・・ ベクトルの要素の数

各関数が必要とする引数はそれぞれ機能によって異なるが、一般的には上の関数のように、計算対象となるベクトルのオフセットとインクリメント、計算結果を格納する領域のオフセットとインクリメント、及びベクトルの要素数が必要とされる。個々の関数については、Vector Accelerator Programmer's Mannualの第13章を参照されたい。

(3)データのストア

ロードの場合と同じく、転送するベクトルの型によっていくつかの関数が用意されている。例えば、

```
mapstrfv(V1addr, V1inc, V0off, V0inc, count)
```

は実数型のベクトルをストアするための関数であり、各引数は整数型で

V1addr・・・ 転送されるベクトルのベクトル・メモリ上のオフセット
V1inc ・・・ 転送されるベクトルのベクトル・メモリにおけるインクリメント
V0off ・・・ ホスト・メモリ上の転送先（配列）の先頭アドレス
V0inc ・・・ ホスト・メモリにおけるインクリメント
count ・・・ 転送されるベクトルの要素の数

を表す。他のストア用の関数としては、

mapstrbrfv, mapstrbrbv, mapstrbv, mapstrbrcfv, mapstrcfv,
mapstrbriv, mapstriv, mapstrbrwv, mapstrwv

がある。

ロードの場合と同じように、ホスト・メモリにおけるデータの転送先は物理的なメモリ上にロックされている必要がある。もし、ロックされていない場合にはエラーとなる。また、ベクトル・メモリとの間でやり取りするデータは、ホスト側ではグローバルに定義された配列に格納されていなければならない。

(6)ベクトル・メモリの領域解放

確保したベクトル・メモリの領域を解放するためには、次の関数を用いる。

```
mapmfree(offset)
```

offset　・・・　解放すべきベクトルのオフセット（整数型）

(7)ＶＡの解放

確保したＶＡを解放するためには次の関数を用いる。

mapfreeva(lun)

lun　・・・　確保していたＶＡのロジカル・ユニット番号（整数型）

(8)その他

MATH及びDMA　PROCESSORは同時に動作し得るが、両プロセッサが順次同じデータを処理対象とする場合には、次のような制約により転送命令とベクトル演算命令の実行タイミングの同期をとる必要がある。

(A)ベクトル演算の実行は、必要なデータのロードが完了後行われなければならない

(B)データのストアはベクトル演算完了後に行われなければならない

(C)ＶＡによって計算されたデータをホスト側で用いるためには、データのストアが完了していなければならない

このような各プロセッサの同期は、次のような関数によって実現できる。

mapsyncdma(packet, lun)

MATH RING BUFFER中の指定したベクトル演算命令が終了するまで次のDMA RING BUFFERの命令実行を保留する。ここで、packetは実行終了を待つベクトル演算を指定するもので、その演算を実行する関数のリターン・バリュとして得られる。lunは、ＶＡのロジカル・ユニット番号である。

mapsyncmath(packet, lun)

DMA RING BUFFER中の指定した転送命令が終了するまで次のMATH RING BUFFERの命令実行を保留する。ここで、packetは実行終了を待つ転送命令を指定するもので、その転送を行う関数のリターン・バリュウとして得られる。

mapwaitdma(lun)

指定したＶＡのDMA RING BUFFER中の全ての命令が実行され終る（またはタイム・アウトが生じる）まで、ホスト側での処理を保留する。

mapwaitmath(lun)

指定したＶＡのMATH RING BUFFER中の全ての命令が実行され終る（またはタイム・アウトが生じる）まで、ホスト側での処理を保留する。

これらは、MATH PROCESSOR、DMA PROCESSOR及びホスト・プロセッサの間の同期をとる最も基本的な関数であるが、他にも機能の異なる関数がいくつか用意されている。

その他、ＶＡを用いるためには、アプリケーション・プログラムで、インクルード・ファイル/usr/include/aplib.hを次のようにインクルードする必要がある。

```
#include <aplib.h>
```

また、コンパイルする場合には、オプションとして

```
-lva
```

を指定しなければならない。

なお、ＶＡについての詳細はVector Accelerator Programmer's Manual[2]を参照されたい。

２．４　サンプル・プログラム

ここではサンプル・プログラムとして、実数型１次元ベクトル中の最大値を求める関数max_1d(in, n)について述べる。プログラム・リストを図２に示す。関数の引数in[1024]は最大値を求めるベクトルが格納されている１次元配列であり、nはベクトルの大きさである。以下リストの行番号に沿って説明していく。なお、用いたRTL関数の後に添えた数字は、Vector Accelerator Programmer's Manual[2]第13章で各関数が説明されているページを表す。

14　：　mapinitva (13-112)
ＶＡの確保。各引数は次の通り。
dslock=1、vacount=1、fftlock=0
ここでは、２次元FFTや、ベクトル・メモリ間のコピーは行わないのでfftlockは0でよい。

17-19:　mapmalloc (13-139)
ベクトル・メモリ領域の確保。ddは、ベクトルを格納する領域のオフセットである。また、mxはRTLの関数によって最大値が返されるロケイションのオフセット、adは最大値をもつ要素のベクトル・メモリ上におけるアドレスが返されるロケイションのオフセット。

22-24:　mapclrfv (13-53)
確保した領域をクリアする。VAFINCはベクトル・メモリにおける実数型データのインクリメント（=1）である。

26-29:
この関数max_1dに引数として渡された配列inにおいて、対象となるベクトルを構成するn個の要素を、グローバルに定義された配列MXINに移す。これは、RTLの関数が扱う配列は先に述べたようにグローバルに定義されていなければならないからである。

32　：　mapsyncdma (13-208)

直前のRTL関数はmapclrfvであるが、これはMATH RING BUFFERにpacketを送るので、転送命令との同期をとる必要がある。pacは、24行目のmapclrfvの関数値で、この関数が終了するまで次の転送命令maplodfvは実行されない。

33 : maplodfv (13-124)
実数型配列MXINにおけるn個の要素からなるベクトルをベクトル・メモリ上のオフセットddへロードする。HFINCはホスト・メモリにおける実数型データのインクリメント (=4) である。

36 : mapsyncmath (13-213)
ベクトルのロードが完了するまで次のベクトル演算命令mapmaxfvの実行を禁止する。ここでのpacは、直前のmaplodfvの関数値である。

37 : mapmaxfv (13-141)
オフセットddから始まるn個の実数の中から最も大きな値を持つものを捜し、その値をオフセットmxに、またその最大値が存在しているベクトル・メモリ上のアドレスをオフセットadに格納する。

39 : mapsyncdma (13-208)
mapmaxfvが完了するまで、次の転送命令mapstrfvの実行を禁止する。

42 : mapstrfv (13-195)
mapmaxfvによってオフセットmxに格納された最大値を、ホスト・メモリにおけるアドレスVMAXに転送する。ここで、VMAXはグローバルに定義された要素数１の配列である（VMAXを配列として定義したのは、その名前でアドレスを表すことができるからであり、ポインタ変数等を用いても同等のことが可能である）。

45 : mapwaitdma (13-222)
ホスト・メモリへのデータ転送が終了するまで、ホスト側での計算を中止する。

48 : mapfreeva (13-102)
ＶＡを解放する。確保したベクトル・メモリの領域も解放される。

max_1d.c from hm07 printed on Tue Jul 3 22:52:46 1990

```
 1: float VMAX[1];
 2: float MXIN[1024];
 3:
 4: float max_1d (in, n)
 5: int n;
 6: float in[1024];
 7: {
 8: int i;
 9: int dd, mx, ad;
10: int pac;
11: float v_max;
12:
13: /* initialize VA */
14: mapinitva(1, 1, 0);
15:
16: /* allocation */
17: dd=mapmalloc(n, 0, VA0);
18: mx=mapmalloc(1, 0, VA0);
19: ad=mapmalloc(1, 0, VA0);
20:
21: /* clear allocation area */
22: pac = mapclrfv(dd, VAFINC, n);
23: pac = mapclrfv(mx, VAFINC, 1);
24: pac = mapclrfv(ad, VAFINC, 1);
25:
26: for (i=0; i<n; i++)
27:         {
28:         MXIN[i] = in[i];
29:         }
30:
31: /* load data */
32: mapsyncdma(pac, VA0);
33: pac = maplodfv(MXIN, HFINC, dd, VAFINC, n);
34:
35: /* calculate maximum value */
36: mapsyncmath(pac, VA0);
37: pac=mapmaxfv(dd, VAFINC, mx, ad, n);
38:
39: mapsyncdma(pac, VA0);
40:
41: /* store data */
42: mapstrfv(mx, VAFINC, VMAX, HFINC, 1);
43:
44: /* wait */
45: mapwaitdma(VA0);
46:
47: /* release VA */
48: mapfreeva(VA0);
49:
50: v_max = VMAX[0];
51: return v_max;
52: }
53:
```

図2サンプル・プログラム

３．ＶＡによるＦＦＴ

ＶＡのRTLには、表２に示すように、FFTを行う関数が既にいくつか用意されている。しかし、これらの関数を用いて実際にアプリケーション・プログラムの中でFFTを行う場合には、第２章で述べたようにＶＡのイニシャライズやベクトル・メモリにおける領域の確保など、いくつかの煩雑な付加的作業が必要となる。そこで我々は、あるFFTを実行するために必要となる一連の処理を１つの関数としてまとめることによって、より簡単にＶＡのFFTを利用することを可能にした。本章では、我々が作成したFFT関数について説明する。

３．１　入出力ベクトルの型と次元

RTLには、入力ベクトルのデータ型（実数型、複素数型）毎に異なるFFT及び逆FFTの関数が用意されている。これに対し我々は、FFT、逆FFTの入力ベクトルのデータ型をそれぞれ実数型、複素数型に限定する。これは、運動視や立体視等の視覚研究におけるFFT、逆FFTの必要性は、殆どの場合図３に示すようなフィルタリングのプロセスの中で生じるからである。一方、これらの入力ベクトルに対する出力ベクトルの型は、一般にはFFT、逆FFTともに複素数型になる。実際、RTLの実数型FFT関数、複素数型逆FFT関数は複素数型ベクトルを出力する。しかし通常我々が扱うデータでは、逆FFTの出力ベクトルは理論的に実数値になることが多く、虚数部に現れる値は誤差と考えられる。そこで、我々が作成した逆FFT関数では、得られた複素数データの実数部のみを出力するようにしてある（若干の修正を加えることによって、複素数のまま出力を取り出すことも可能）。

また、RTLには１次元、及び２次元の入力ベクトルに対するFFT関数が用意されているが、動画像処理などを行う場合には時空間の３次元画像を扱う必要がある。そこで、我々はRTLにおける１次元、と２次元のFFT、逆FFTを組み合わせて、３次元ベクトルに対するFFT、逆FFTも作成した。

表２　RTLに用意されたFFT関数

maprfftttab	実数FFTのための係数テーブル生成
mapfftttab	複素数FFTのための係数テーブル生成
maprfftnc	実数データに対する１次元FFT
mapirfftnc	実数データに対する１次元逆FFT
mapfftnc	複素数データに対する１次元FFT
mapifftnc	複素数データに対する１次元逆FFT
mapbigfftva	複素数データに対する１次元FFT（大きなベクトル用）
mapr2dfftva	実数データに対する２次元FFT
mapir2dfftva	実数データに対する２次元逆FFT
map2dfftva	複素数データに対する２次元FFT
mapi2dfftva	複素数データに対する２次元逆FFT

注：ここで、FFTのためのテーブル生成関数はすべて１次元FFTのためのものであり、２次元FFTの関数はその中で自動的に係数テーブルを生成する。

実数データ（入力画像）
↓
ＦＦＴ
↓
フィルタリング
↓
逆ＦＦＴ
↓
実数データ（出力画像）

図３　運動視及び立体視の研究において
FFTが用いられるプロセス

３．２　FFT関数を用いるための準備

３．３で述べるFFT関数を用いるためには、アプリケーション・プログラムの中で次の定数、及び構造体を定義する必要がある。

(A)ベクトルの大きさを表わす定数

１次元FFT	NFFT1D	
逆FFT	NFFT1DI	
２次元FFT	NFFT2DX	第一添字
	NFFT2DY	第二添字
逆FFT	NFFT2DXI	第一添字
	NFFT2DYI	第二添字
３次元FFT	NFFT3DT	第一添字
	NFFT3DX	第二添字
	NFFT3DY	第三添字
逆FFT	NFFT3DTI	第一添字
	NFFT3DXI	第二添字
	NFFT3DYI	第三添字

(B)複素数を表す構造体

```
typedef struct {float real, imag;} COMPLEX;
```

(C)ベクトル・アクセレータのための定数

HFINC　実数型ベクトルのホスト・メモリ上における

インクリメントの値（=4）
VAFINC　実数型ベクトルのベクトル・メモリ上における
インクリメントの値（=1）
HCINC　複素数型ベクトルのホスト・メモリ上における
インクリメントの値（=8）
VACINC　複素数型ベクトルのベクトル・メモリ上における
インクリメントの値（=2）

3．3　FFT関数

作成した6種類のFFT関数を示す。いずれも、配列inによって与えた入力ベクトルに対しFFTまたは逆FFTを実行し、その結果を配列outに出力する。なお、引数の配列in、outをグローバルに定義する必要はない。

(1)　fft_1d(in, out)　　1次元FFT
```
float in[NFFT1D]
COMPLEX out[NFFT1D]
```

(2)　fft_1d_inv(in, out)　　1次元逆FFT
```
COMPLEX in[NFFT1DI]
COMPLEX out[NFFT1DI]
```

(3)　fft_2d(in, out)　　2次元FFT
```
float in[NFFT2DX][NFFT2DY]
COMPLEX out[NFFT2DX][NFFT2DY]
```

(4)　fft_2d_inv(in, out)　　2次元逆FFT
```
COMPLEX in[NFFT2DXI][NFFT2DYI]
COMPLEX out[NFFT2DXI][NFFT2DYI]
```

(5)　fft_3d(in, out)　　3次元FFT
```
float in[NFFT3DT][NFFT3DX][NFFT3DY]
COMPLEX out[NFFT3DT][NFFT3DX][NFFT3DY]
```

(6)　fft_3d_inv(in, out)　　3次元逆FFT
```
COMPLEX in[NFFT3DTI][NFFT3DXI][NFFT3DYI]
COMPLEX out[NFFT3DTI][NFFT3DXI][NFFT3DYI]
```

(1)の1次元FFT関数の出力配列には次の離散フーリエ変換（DFT）を計算した結果が代入される。

$$out[n]=(1/N)\sum_{k=0}^{N-1} in[k]\exp(-j2\pi kn/N) \qquad n=0,1,2,\cdots,N-1$$

j:虚数単位
N:データ数（=NFFT1D）

一方、(2)の１次元逆FFT関数は

$$out[n]=\sum_{k=0}^{N-1} in[k]\exp(j2\pi kn/N) \qquad n=0,1,2,\cdots,N-1$$

j:虚数単位
N:データ数（=NFFT1DI）

を計算する（実際には３．１で述べたように、この式で得られる複素数データの実数部のみが配列out[n]に代入される）。２次元、３次元のFFT（または逆FFT）は、基本的にはこの１次元FFT（または逆FFT）をそれぞれの方向に繰り返して計算したものになっている。

これらの関数を用いるためには、３．２で述べた定数と構造体を定義する必要があるが、その他はＶＡを意識することなく、入出力ベクトルを与えるだけでFFTを実行することができる。各関数のプログラム・リストを付録１に示す。

３．４　サンプル・プログラム

図４に３次元FFT及び逆FFTを用いた簡単なサンプル・プログラムを示す。これは、関数gauss_3d(in)によって３次元配列inに生成された３次元パターンのFFTを計算し、さらにその出力outに対して逆FFTを施して、最初の配列inに戻している。インクルードされているファイル

/usr/ogata/FFTLIB/fft_3d.c
/usr/ogata/FFTLIB/fft_3d_inv.c

には、それぞれ３次元のFFT及び逆FFTの関数が定義されている。また、

/usr/ogata/FFTLIB/gauss_3d.c

には、関数gauss_3dが定義されている（この関数もＶＡを用いているので、参考のためにそのリストを付録２に示す）。

fft_test.c from hm07 printed on Thu Jul 5 12:50:14 1990

```
 1: #include <stdio.h>
 2: #include <fcntl.h>
 3: #include <aplib.h>
 4:
 5: /* size 3D vector */
 6: #define NOFX 256
 7: #define NOFY 256
 8: #define NOFT 16
 9:
10: /* size for fft */
11: #define NFFT3DX  NOFX
12: #define NFFT3DY  NOFY
13: #define NFFT3DT  NOFT
14:
15: #define NFFT3DXI NOFX
16: #define NFFT3DYI NOFY
17: #define NFFT3DTI NOFT
18:
19: /* increments of vector and host memory */
20: #define HFINC  4
21: #define HCINC  8
22: #define VAFINC 1
23: #define VACINC 2
24:
25: /* definition of the complex data type */
26: typedef struct {float real, imag;} COMPLEX;
27:
28: /* functions of 3D FFT and inverse FFT */
29: #include "/usr/ogata/FFTLIB/fft_3d.c"
30: #include "/usr/ogata/FFTLIB/fft_3d_inv.c"
31:
32: /* function of making 3D gaussian data */
33: #include "/usr/ogata/FFTLIB/gauss_3d.c"
34:
35: main()
36: {
37:
38: float in[NFFT3DT][NFFT3DX][NFFT3DY];
39: COMPLEX out[NFFT3DT][NFFT3DX][NFFT3DY];
40:
41: printf("START gauss_3d\n");
42: gauss_3d(in);
43: fft_3d(in, out);
44: fft_3d_inv(out, in);
45: }
46:
47:
```

図４サンプル・プログラム

４．おわりに

MASSCOMP5600上で高速にFFTを行うために、同計算機に登載されたベクトル演算装置ＶＡを用いた６種類のFFT関数を作成した。これらの関数を用いることによって、いくつかの定数、配列、及び構造体を定義するだけで、簡単に１次元、２次元、及び３次元のFFTまたは逆FFTを実行することができる。

ここでは、これらのFFT関数が利用される状況を考慮して、入力ベクトルの型を３．１で述べたように制限した。他の型のベクトルに対する関数、すなわち複素数型FFTや実数型逆FFTは、付録１に示したプログラム・リストにおいて、用いるRTL関数を適当なものに取り替え、若干の修正を加えることによって作成できる。また、我々が作成した逆FFT関数は、入力ベクトルの型の制限と同じ理由によって、RTLの関数が出力した複素数データの実数部のみを取り出して出力配列に代入しているが、より一般的に複素数のまま取り出すように変更することも容易にできる。

参考文献

1. Vecrot Accelerator Programmer's Manual
2. 日野幹雄:スペクトル解析，朝倉書店

付録１　FFT、及び逆FFT関数のプログラム・リスト

第３章で述べたFFT及び逆FFTのプログラム・リストを示す。これらは、MASSCOMP5600（ホスト名hm07）上のファイル

/usr/ogata/FFTLIB/fft_1d.c
/usr/ogata/FFTLIB/fft_1d_inv.c
/usr/ogata/FFTLIB/fft_2d.c
/usr/ogata/FFTLIB/fft_2d_inv.c
/usr/ogata/FFTLIB/fft_3d.c
/usr/ogata/FFTLIB/fft_3d_inv.c

に存在する。

```
 1: /*
 2: fft_1d.c
 3:
 4: This program peforms 1D FFT
 5:         by using vector accelerator
 6:
 7: April 7, 1989
 8: */
 9:
10: float   TEMP1D_IN[NFFT1D];
11: COMPLEX TEMP1D_OUT[NFFT1D/2+1];
12: float SCALE;
13:
14: fft_1d(in, out)
15:
16: COMPLEX out[NFFT1D];
17: float in[NFFT1D];
18:
19: {
20: int dslock=1, vacount=1, fftlock=0;
21: int fftdst;
22: int loglen;
23: int i;
24: int nx2 = NFFT1D/2+1;
25: int coef, fftsrc, ffttmp;
26: int pk_fft;
27: int scl;
28:
29: mapinitva(dslock, vacount, fftlock);
30:
31: /* allocation */
32: coef   =mapmalloc(NFFT1D+2, 1, VA0);
33: ffttmp =mapmalloc(NFFT1D+2, 1, VA0);
34: scl    =mapmalloc(1, 1, VA0);
35: fftsrc =mapmalloc(NFFT1D+2, NFFT1D/2, VA0);
36:
37: /* clear allocated area */
38: pk_fft=mapclrcfv(coef, VACINC, nx2);
39: pk_fft=mapclrfv(fftsrc, VAFINC, NFFT1D+2);
40: pk_fft=mapclrfv(ffttmp, VAFINC, NFFT1D+2);
41: pk_fft=mapclrfv(scl, VAFINC, 1);
42:
43: mapsyncdma(pk_fft, VA0);
44:
45:
46: /* 1 dimensional FFT */
47: printf("START 1D FFT\n");
48: loglen=mapilog2(NFFT1D);
49:
50: SCALE = 1.0/((float)(400*NFFT1D));
51: /* Scale factor should be 1/(400*NFFT1D) to get original data  */
52: /* by using the inverse FFT                                    */
53:
54: maplodfs(&SCALE, scl);
55:
56: /* make coeff. table */
57: maprfftab(coef, loglen);
58:
59: fftdst=(loglen & 1) ? ffttmp : fftsrc;
60:
61: for (i=0; i<NFFT1D; i++)
62:         {
63:         TEMP1D_IN[i]=in[i];
64:         }
65:
```

```
66: pk_fft=maplodfv(TEMP1D_IN, HFINC, fftsrc, VAFINC, NFFT1D);
67:
68: mapsyncmath(pk_fft, VA0);
69: pk_fft=maprfftnc(fftsrc, VAFINC, coef, VACINC, ffttmp, VAFINC, NFFT1D);
70: pk_fft=mapmulfsv(scl, fftdst, VAFINC, fftdst, VAFINC, NFFT1D+2);
71:
72: mapsyncdma(pk_fft, VA0);
73: mapstrcfv(fftdst, VACINC, TEMP1D_OUT, HCINC, nx2);
74:
75: mapwaitdma(VA0);
76:
77: for (i=0; i<=NFFT1D/2; i++)
78:         {
79:         out[i].real=  TEMP1D_OUT[i].real;
80:         out[i].imag=  TEMP1D_OUT[i].imag;
81:         }
82: for (i=1; i<(NFFT1D/2); i++)
83:         {
84:         out[NFFT1D-i].real=  TEMP1D_OUT[i].real;
85:         out[NFFT1D-i].imag= -TEMP1D_OUT[i].imag;
86:         }
87:
88: mapfreeva(VA0);
89: }
90:
```

```
 1: /*
 2: fft_1d_inv.c
 3:
 4: This program peforms 1D inverse FFT
 5:         by using vector accelerator
 6:
 7: April 7, 1989
 8: */
 9:
10: COMPLEX TEMP1D_C[NFFT1DI];
11:
12: fft_1d_inv(in, out)
13:
14: float out[NFFT1DI];
15: COMPLEX in[NFFT1DI];
16:
17: {
18: int dslock=1, vacount=1, fftlock=0;
19: int fftdst;
20: int loglen;
21: int i;
22: int nx2 = NFFT1DI/2+1;
23: int coef, fftsrc, ffttmp;
24: int pk_fft;
25:
26: mapinitva(dslock, vacount, fftlock);
27:
28: /* allocation */
29: coef   =mapmalloc(NFFT1DI, 1, VA0);
30: ffttmp =mapmalloc(2*NFFT1DI, 1, VA0);
31: fftsrc =mapmalloc(2*NFFT1DI, VACINC*(NFFT1DI/2), VA0);
32:
33: /* clear allocated area */
34: pk_fft=mapclrcfv(coef, VACINC, NFFT1DI/2);
35: pk_fft=mapclrcfv(fftsrc, VACINC, NFFT1DI);
36: pk_fft=mapclrcfv(ffttmp, VACINC, NFFT1DI);
37:
38: mapsyncdma(pk_fft, VA0);
39:
40:
41: /* 1 dimensional FFT */
42: printf("START 1D FFT\n");
43: loglen=mapilog2(NFFT1DI);
44:
45: /* make coeff. table */
46: mapffttab(coef, loglen);
47:
48: fftdst=(loglen & 1) ? ffttmp : fftsrc;
49:
50: for (i=0; i<NFFT1DI; i++)
51:         {
52:         TEMP1D_C[i]=in[i];
53:         }
54:
55: pk_fft=maplodcfv(TEMP1D_C, HCINC, fftsrc, VACINC, NFFT1DI);
56:
57: mapsyncmath(pk_fft, VA0);
58: pk_fft=mapifftnc(fftsrc, VACINC, coef, VACINC, ffttmp, VACINC, NFFT1DI);
59:
60: mapsyncdma(pk_fft, VA0);
61: mapstrcfv(fftdst, VACINC, TEMP1D_C, HCINC, NFFT1DI);
62:
63: mapwaitdma(VA0);
64:
65: for (i=0; i<NFFT1DI; i++)
```

```
66:          {
67:          out[i]=  TEMP1D_C[i].real;
68:          }
69: mapfreeva(VA0);
70: }
71:
```

```
 1: /*
 2: fft_2d.c
 3: A program for 2 dimensional FFT
 4:    using Vector Accelerator
 5:
 6: MAY 17, 1998
 7: */
 8: float TEMP2D_IN[NFFT2DX][NFFT2DY];
 9: float SCALE_S;
10: COMPLEX TEMP2D_OUT[NFFT2DX][NFFT2DY/2+1];
11:
12: fft_2d(in, out)
13: float in[NFFT2DX][NFFT2DY];
14: COMPLEX out[NFFT2DX][NFFT2DY];
15: {
16: int dslock=1, vacount=1, fftlock=1;
17: int i, j;
18: int ii, jj;
19: int pk_fft;
20:
21: mapinitva(dslock, vacount, fftlock);
22:
23: SCALE_S = 1.0/((float)(NFFT2DX*NFFT2DY*2));
24:
25: /* 2 dimensional FFT */
26: printf("START 2D FFT\n");
27: for (i=0; i<NFFT2DX; i++)
28:         {
29:         for (j=0; j<NFFT2DY; j++)
30:                 {
31:                 TEMP2D_IN[i][j] = in[i][j];
32:                 }
33:         }
34:
35: mapr2dfftva(TEMP2D_IN, TEMP2D_OUT, NFFT2DX, NFFT2DY, VA0);
36: printf("END 2D FFT\n");
37:
38: for (i=0; i<NFFT2DX; i++)
39:         {
40:         for (j=0; j<NFFT2DY/2+1; j++)
41:                 {
42:                 out[i][j].real=SCALE_S*TEMP2D_OUT[i][j].real;
43:                 out[i][j].imag=SCALE_S*TEMP2D_OUT[i][j].imag;
44:                 }
45:         }
46: for (i=0; i<NFFT2DX; i++)
47:         {
48:         ii=NFFT2DX-i;
49:         if(ii == NFFT2DX) ii=0;
50:         for (j=NFFT2DY/2+1; j<NFFT2DY; j++)
51:                 {
52:                 jj=NFFT2DY-j;
53:                 out[i][j].real=SCALE_S*TEMP2D_OUT[ii][jj].real;
54:                 out[i][j].imag=SCALE_S*TEMP2D_OUT[ii][jj].imag*(-1.0);
55:                 }
56:         }
57:
58:         mapfreeva(VA0);
59: }
60:
61:
62:
```

```
 1: /*
 2: fft_2d_inv.c
 3: A program for 2 dimensional inverse FFT
 4:    using Vector Accelerator
 5:
 6: May 17, 1989
 7: */
 8:
 9: COMPLEX TEMP2D_C[NFFT2DXI][NFFT2DYI];
10: fft_2d_inv(in, out)
11: float out[NFFT2DXI][NFFT2DYI];
12: COMPLEX in[NFFT2DXI][NFFT2DYI];
13: {
14: int i, j;
15: int ii, jj;
16: int dslock=1, vacount=1, fftlock=1;
17: int pk_fft;
18:
19: mapinitva(dslock, vacount, fftlock);
20:
21: /* 2 dimensional FFT */
22: printf("START 2D FFT\n");
23: for (i=0; i<NFFT2DXI; i++)
24:         {
25:         for (j=0; j<NFFT2DYI; j++)
26:                 {
27:                 TEMP2D_C[i][j] = in[i][j];
28:                 }
29:         }
30:
31: mapi2dfftva(TEMP2D_C, TEMP2D_C, NFFT2DXI, NFFT2DYI, VA0);
32: printf("END 2D FFT\n");
33:
34: for (i=0; i<NFFT2DXI; i++)
35:         {
36:         for (j=0; j<NFFT2DYI; j++)
37:                 {
38:                 out[i][j]=TEMP2D_C[i][j].real;
39:                 }
40:         }
41:         mapfreeva(VA0);
42:
43: }
44:
45:
46:
```

```
 1: /*
 2: 3d_fft.c
 3: A program for 3 dimensional FFT
 4:    using Vector Accelerator
 5:
 6: November 15, 1988
 7: */
 8: float TEMP3D_IN[NFFT3DX][NFFT3DY];
 9: float SCALE_T, SCALE_S;
10: COMPLEX TEMP3D_OUT[NFFT3DX][NFFT3DY/2+1];
11:
12: /* temporal array for 3D fft */
13: COMPLEX TEMPFFT3D[NFFT3DY][NFFT3DT];
14:
15: fft_3d(in, out)
16: float in[NFFT3DT][NFFT3DX][NFFT3DY];
17: COMPLEX out[NFFT3DT][NFFT3DX][NFFT3DY];
18: {
19: int dslock=1, vacount=1, fftlock=1;
20: int fftdst;
21: int loglen;
22: int ni, nj, iflg, jflg;
23: int i, j, k;
24: int ii, jj;
25: int scl;
26: int coef, fftsrc, ffttmp;
27: int pk_fft;
28: int ntinc=VACINC*NFFT3DT;
29: int nyt=NFFT3DY*NFFT3DT;
30: mapinitva(dslock, vacount, fftlock);
31:
32: /* allocation */
33: coef  =mapmalloc(NFFT3DT, 1, VA0);
34: fftsrc=mapmalloc(2*nyt, 1, VA0);
35: ffttmp=mapmalloc(2*nyt, 1, VA0);
36: scl   =mapmalloc(1, 1, VA0);
37:
38: /* clear allocated area */
39: pk_fft=mapclrcfv(coef, VACINC, NFFT3DT/2);
40: pk_fft=mapclrcfv(fftsrc, VACINC, nyt);
41: pk_fft=mapclrcfv(ffttmp, VACINC, nyt);
42: pk_fft=mapclrcfv(scl, VACINC, 1);
43:
44: mapsyncdma(pk_fft, VA0);
45:
46: SCALE_S = 1.0/((float)(NFFT3DX*NFFT3DY*2));
47:
48: /* 2 dimensional FFT */
49: printf("START 2D FFT\n");
50: for (k=0; k<NFFT3DT; k++)
51:         {
52:         printf("frame=%d\n",k);
53:         for (i=0; i<NFFT3DX; i++)
54:                 {
55:                 for (j=0; j<NFFT3DY; j++)
56:                         {
57:                         TEMP3D_IN[i][j] = in[k][i][j];
58:                         }
59:                 }
60:         mapr2dfftva(TEMP3D_IN, TEMP3D_OUT, NFFT3DX, NFFT3DY, VA0);
61:         printf("END 2D FFT\n");
62:         for (i=0; i<NFFT3DX; i++)
63:                 {
64:                 for (j=0; j<NFFT3DY/2+1; j++)
65:                         {
```

```
66:                             out[k][i][j].real=SCALE_S*TEMP3D_OUT[i][j].real;
67:                             out[k][i][j].imag=SCALE_S*TEMP3D_OUT[i][j].imag;
68:                             }
69:                     }
70:             for (i=0; i<NFFT3DX; i++)
71:                     {
72:                     ii=NFFT3DX-i;
73:                     if(ii == NFFT3DX) ii=0;
74:                     for (j=NFFT3DY/2+1; j<NFFT3DY; j++)
75:                             {
76:                             jj=NFFT3DY-j;
77:                             out[k][i][j].real=SCALE_S*TEMP3D_OUT[ii][jj].real;
78:                             out[k][i][j].imag=SCALE_S*TEMP3D_OUT[ii][jj].imag*(-1.0);
79:                             }
80:                     }
81:             }
82:
83: /* 1 dimensional FFT */
84: printf("START 1D FFT\n");
85: loglen=mapilog2(NFFT3DT);
86: SCALE_T = 1.0/((float)NFFT3DT);
87:
88: pk_fft=maplodfs(&SCALE_T, scl);
89:
90: /* make coeff. table */
91: mapffttab(coef, loglen);
92:
93: fftdst=(loglen & 1) ? ffttmp : fftsrc;
94: for (i=0; i<NFFT3DX; i++)
95:         {
96:         for (j=0; j<NFFT3DY; j++)
97:                 {
98:                 for (k=0; k<NFFT3DT; k++)
99:                         {
100:                        TEMPFFT3D[j][k]=out[k][i][j];
101:                        }
102:                }
103:        pk_fft=maplodcfv(TEMPFFT3D, HCINC, fftsrc, VACINC, nyt);
104:        mapsyncmath(pk_fft, VA0);
105:        for (j=0; j<NFFT3DY; j++)
106:                {
107:                pk_fft=mapfftnc(fftsrc+j*ntinc, VACINC,
108:                coef, VACINC, ffttmp+j*ntinc, VACINC, NFFT3DT);
109:                }
110:        pk_fft=mapmulfsv(scl, fftdst, VAFINC, fftdst, VAFINC, 2*nyt);
111:
112:        mapsyncdma(pk_fft, VA0);
113:
114:        mapstrcfv(fftdst, VACINC, TEMPFFT3D, HCINC, nyt);
115:
116:        mapwaitdma(VA0);
117:
118:        for (j=0; j<NFFT3DY; j++)
119:                {
120:                for (k=0; k<NFFT3DT; k++)
121:                        {
122:                        out[k][i][j]=TEMPFFT3D[j][k];
123:                        }
124:                }
125:        }
126:        mapfreeva(VA0);
127: }
128:
```

```
 1: /*
 2: 3d_fft_inv.c
 3: A program for 3 dimensional inverse FFT
 4:    using Vector Accelerator
 5:
 6: December 7, 1988
 7: */
 8:
 9: COMPLEX TEMP3D_C[NFFT3DXI][NFFT3DYI];
10:
11: /* temporal array for 3D fft */
12: COMPLEX TEMPFFT3DI[NFFT3DYI][NFFT3DTI];
13:
14: fft_3d_inv(in, out)
15: float out[NFFT3DTI][NFFT3DXI][NFFT3DYI];
16: COMPLEX in[NFFT3DTI][NFFT3DXI][NFFT3DYI];
17: {
18: int dslock=1, vacount=1, fftlock=1;
19: int fftdst;
20: int loglen;
21: int ni, nj, iflg, jflg;
22: int i, j, k;
23: int ii, jj;
24: int coef, fftsrc, ffttmp;
25: int pk_fft;
26: int ntinc=VACINC*NFFT3DTI;
27: int nyt=NFFT3DYI*NFFT3DTI;
28: mapinitva(dslock, vacount, fftlock);
29:
30: /* allocation */
31: coef=mapmalloc(NFFT3DTI, 1, VA0);
32: fftsrc=mapmalloc(2*nyt, VACINC*NFFT3DTI/2, VA0);
33: ffttmp=mapmalloc(2*nyt, 1, VA0);
34:
35: /* clear allocated area */
36: pk_fft=mapclrcfv(coef, VACINC, NFFT3DTI/2);
37: pk_fft=mapclrcfv(fftsrc, VACINC, nyt);
38: pk_fft=mapclrcfv(ffttmp, VACINC, nyt);
39: mapsyncdma(pk_fft, VA0);
40:
41: /* 2 dimensional FFT */
42: printf("START 2D FFT\n");
43: for (k=0; k<NFFT3DTI; k++)
44:         {
45:         printf("frame=%d\n",k);
46:         for (i=0; i<NFFT3DXI; i++)
47:                 {
48:                 for (j=0; j<NFFT3DYI; j++)
49:                         {
50:                         TEMP3D_C[i][j] = in[k][i][j];
51:                         }
52:                 }
53:         mapi2dfftva(TEMP3D_C, TEMP3D_C, NFFT3DXI, NFFT3DYI, VA0);
54:         printf("END 2D FFT\n");
55:         for (i=0; i<NFFT3DXI; i++)
56:                 {
57:                 for (j=0; j<NFFT3DYI; j++)
58:                         {
59:                         in[k][i][j]=TEMP3D_C[i][j];
60:                         }
61:                 }
62:         }
63:
64: /* 1 dimensional FFT */
65: printf("START 1D FFT\n");
```

```
 66: loglen=mapilog2(NFFT3DTI);
 67:
 68: /* make coeff. table */
 69: mapffttab(coef, loglen);
 70:
 71: fftdst=(loglen & 1) ? ffttmp : fftsrc;
 72: for (i=0; i<NFFT3DXI; i++)
 73:         {
 74:         for (j=0; j<NFFT3DYI; j++)
 75:                 {
 76:                 for (k=0; k<NFFT3DTI; k++)
 77:                         {
 78:                         TEMPFFT3DI[j][k]=in[k][i][j];
 79:                         }
 80:                 }
 81:         pk_fft=maplodcfv(TEMPFFT3DI, HCINC, fftsrc, VACINC, nyt);
 82:         mapsyncmath(pk_fft, VA0);
 83:         for (j=0; j<NFFT3DYI; j++)
 84:                 {
 85:                 pk_fft=mapifftnc(fftsrc+j*ntinc, VACINC,
 86:                 coef, VACINC, ffttmp+j*ntinc, VACINC, NFFT3DTI);
 87:                 }
 88:         mapsyncdma(pk_fft, VA0);
 89:         mapstrcfv(fftdst, VACINC, TEMPFFT3DI, HCINC, nyt);
 90:         mapwaitdma(VA0);
 91:         for (j=0; j<NFFT3DYI; j++)
 92:                 {
 93:                 for (k=0; k<NFFT3DTI; k++)
 94:                         {
 95:                         out[k][i][j]=TEMPFFT3DI[j][k].real;
 96:                         }
 97:                 }
 98:         }
 99:         mapfreeva(VA0);
100: }
101:
102:
103:
```

付録２　関数gauss_3dのプログラム・リスト

図４に示したサンプル・プログラムで用いられている関数gauss_3dのプログラム・リストを示す。ここで、この関数の引数in[k][i][j]は２次元画像の時系列を表現しており、（i,j）が２次元画像を、kが時刻を表している。本関数は、各時刻（k）の画像上に、中心が（x_stp+k*vx, y_stp+k*vy）で標準偏差stdv_sの２次元ガウスパターンを作成し、さらに各位置（i,j）における画素値の時間方向の変化を、中心NOFT/2、標準偏差stdv_tの１次元ガウス関数で変調した３次元パターンを生成する。このような画像の時系列を、各画素値を輝度としてグラフィックスで呈示すると、２次元ガウス関数が、位置（x_stp, y_stp）から速度（vx, vy）でコントラストを変化させながら移動しているように見える。x_stp、y_stp、vx、vy、stdv_s、stdv_tはgauss_3dの要求に応じて入力すればよい。この関数は、MASSCOMP5600（ホスト名hm07）上のファイル

/usr/ogata/FFTLIB/gauss_3d.c

に存在する。

```
 1: float X_DD[NOFX];
 2: float Y_DD[NOFY];
 3: float T_DD[NOFT];
 4: short x_len, y_len, t_len;
 5: gauss_3d (in)
 6: float in[NOFT][NOFX][NOFY];
 7: {
 8: int i, j, k;
 9: int x_stp, y_stp;
10: int x_axis, y_axis, t_axis;
11: int x_gauss, y_gauss, t_gauss;
12: int x_ovrf, y_ovrf, t_ovrf;
13: int x_ovlen, y_ovlen, t_ovlen;
14: int hf, lmt;
15: int pac;
16: int x_spd, y_spd;
17: int work;
18: int n_max;
19: float stdv_s, stdv_t;
20: float x_mean, y_mean, t_mean;
21: float x_base, y_base, t_base;
22: float delta_s, delta_t;
23: float half;
24: float limit;
25:
26: printf ("Enter standard deviation (spatial temporal)\n");
27: scanf ("%f %f", &stdv_s, &stdv_t);
28: printf ("Enter speed (vx vy)\n");
29: scanf ("%d %d", &x_spd, &y_spd);
30: printf ("Enter start point (x y)\n");
31: scanf ("%d %d", &x_stp, &y_stp);
32:
33: n_max = (NOFX<NOFY)?NOFY:NOFX;
34: n_max = (n_max<NOFT)?NOFT:n_max;
35: half = -0.5;
36: limit = -80.0;
37: delta_s = 1.0/stdv_s;
38: delta_t = 1.0/stdv_t;
39:
40: /* VA initialization */
41: mapinitva(1, 1, 0);
42:
43: /* VA memory allocation */
44:         /* work = work area for mapexpfv */
45:         work=mapmalloc(2*n_max, 0, VA0);
46:
47:         /* axis = argument area of gauss function  */
48:         x_axis=mapmalloc(NOFX, 0, VA0);
49:         y_axis=mapmalloc(NOFY, 0, VA0);
50:         t_axis=mapmalloc(NOFT, 0, VA0);
51:
52:         /* gauss = gauss function value area */
53:         x_gauss=mapmalloc(NOFX, 0, VA0);
54:         y_gauss=mapmalloc(NOFY, 0, VA0);
55:         t_gauss=mapmalloc(NOFT, 0, VA0);
56:
57:         /* ovrf = overflow address area for exp. calculation */
58:         x_ovrf=mapmalloc(NOFX, 0, VA0);
59:         y_ovrf=mapmalloc(NOFY, 0, VA0);
60:         t_ovrf=mapmalloc(NOFT, 0, VA0);
61:
62:         /* ovlen = number of overflow */
63:         x_ovlen=mapmalloc(1, 0, VA0);
64:         y_ovlen=mapmalloc(1, 0, VA0);
65:         t_ovlen=mapmalloc(1, 0, VA0);
```

```
 66:
 67:         /* constant area */
 68:         hf=mapmalloc(1, 0, VA0);
 69:         lmt=mapmalloc(1, 0, VA0);
 70:
 71: /* VA memory clear */
 72: mapclrfv(work, VAFINC, 2*n_max);
 73:
 74: mapclrfv(x_axis, VAFINC, NOFX);
 75: mapclrfv(y_axis, VAFINC, NOFY);
 76: mapclrfv(t_axis, VAFINC, NOFT);
 77:
 78: mapclrfv(x_gauss, VAFINC, NOFX);
 79: mapclrfv(y_gauss, VAFINC, NOFY);
 80: mapclrfv(t_gauss, VAFINC, NOFT);
 81:
 82: mapclrfv(x_ovrf, VAFINC, NOFX);
 83: mapclrfv(y_ovrf, VAFINC, NOFY);
 84: mapclrfv(t_ovrf, VAFINC, NOFT);
 85:
 86: pac=mapclrfv(hf, VAFINC, 1);
 87: pac=mapclrfv(lmt, VAFINC, 1);
 88:
 89: /* VA load constant */
 90: mapsyncdma(pac, VA0);
 91: pac=maplodfs(&half, hf);
 92: pac=maplodfs(&limit, lmt);
 93:
 94: /* temporal gauss */
 95: t_mean = 8.0;
 96: t_base = -t_mean/stdv_t;
 97:
 98: mapsyncmath(pac, VA0);
 99: mapclrfv(t_ovlen, VAFINC, 1);
100: maprampfv(&t_base, &delta_t, t_axis, VAFINC, NOFT);
101: mapmulfvv(t_axis, VAFINC, t_axis, VAFINC, t_axis, VAFINC, NOFT);
102: mapmulfsv(hf, t_axis, VAFINC, t_axis, VAFINC, NOFT);
103: pac=mapcxafslt(t_axis, VAFINC, lmt, t_axis, VAFINC, t_ovrf, VAFINC, t_ovlen, NOFT);
104: mapsyncdma(pac, VA0);
105: mapstrwv(t_ovlen, VAFINC, &t_len, 2, 1);
106: mapwaitdma(VA0);
107: if(t_len > 0) mapsctrfv(lmt, 0, t_ovrf, VAFINC, t_ovlen);
108: pac=mapexpfv(t_axis, VAFINC, t_gauss, VAFINC, NOFT);
109: mapsyncdma(pac, VA0);
110: pac=mapstrfv(t_gauss, VAFINC, T_DD, HFINC, NOFT);
111:
112: /* motion of 2D gauss */
113: for (k=0; k<NOFT; k++)
114:         {
115:         printf ("k=%d\n", k);
116:
117:         /* spatial gauss at k-frame */
118:         x_mean = x_spd*k + x_stp;
119:         y_mean = y_spd*k + y_stp;
120:         x_base = -x_mean/stdv_s;
121:         y_base = -y_mean/stdv_s;
122:
123:         mapclrfv(x_ovlen, VAFINC, 1);
124:         mapclrfv(y_ovlen, VAFINC, 1);
125:
126:         maprampfv(&x_base, &delta_s, x_axis, VAFINC, NOFX);
127:         maprampfv(&y_base, &delta_s, y_axis, VAFINC, NOFY);
128:
129:         mapmulfvv(x_axis, VAFINC, x_axis, VAFINC, x_axis, VAFINC, NOFX);
130:         mapmulfvv(y_axis, VAFINC, y_axis, VAFINC, y_axis, VAFINC, NOFY);
```

```
131:
132:        pac=mapmulfsv(hf, x_axis, VAFINC, x_axis, VAFINC, NOFX);
133:        pac=mapmulfsv(hf, y_axis, VAFINC, y_axis, VAFINC, NOFY);
134:
135:        pac=mapcxafslt(x_axis, VAFINC, lmt, x_axis, VAFINC, x_ovrf, VAFINC, x_ovlen, NOFX);
136:        pac=mapcxafslt(y_axis, VAFINC, lmt, y_axis, VAFINC, y_ovrf, VAFINC, y_ovlen, NOFY);
137:
138:        mapsyncdma(pac, VA0);
139:        mapstrwv(x_ovlen, VAFINC, &x_len, 2, 1);
140:        mapstrwv(y_ovlen, VAFINC, &y_len, 2, 1);
141:
142:        mapwaitdma(VA0);
143:        if(x_len > 0) mapsctrfv(lmt, 0, x_ovrf, VAFINC, x_len);
144:        if(y_len > 0) mapsctrfv(lmt, 0, y_ovrf, VAFINC, y_len);
145:
146:        pac=mapexpfv(x_axis, VAFINC, x_gauss, VAFINC, NOFX);
147:        pac=mapexpfv(y_axis, VAFINC, y_gauss, VAFINC, NOFY);
148:
149:        mapsyncdma(pac, VA0);
150:        mapstrfv(x_gauss, VAFINC, X_DD, HFINC, NOFX);
151:        mapstrfv(y_gauss, VAFINC, Y_DD, HFINC, NOFY);
152:
153:        mapwaitdma(VA0);
154:
155:        for (i=0; i<NOFX; i++)
156:                {
157:                for (j=0; j<NOFY; j++)
158:                        {
159:                        in[k][i][j] = X_DD[i]*Y_DD[j]*T_DD[k];
160:                        }
161:                }
162:        }
163:
164:        /* VA free */
165:        mapfreeva(VA0);
166: printf ("END OF GAUSS\n");
167: }
168:
```