◇◇◇◇◇
解　説
◇◇◇◇◇

MathLink を利用した Mathematica のちょっと変わったプログラミング

木村　広[1]

1　はじめに

Mathematica が生まれてから 10 年がたつ．ユーザの数も着実に増えている．フロントエンドが大きく強化された最新版のバージョン 3.0 では，アプリケーション自体の名前も Mathematica から大文字のいかつい MATHEMATICA へと変わった．

人間が手でおこなうには非効率的な計算を人間に代わって実行し，人間がもっと本来の仕事をできるように発明されたものがコンピュータであるとすれば，C や Fortran など低級な言語で解くには実際的でない問題を扱うために開発された言語が Mathematica であると言えるだろう．

たとえば，1000 桁の円周率が必要になる計算を C や Fortran でプログラムしようとしたら，円周率を 1000 桁求めるためのプログラムをユーザ自身が開発するか，あるいは，どこからか見つけて来る必要がある．一方，Mathematica では `N[Pi, 1000]` とタイプするだけである．ユーザは，いかに円周率を 1000 桁計算するかに腐心することなく，1000 桁の円周率を必要とした問題を解くことに集中することができる．

Mathematica はインタプリタ型の言語である．C や Fortran などでコンパイルし，カリカリに最適化したコードよりも確かに実行は遅い．しかし，コードの柔軟性，再利用性はもとより，最近のハードウェアの向上のおかげか，それとも Mathematica 自体の最適化が進んだのか，以前ほど遅さは感じなくなっている．実行に時間がかかるのはプログラムのしかたが悪いせいもある．

以下の Mathemetica セッションを見てみよう．これは 100 行 100 列の実数行列を作り，その行列の掛け算を Mathematica の組み込み関数を使ったプログラム (In[4]) と行列の掛け算の定義を素直に書き写したプログラム (In[5]) とで計算し，その計算時間を測ったものである．二つのプログラムの実行結果はまったく同じだが (Out[6])，計算時間は実に 1,000 倍近い差が生じている．

```
In[1]:= size=100;
In[2]:= a=Table[Random[Real],{size},{size}];
In[3]:= b=Table[Random[Real],{size},{size}];
```

[1]工学部 数理情報基礎講座，hkim@mns.kyutech.ac.jp

```
In[4]:= Timing[c1=a.b;]
Out[4]= {0.269989 Second,Null}

In[5]:= Timing[c2=Table[
        (For[sum=0.0;m=1, m<=size, m++, sum+=a[[i,m]] b[[m,j]]];
        sum),{i,size},{j,size}];]
Out[5]={255.73 Second,Null}

In[6]:= c1==c2
Out[6]= True
```

プログラムを工夫すれば Mathematica はかなり速くなる.

それでも，やはり，コンパイルしたコードを利用してプログラムのスピードアップをはかりたい場合もある．既存の C や Fortran のライブラリを Mathematica から利用したいこともあるだろうし，純粋な Mathematica では実現が不可能な機能もある.

ここで登場するのが MathLink である.

MathLink は Mathematica のカーネルとフロントエンドとを結ぶ通信のためのプロトコル，通信経路 (図 1のフロントエンドとカーネルとを結ぶトンネルみたいなもの．リンクと呼ぶ)，および，プログラム開発のためのライブラリの総称である.

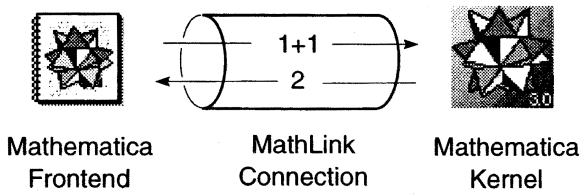

図 1: フロントエンドに入力された式 $1+1$ はリンクを通ってカーネルに届けられる．カーネルはリンクから式を受け取り，評価する．評価値 2 がカーネルからリンクに返され，フロントエンドがそれを受け取る.

Mathematica の成功の理由の一つは，ユーザインタフェースを提供するフロントエンドと計算を実行するカーネルとを上手に分離したことにある．フロントエンドとカーネルとが通信できなければセッションそのものが成り立たないから，Mathematica において MathLink は影ながら中心的な役割を果たしていると言っても過言ではない.

カーネルとフロントエンドの通信ばかりではなく，MathLink はカーネルと C や Fortran などの外部プログラムとの通信，あるいはカーネルどうしの通信に利用することが可能である．そのような通信の手段によって仕事をプログラム間で上手に分担することができれば，計算のスピードアップはもちろん，Mathematica の可能性はさらに広がるだろう.

本稿ではその MathLink を利用した Mathemaitca のプログラミングについて解説する．C プログラムから Mathematica の関数を呼ぶ方法，次に Mathematica から C の関数を呼ぶ方法，最後に Mathematica カーネル同士を通信させる方法について順に具体例をあげて説明する．技術評論社のコンピュータ雑誌，SoftwareDesign の 1996 年 7~9 月号に連載した「MathLink+DO で NEXTSTEP プログラミング」も合わせて読んでもらえれば，理解の助けになると思う．

2 Mathematica の関数を C プログラムから呼び出す

まず，Mathematica の関数を C プログラムの中から呼び出し，その結果を C プログラムで利用する方法を紹介する．この節の例題は円周率 1000 桁に含まれる 0...9 の数字の出現頻度を数えることとする．

2.1 Mathematica を呼ぶ C プログラムの例

以下が Mathematica と通信をしながら問題を解く C プログラムの例である．プログラムは，Mathematica との通信経路を確立し，円周率を 1000 桁求める Mathematica の関数を呼び，その結果を文字列として受け取り，受け取った文字列中の数字の出現頻度を数える．

```
/* pidigit.c */
/* calling mathematica function from within c program */
/* 1997/07/20, hkim@melt.kyutech.ac.jp */
#include <stdio.h>
#include "mathlink.h"
void main(argc, argv)
int argc;
char* argv[];
{
    MLINK link;
    char **pi;
    char *cp;
    int count[]={0,0,0,0,0,0,0,0,0,0};
    int i;

    link = MLOpen(argc, argv);
    if (link==NULL) exit(-1);

    MLPutFunction(link, "ToString", 1);
    MLPutFunction(link, "N", 2);
```

```
    MLPutSymbol(link, "Pi");
    MLPutInteger(link, 1000);
    MLEndPacket(link);
    while (MLNextPacket(link) != RETURNPKT);
    MLNewPacket(link);
    MLGetString(link, pi);
    MLClose(link);

    cp=*pi;
    while(*cp!=(char)NULL) {
        count[*cp-'0']++;
        cp++;
    }
    for(i=0;i<10;i++)
        printf("%2d %3d\n",i,count[i]);
}
```

インクルードしている “mathlink.h” は，MathLink のヘッダファイルである．このヘッダファイルは後述する MathLink コンパイラ mcc のサーチパス上に見つかる．

プログラム中，ML で始まる関数が MathLink のライブラリ関数である．どんなライブラリ関数が提供されているかについては，The MATHEMATICA Book [5] を参照されたい．

関数 MLOpen() は用意されたリンクにこのプログラムをつなぐ役目をする．リンク自体の作成法は次の 2.2 節で述べる．

MLOpen() に成功したら，Mathematica に実行させたい関数を以下の手順でリンクへ送信する．

1. まず，MLPutFunction() で Mathematica の関数名とその関数の引き数の個数を MLOpen() の返したリンク link へ送る．
2. そのあと，引き数の型に応じて，MLPutInteger()，MLPutReal()，MLPutString()，MLPutSymbol() などを適宜呼び出し，関数の引き数をリンクへ送る．
3. 関数とその引き数をすべて送信したら，最後に MLEndPacket() を唱える．

この例題でカーネルに実行させたいのは合成関数 `ToString[N[Pi,1000]]` である．`ToString[exp]` は `exp` を文字列に変換する関数，`N[exp, digit]` は `exp` の評価値を精度 `digit` で求める関数である．`ToString[N[Pi,1000]]` をカーネルに実行させるには，まず外側の関数の名前 `ToString` を送り，続いて `ToString[]` の唯一の引き数となる `N[Pi,1000]` を送信する．Mathematica 中では，`Pi` はたんなる文字

列ではなく，円周率を表すシンボルとして扱われるため，文字列 "Pi" を MLPutSymbol() で送信することに注意しよう．関数 N は Pi と 1000 との二つの引き数をとるので， MLPutFunction() の第 3 引き数は 2 となる．

カーネルは MLEndPacket() を受け取ると式の評価をはじめ，計算を終えると RETURNPKT という特別なパケットをリンクに返す．

C プログラム側ではそのパケットをリンクから受け取った後，カーネルの返すデータの型に応じた関数を呼んでデータを受信する．ここでは文字列が返されるので，呼ぶべき関数は MLGetString() である．

2.2 コンパイルおよび実行

pidigit.c を Mathematica と一緒に配布される MathLink コンパイラ mcc (実体はシェルスクリプト) を用いてコンパイルする．mcc は Mathematica をデフォルトでインストールした UNIX マシンであれば，/usr/local/bin/ の下に見つかるはずだ．

```
% mcc pidigit.c -o pidigit
```

コンパイルに成功したら，pidigit を以下の引き数つきで実行すると，pidigit は MathLink 通信のためのリンクをひとつ作成し，リンクの一方に自分自身を接続する．

```
% pidigit -mathlink -linkmode listen
```

しかし，画面には，

```
Link created on:  3713@rosinante
```

が出たきり，何も起こらない．それもそのはず，リンクを作るには作ったが，リンクのもう一方がちゅうぶらりんで，ここに Mathematica がつながらなければ通信が成立しない．pidigit はリンクのもう一方に Mathematica がつながるのを待っている．

画面に表示された “3713@rosinante” はホスト名 rosinante のマシン の TCP/IP の 3713 番ポートにリンクが作成されたことを意味している．MathLink はコマンドが呼びだされた時点で空いているポートを自動的に探し出し割り当てるので[2]，ユーザは通信に使うポートを心配する必要はない．

Mathematica を起動し，リンクに接続させよう．

リンクはすでに作成ずみで，ポート番号とホスト名もわかっている．この場合，コマンド math を以下の引き数つきで実行すればよい．

```
% math -mathlink -linkmode connect -linkname "3713@rosinante"
```

[2]リンクは TCP/IP ばかりでなく AppleTalk 上にも作成可能であるが，ここでは述べない．

オプションなしのコマンド名 `math` で起動した Mathematica は標準入力から入力された式を評価してその結果を標準出力へ出力する．一方，`-mathlink` のオプションを与えられた Mathematica は MathLink のリンクと入出力するようになる．今回，リンクそのものはすでに `pidigit` によって作成済みだから新たに作成の必要はない．これが `-linkmode connect` の意味である．リンクの実体はホスト名 rosinante のマシン上の 3713 ポートにある．`-linkname` にはそのリンクの情報を与える．これで Mathematica は `pidigit` が一方につながったリンクのもう一方につながる．そして通信が開始される．

無事，円周率 1000 桁に含まれる 0...9 の数字の出現頻度が，93，116，103，103，93，97，94，95，100，106 と求まったはずだが，どうだろう?

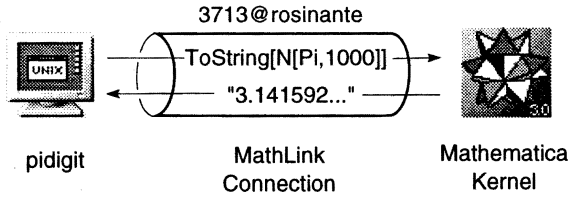

図 2: `pidigit` が式 `ToString[N[Pi, 1000]]` をリンク "3713@rosinante" に送信し，その式をリンクから受け取った Mathematica カーネルが式の評価値をリンクに戻す．

3 Mathematica 内部から C の関数を呼び出す

Mathematica 内部から C の関数を呼びだすことはさらに簡単だ．C のプログラムを MathLink コンパチブルにするためのテンプレートファイルが提供されていて，多くの手続きが自動化されているためである．前節では，MathLink のライブラリ関数，MLPutFunction() や MLGetString() を直接呼びだしてデータのやり取りをしていた．テンプレートファイルを利用すると，特別な場合をのぞいて，データをやり取りするコードをプログラム中に明示的に書く必要がない．

ここでは，[0,1] の範囲で発生させた乱数のペア (x, y) が半径 1 の円の内部に落ちる確率から円周率を求めるモンテカルロ法を例題とする．モンテカルロ法では，コンピュータが発生する乱数の乱数としての正しさとともに，十分な回数，実験を繰り返すことも円周率を高い精度で推定するための重要なファクタである．

一回の実験で発生する乱数を 1,000,000 組みとし，その実験を 100 回繰り返し，実験ごとに推測された円周率をグラフで表してみよう．

ちなみに，Mathematica で円周率をモンテカルロ法で解くプログラムの一例は以下のようなものである．

```
In[7]:= monte[n_]:=If[Random[Real]^2+Random[Real]^2<=1, n+1, n]
In[8]:= monteCalro[n_]:= 4N[Nest[monte,0,n]/n]
In[9]:= monteCalro[10^6]//Timing
```

```
Out[9]={418.217 Second,3.142}
```

1 回の実験に 420 秒弱を要していて，とても 100 回も実験する気にはなれない．計算自体は単純な繰り返しなので，C の利用によるスピードアップは十分に見込まれる．

3.1 テンプレートファイル

テンプレートファイルは Mathematica のパターンと C の関数とのインタフェース，つまり，Mathematica でどんなパターンが評価されようとした時に C プログラムのどんな関数をどのような引き数で呼び出し，その関数はどんな戻り値をとるかを定義するファイルである．テンプレートファイルには ".tm" のサフィックスをつける決まりになっている．

MathLink コンパイラ mcc はテンプレートファイル *file*.tm をプリプロセスし，C のファイル *file*.tm.c を作成する．このファイルは mcc の引き数に与えられた他のソースファイル *file*.c とともに cc でコンパイルされる．

C の関数 monte_calro() を Mathematica の monteCalro[] の評価時に呼び出したいとすれば，テンプレートファイルに記述すべき内容は以下のようになる．テンプレートファイル中，コロン二つ (::) で始まる行はコメント行である．大文字と小文字は区別される．

```
:: monteCarlo.tm
:: template file to convert
:: C program into MathLink compatible one
:: 1997/07/22, hkim@melt.kyutech.ac.jp
:Begin:
:Function:      monte_calro
:Pattern:       monteCalro[seed_Integer, iter_Integer]
:Arguments:     {seed, iter}
:ArgumentTypes: {Integer, Integer}
:ReturnType:    Real
:End:
```

一対の:Begin: と :End: とで囲まれる部分をエントリーと呼ぶ．エントリー中の :Function: には呼び出したい C の関数名，:Pattern: には :Function: に示された C の関数を呼び出すべき Mathematica のパターン，:Arguments: には関数への引き数，: ArgumentTypes: には引き数の型，そして: ReturnType: には関数の戻り値の型を書く．C の int，double に対応するものは Mathematica では Integer，Real である．マシンサイズを越える Integer はとうぜん C の int では扱い切れない．ここのところはちょっとだけ注意が必要である．

Mathematica から呼び出したい C の関数が複数個ある場合は，それぞれの関数に応じたエントリーをテンプレートファイル中に記述する必要がある．

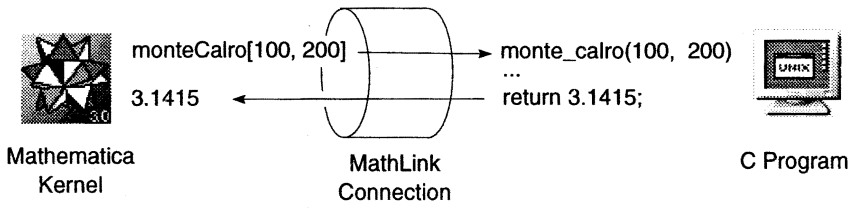

図 3: カーネルが パターン monteCalro[Integer, Integer] にマッチする式を評価しようとすると，double を返す外部関数 monte_calro(int, int) が呼び出される．その戻り値はカーネルに Real として届けられる．

3.2 C プログラム

以下はモンテカルロ法を実行する C プログラムである．main() 中で呼び出される唯一の関数 MLMain() の定義が見当たらないこと，関数 monte_calro() の定義はあっても呼び出しが見当たらないことにすぐ気がつくだろう．それらは前節で説明したテンプレートファイルから MathLink コンパイラ mcc が自動的に定義，あるいは，Mathematica から呼びだせるよう，準備してくれる．

```
/* monteCalro.c */
/* calcurate pi by monte-calro method */
/* 1997/07/21, hkim@melt.kyutech.ac.jp */
#include <stdio.h>
#include "mathlink.h"
#define MAX 2147483647 /* 2**31-1 */
#define sq(x) (x)*(x)

int main(argc,argv)
    int argc;
    char *argv[];
{
    return MLMain(argc, argv);
}

double monte_calro(seed, iter)
    int seed;
    int iter;
{
    int i;
    int inside;
    double x,y;
```

```
    srandom(seed);
    inside=0;
    for(i=0;i<iter;i++) {
        x = (double)random()/MAX;
        y = (double)random()/MAX;
        if (sq(x)+sq(y)<=1)
          inside++;
    }
    return(double)inside/iter;
}
```

3.3 コンパイルそして実行

インタフェースを定義するテンプレートファイル，および，インプリメンテーションを記述する C のファイルとができ上がったら mcc でそれらをコンパイルする．mcc を -g オプションつきで実行すると，デバッグ用のコードを生成するとともに，テンプレートファイルをプリプロセスして自動的に生成した C プログラム (ここでは monteCalro.tm.c) をコンパイル終了後もデリートせず，カレントディレクトリに残す．興味のあるユーザはそのファイルを調べてみるのもいいだろう．

```
% mcc -g monteCalro.c monteCalro.tm -o monteCalro
```

コンパイルが完了したら，-mathlink -linkmode listen の引き数を与えて monteCalro を起動する．

```
% monteCalro -mathlink -linkmode listen
Link created on: 3825@rosinante
```

monteCalro が表示したリンクの識別子 “3285@rosinante” を引き数に，LinkMode は Connect を指定して，Mathematica で Install[] を評価する．

```
In[10]:= link=Install["3285@rosinante", LinkMode->Connect];
```

Install[] は monteCalro[] の定義をリンクから読み込むとともに，テンプレートファイル中にインタフェースが記述された C の関数を Mathematica から呼びだせるようにお膳立てをする関数である．膳立ての内容は?monteCalro によって確認できる．

```
In[11]:= ?monteCalro
```

```
"Global`monteCalro"
monteCalro[seed_Integer, iter_Integer] :=
  ExternalCall[LinkObject["3825@rosinante", 3, 3],
    CallPacket[0, {seed, iter}]]
```

C で定義した関数 monte_calro() に対応する Mathematica の関数 `monteCalro[]` が Global コンテキスト中に見つかっている．`monteCalro[]` の呼び出しは，リンクオブジェクトと関数 `CallPacket[]` とを引き数とする `ExternalCall[]` 関数の呼び出しに引き継がれる．`CallPacket[]` の第 1 引き数の 0 が，C の関数 monte_calro() を指しているのだが，これは monteCalro.tm.c ファイルを読めばわかる．

こうしてでき上がった C の関数を呼び出す MathLink 関数 `monteCalro[]` は他の Mathematica 関数と同様に扱うことができる．

```
In[12]:= ListPlot[
    result = 4 Table[monteCalro[Random[Integer,2^16-1],10^6],{100},
    PlotJoined->True];
In[13]:= (Plus  @@ result)/Length[result]
Out[13]= 3.14163
```

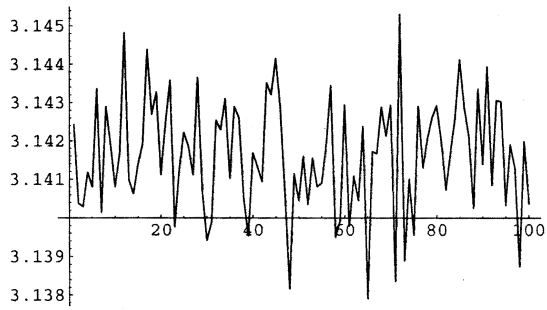

図 4: モンテカルロ法を使った円周率の推測値．C で計算し，Mathematica でグラフを描いた．

総計一億回の実験で，小数第 3 位まで正しい円周率が求まった．計算時間は 127.7 秒で，Mathematica だけで実行したときの 1/350 以下の時間しかかからない．グラフィクスも Mathematica のライブラリを利用したのでストレスなくプログラムできた．

4 複数のカーネルを通信させて並列計算

この章のテーマは MathLink を使ったカーネルどうしの通信である．

情報科学センターの山之上助教授が PVM の説明で扱った数値積分の並列化 [6] を MathLink を利用して模倣してみる．並列化のココロは，積分区間を分割して別々に計算しても全体の積分値は変わらないことにある．

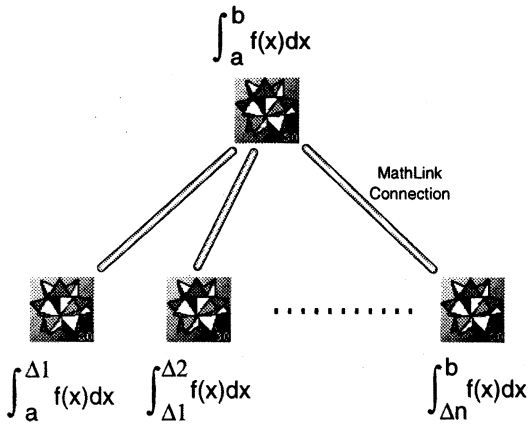

図 5: 積分区間を分割し，別のカーネルで計算させる．

積分区間を n 個にわけ，各小区間の積分を n 個のカーネル (以後，子カーネル) に実行させ，求まった小区間の積分値を別のカーネル (以後，親カーネル) で総和するという作戦で並列化を考えよう．

4.1 子カーネル側のコード

小区間の積分を受け持つ子カーネルでは以下のコードを実行しておく．このプログラムはリンクから式を読み込み，それを評価してリンクに書き出すことを，リンクから文字列 “bye” が読み込まれるまで延々と繰り返す．

```
readEvalWriteLoop[link_]:=
  Module[{exp},
      While[True,
        While[! LinkReadyQ[link], (* should we sleep here? *)];
        exp=LinkRead[link];
        While[Head[exp]==Hold,exp=ReleaseHold[exp]];
        LinkWrite[link,exp];
        If[exp=="bye", Break[]]]]
```

LinkReadyQ[link] は link にデータが届いており，かつ，データが読み出し可の状態であれば True，それ以外は False を返す．式は親カーネルでの評価を抑制するため，Hold[] でくくられて子カーネルに届く (次の 4.2 節を参照)．ReleaseHold[] はその Hold[] をはがす役目を持っている．

4.2 親カーネル側のコード

子カーネルが返す小区間の積分値を総和する親カーネルのコードは次のようなものである．

```
par[links_,funcs_]:=
```

```
  Module[{},
    MapThread[LinkWrite[#1, #2]&, {links, funcs}];
    While[! (And @@ Map[LinkReadyQ, links]), (* should we sleep here? *) ];
    Map[LinkRead, links]]

stop[links_]:= par[links,Table["bye",{Length[links]}]]

parIntegrate[links_, f_, {x_,min_,max_}, opt___]:=
  Module[{i,n,ranges},
    n=Length[links];
    ranges =
      Map[Flatten[{x,#}]&,Partition[Table[i,{i,min, max, (max-min)/n}],2,1]];
    Plus @@ par[links, Map[Hold[NIntegrate[f,#,opt]]&,ranges]]]
```

parIntegrate[] は第一引き数に与えられた子カーネルの数に応じて積分区間を分割し，それを被積分関数とともに Hold[] で包んで関数 par[] に渡す．par[] は，第一引き数に与えられたリンクに第二引き数で与えられた式を書きだす．リンクにつながる子カーネルは受け取った式を評価し，リンクに戻す．par[] の戻り値は子カーネルが評価した小区間の積分値のリストになっているから，そのリストを総和すれば求めるべき積分値が得られる．

4.3 計算の実行

準備ができたら子カーネルの数に応じて親カーネルで以下の式を実行し，通信の経路となるリンクを作成し，親カーネルをそれぞれのリンクの一方に接続しておく．ここでは 4 つの子カーネルを利用するのでリンクを 4 つ作成している．

```
In[14]:= mac=LinkCreate[]
Out[14]:= LinkObject["2639@slab",2,2]

In[15]:= windows=LinkCreate[]
Out[15]= LinkObject["2640@slab",3,3]

In[16]:= hpux=LinkCreate[]
Out[16]= LinkObject["2641@slab",4,4]

In[17]:= next=LinkCreate[]
Out[17]= LinkObject["2642@slab",5,5]
```

そのあと，LinkCreate[] の返したリンクの識別子 (LinkObject[] の第一引き数の *"port@host"*) に応じて，それぞれの子カーネルで次の式を評価し，リンクのもう一方に子カーネルを接続する．

```
(mac)In[2]:= link=LinkConnect["2639@slab", LinkProtocol->"TCP"];
(mac)In[3]:= LinkWrite[link, "mac is ready"]; readEvalWriteLoop[link]

(windows)In[2]:= link=LinkConnect["2640@slab", LinkProtocol->"TCP"];
(windows)In[3]:= LinkWrite[link, "windows is ready"]; readEvalWriteLoop[link]

(hpux)In[2]:= link=LinkConnect["2641@slab", LinkProtocol->"TCP"];
(hupx)In[3]:= LinkWrite[link, "hpux is ready"]; readEvalWriteLoop[link]

(nextstep)In[2]:= link=LinkConnect["2642@slab", LinkProtocol->"TCP"];
(nextstep)In[3]:= LinkWrite[link, "nextstep is ready"]; readEvalWriteLoop[link]
```

次に親カーネルで次の式を評価する．

```
In[18]:= (Map[While[ ! LinkReadyQ[#], Pause[1]]; LinkRead[#])&,
            {mac, windows, hpux, next}]
```

子カーネル全員が ready と言ってきたところで準備は整った．積分開始だ．

まず，子カーネルの能力を測っておこう．parIntegrate[] の第一引き数に積分を実行させたいマシンへのリンクを一つだけ指定すると，指定したマシン単体で積分を実行した時の計算時間がわかる[3]．

```
In[19]:=(st=SessionTime[];
    ans=parIntegrate[{hpux},
    4/(1+x^2),{x,0,1},WorkingPrecision->200];
    SessionTime[]-st)
```

この方法で測った積分の計算時間は NEXTSTEP(Petium, 166MHz) で 5.67 秒，Windows(Pentium, 166MHz) で 5.27 秒，HP-UX(hp-pa, 100MHz) で 3.70 秒，PowerBook(PowerPC, 180MHz) で 3.94 秒である．PowerBook が意外に速い．残念ながらバージョンの違いで今回の並列計算に参加できなかった情報科学センターの SONY NEWS 上の Mathematica 2.2 で同等の計算を実行すると計算時間は 2.21 秒であった．

それではいよいよ 4 つのカーネルに仕事を分担させて積分を実行してみよう．それには parInegrate[] の第一引き数を {hpux, next, mac, windows} に変えるだけでよい．

[3]式中の WorkingPrecision→200 は，計算中の数値の精度を 200 桁確保するオプションである．デフォルトは 16 桁なのだが，それでは計算時間が短すぎて並列化によるスピードアップが測定できなかった．

```
In[23]:=(st=SessionTime[];
    ans=parIntegrate[{hpux, next, mac, windows},
    4/(1+x^2),{x,0,1},WorkingPrecision->200];
    SessionTime[]-st)
```

このプログラムを 10 回実行したときの平均計算時間は 1.70 秒 (最高 1.61 秒，最低は 1.75 秒) であり，情報センターの単体の NEWS で Mathematica を使って計算した場合よりも 30% 程度高い性能を示している．

ちなみにこの並列計算で求められた円周率は以下の値である．一番最後の '9' をのぞく 189 桁まで正しい．正しい円周率は 190 桁目以降，89543... と続く．

```
3.14159265358979323846264338327950288419716939937510582097494459230781
64062862089986280348253421170679821480865132823066470938446095505822 31
72535940812848111745028410270193852110555964462294895 49
```

この MathLink を利用した並列プログラムは，スタンダードな Mathematica の枠内で完全に実現できる．ヘテロジニアスな CPU や OS 上の Mathematica を利用する場合であってもプログラムを一切変えることなく動作し，かつ，仕事を分担させる子カーネルをインタラクティブに選択できるという特徴がある．プログラム開発のしやすさはもちろんだが，Mathematica の性格に応じて仕事を分担させるといった，きめの細かい並列計算ができる可能性もある．同じ Mathematica と言ってもプラットフォームによって数値計算に強いものやパターンマッチが速いものなど，得手不得手がある．

プログラムがそのまま積分以外の計算にも応用可能なことも特徴のひとつにあげておこう．たとえば次の式は Mac と Windows の子カーネルに並列に作成した行列の掛け算を求めるものである．

```
In[33]:= par[{mac, windows}, {Hold[size=100], Hold[size=100]}];
In[34]:= Dot @@ par[{mac, windows},
                    {Hold[a=Table[Random[Real],{size},{size}]],
                     Hold[b=Table[Random[Real],{size},{size}]]}]
```

以上の特徴は Mathematica がインタプリタ型の言語であることの影響が大だが，MathLink が高いレベルでプログラム間の通信をサポートしているためであることも忘れてはいけない．

まとめの節に入る前に，子カーネルのループを止めよう．

```
In[35]:= stop[{mac, windows, next, hpux}]
Out[35]= {"bye","bye","bye","bye"}
```

5 まとめ

“Mathematica は使いたいんだがスピードがどうも” とか, “過去のライブラリがあるので Mathematica に簡単には乗り換えられない” という話をちらほら耳にする. そういう場合, MathLink を利用したプログラミングが威力を発揮する. 本稿がそのようなユーザの刺激となったらさいわいである. MathLink のプログラミングにぜひ挑戦してみてほしい. MathLink に関する書籍は現在, The MATHEMATICA Book [5] をのぞいて皆無と言っていいほど見かけないが, その The MATHEMATICA Book も通り一辺の説明に終わっている印象が強い. 文献 [3] が出版されていれば良い参考になるだろう.

参考文献

[1] 木村 広: MathLink+DO で NEXTSTEP プログラミング, SoftwareDesign 7,8,9 月号, 1996.

MathLink を使って Mathematica から NEXTSTEP の GUI を利用するプログラムについて解説しました.

`http://www.melt.kyutech.ac.jp/~hkim/Programming/SD/`

[2] Kimura, H. and C. Miyaji: An Implementation of Internet Applications in Mathematica, Innovaion of Mathematics, Proceesings of the Second International Mathematica Symposium, 307–314. 1997.

Mathematica で書いたメールリーダおよび Web クライアントです. POP, SMTP, HTTPD との通信を MathLink プログラムで実現し, html のインタプリタを Mathematica で書いています.

`http://www.melt.kyutech.ac.jp/~hkim/Programming/IMS-97/`

[3] 宮地 力: Mathematica によるネットワークプログラミング, 岩波コンピュータサイエンスシリーズ (もうすぐ出版).

MathLink プログラム全般について系統だてて書いてある始めての本になる予定です. MovieDigitizer や Macintosh の ToolBox をアクセスして実現する Mathematica のリアルタイムグラフィクスなど, すぐ使えるプログラムが満載です.

[4] Miyaji, C. and H. Kimura: Writing A Graphical User Interface for Mathemtica using Mathematica and MathLink, Innovaion of Mathematics, Proceesings of the Second International Mathematica Symposiun, 345–352, 1997.

`http://www.taiiku.tsukuba.ac.jp/~miyaji/ims97/`

[5] Wolfram, S. : *The MATHEMATICA Book*, Wolfram Media, Cambridge, 1996.

Mathematica 3.0 のパッケージにはこの本が丸ごとオンラインドキュメントとして提供されています. 驚き. 1997 年 3 月に日本語版が出る予定でしたが...

[6] 山之上卓: PVM による並列計算,
`http://www.tobata.isc.kyutech.ac.jp/res-tebiki/pvm/index.html`

付録: サイトライセンスによる MATHEMATICA のインストール

九工大は Mathematica の開発元である Wolfram リサーチ社とサイトライセンスの契約を結んでいる．九工大の帳簿に載っており (つまり備品であり)，かつ，ハードウェア，OS がインストールの条件を満たしているコンピュータであれば，あらゆるコンピュータに Mathematica をインストールすることができる．利用にあたってはサイトライセンスの内容を遵守すべきことは言うまでもない．契約違反があった場合は，違反したユーザだけでなく，九工大に対するライセンスが永久に破棄されてしまう．

サイトライセンスを利用するには，

1. サイトライセンス利用申請の手続き
2. オブジェクトのインストール
3. パスワードの申請
4. パスワードのインストール

の 3 つの手続きが必要である．

サイトライセンス利用申請の手続き

これはサイトライセンスされた Mathematica をワークステーションやパソコンにインストールしようと考えるユーザが，九工大のサイトライセンスを管理する情報科学センターにたいしてインストールの許可を得るためのものである．以下の URL から申請の画面にたどり着ける．

`http://www.isc.kyutech.ac.jp/touroku/mathematica.html`

情報科学センターは電子メール以外の申請を現在は受けつけていないようである．申請が認められると，センターからメールが届く．そのメールを印刷し，署名・捺印をしてセンターに向かおう．印刷したメールと引き替えに，Mathematica のインストーラが入った CD-ROM が手渡される．Mathematica 2.2 までは，Macintosh，Windows のインストーラはフロッピーで配布されていたが，3.0 からはすべて CD-ROM となった．この原稿を書いている時点で，Mathematica 3.0 がインストール可能な OS は，Windows95, WindowsNT, Macintosh, PowerMacintosh, SunOS, Solaris, HP-UX, SGI, Linux, NEXTSTEP である．残念ながらセンターの SONY NEWS 用の Mathematica 3.0 の開発は中止とのことである．SONY NEWS で利用可能な Mathematica はバージョン 2.2 である．

オブジェクトのインストール

借りて来た CD-ROM をしかるべき方法でマウントする．

Windows や Macintosh はマウントできた CD-ROM をダブルクリックするとインストーラが起動する．UNIX マシンなら，CD-ROM を /MATHEMATICA としてマウントしたとすれば，

```
# cd /MATHEMATICA/Unix/Installers/machine/
# ./MathInstall
```

を実行するとインストールが始まる．SunOS にインストールする場合は上のコマンドライン中の *machine* を SunOS とすること．

インストーラがインストール先ディレクトリを聞いてくるので，デフォルト (/usr/local/mathematica/) が気にいらなかったら適当なディレクトリを指定する．

さらにインストールが進むと，インストーラは Mathematica を利用するユーザ名，ユーザの所属する組織，Mathematica のパスワードを聞いて来る．Windows，Macintosh のばあいは，情報科学センターで教えてもらった情報をしかるべき場所に正確に打ち込む．英字の大文字と小文字は区別される．UNIX マシンのばあいは，パスワードの入力を後回しにして先に進もう．パスワードの取得については後の節で述べる．インストールが終了したあとでパスワードを設定してもまったく問題はない．パスワードを設定せずにインストールした Mathematica を起動すると利用に必要な情報を聞いて来るから，取得したパスワードその他をここで正確に入力すればよい．

オブジェクトがインストールできたら CD-ROM は忘れずに情報科学センターに戻すこと．

パスワードの取得

Mathematica は正しいパスワードがなければ動かない．Mac 版や Windows 版は九工大のサイトライセンス用にパスワードがすでに発行されていてそれを使えるのだが，UNIX 版の場合は，Mathematica をインストールしたユーザ (UNIX マシンの管理者) がなんらかの方法でパスワードを取得する必要がある．

センターで CD-ROM と一緒に貸し出してくれるインストールマニュアルには，パスワード取得の方法がいくつも書いてあるが，この原稿を書いている時点 (1997 年 8 月 7 日) で，

- FAX は不安．未処理の FAX が山のように積み上げられているのを Wolfram 社で目撃したという証言がある．
- サイトライセンスのパスワードは WWW では取得できたことがない．ページには “3.0 Site Password Generator” とあるのだが，正しい情報を入力しても返って来るのは machine type mismatch ばかり．
- register@wolfram.com はちょっとおとぼけ．3.0 のパスワードを申請してもバージョン 2.2 のパスワードしか送って来ない．

ということである．そこで，パスワード申請に関しては，情報科学センターが用意している以下の Web ページを参照してほしい．

`http://www.isc.kyutech.ac.jp/touroku/mathe_passwd.html`

パスワードを申請するためには，Mathematica を利用するユーザの情報と共に，サイトライセンスのライセンス ID，Mathematica をインストールするマシンの名前と ID(MathId と呼ばれる) が必要になる．ライセンス ID はセンターで教えてもらう．マシン名と MathId はさきほどオブジェクトをインストールした CD-ROM のディレクトリ上で，

```
# ./MathInstaller -info
```

を実行したときの出力中の Machine name: と MathID: の右の文字列である．MathID の他，regcard.txt 中の適切な場所に必要な情報を書き込んで送信すると，二，三日中にはパスワードが届く．

パスワードのインストール

パスワードをインストールしていない Mathematica を立ち上げるとウィンドウが開きパスワードを聞いて来るので，取得したパスワードその他の情報をそのウィンドウ上のしかるべき場所にタイプする．この方法がもっとも確実だと思う．一度正確なパスワードを入力したらライセンスがエクスパイアするまでそのウィンドウは二度と現われない．

もうひとつの方法は /usr/local/mathematica/Configuration/Licensing/mathpass ファイルに Machine name, MathID, ライセンス ID, 取得したパスワードをスペースで区切ってエディタ等で書き込むという手である．このファイルパスはデフォルト値であり，Mathematica をインストールする際にデフォルト以外のディレクトリを指定した場合はそちらのディレクトリ以下のしかるべきファイルをエディットすることになる．ファイルの内容はたとえばこんな風になる．

```
rosinate 4741-62900-32437 L2614-6268 9847-057-196:::980422
```

スタンドアローンとネットワークライセンス

UNIX 版の Mathematica には，パスワードを取得したそのマシンでのみ Mathematica を実行できるスタンドアローンライセンスと，取得したひとつのパスワードをネットワークで共有するネットワークライセンスがある．ネットワークライセンスを利用すれば Mathematica を起動するマシン毎にパスワードを申請するのは不要なので，管理はぐっと楽になる．ネットワークライセンスを利用する場合，パスワードを取得したマシンで，

```
# /usr/local/mathematica/SystemFiles/LicenseManager/Binaries/machine/mathlm
```

を実行しておく．mathlm をライセンスサーバと言う．*machine* の部分はライセンスサーバを起動するマシンのアーキテクチャに合わせる．SunOS の場合は SunOS となる．Mathematica 2.2 のライセンスサーバは mathlm ではなく mathserver であり，両者に互換性はない．その上で，ネットワークライセンスを利用するすべてのマシンの Mathematica について，取得したネットワークライセンスのパスワードをインストールしておく．ネットワークライセンスを利用するマシンで Mathematica を起動すると，まずライセンスサーバにパスワードをチェックに行く．ライセンスサーバが起動を許可すると，あとはスタンドアロンの Mathematica とまったく同じく利用できる．ライセンスサーバに起動の許可を得に来たマシンのログはデフォルトでサーバ上の mathlm と同じディレクトリの log ファイルに残る．

1 年ごとの更新

サイトライセンスは 1 年ごとに更新が必要である．更新時期が近づいたら情報科学センターからサイトライセンスに関するアナウンスがあるので，忘れずにパスワードを更新しよう．