数値計算 第２回課題 高速行列乗算
(6.43sec.@Celeron 466MHz x2)

G99P126-1 牧野知仁

出題日 2001/ 5/18
提出日 2001/ 6/ 2

1 問題

1000x1000 行列 (double) の積を高速に計算するプログラムを作れ。10 回別の行列のペアに適用して，平均の計算時間を出せ。尚，自分の計算機環境を示し，最も早い計算時間を出したものは試験を免除する（試験の点の配分分だけ満点をあげる)。

2 制作・実行環境

自分の自作デスクトップ PC(1999 年 11 月購入・組立) を使用した。

Main Board ABIT BP6

FSB 66MHz

CPU1 Intel Celeron 466MHz 128KBytes L2 Cache

CPU2 Intel Celeron 466MHz 128KBytes L2 Cache

Memory SDRAM 256MBytes

OS Microsoft Windows 2000 Professional Service Pack 1

Compiler Microsoft Visual C++ 6.0 Professional Edition

3 方針

今回の課題では先人のやってきたこととして、[1] が紹介されている。読んでみたが、先人と同じことをやったのでは生産的ではない。というよりむしろ、そう簡単に越えられるものではないだろう。

さて、自分のマシンの最大の特徴はデュアルプロセッサである。そこで、マルチスレッドプログラムを組めば性能を活かしきれると考えられる。これは結構大変な作業だと思われるが、私の C

の実力は全然たいしたこと無いし、勿論マルチスレッドプログラミングが未経験である（さらには他の教科のレポートに忙しいとかいろいろあるがまあ、関係無い)。

しかし、折角このようなマシンを持っているのだし、是非やってみたいので、[1] のプログラムをそのまま借用し、マルチスレッドに改造することに時間的にも精神的にも集中することとする。

4 オリジナル版

方針で述べた事と矛盾しているが、プログラムの観察も兼ねてまずはシングルスレッド版、つまりオリジナル版のプログラムで NBLC(部分行列の大きさ) の調整を行ってみた。いろいろ変えてみて実行してみた。

4.1 プログラム

dimmats.h 設定のためのヘッダファイル。

```
/* Header File for Function: mult_mats1 */
#define NMAX 1000    /* Maximum Matrix Size */
#define NBLC 95        /* Block Matrix Size */
```

multmats.c 行列高速乗算関数の本体。

```
/*
        Function multmats(int n, double A[NMAX][NMAX], double B[NMAX][NMAX], double C[NMAX][NMAX]
        Purpose             : Matrix Multiplication for Full Square Matrices
        File Name           : multmats.c
        Version (Date)      : 1.0 (2001/1/22)
        Written by          : Takeshi OGITA (luther@geocities.co.jp)
        Platform            : Any
        CPU                 : Any
        OS                  : Any (Windows, UNIX, etc.)
        Compiler            : Any (Windows;Visual C++, UNIX;gcc, etc.)
        Remarks             : Not Any.
*/

#include "dimmats.h"

/* Prototype */
void multmats(int n, double A[NMAX][NMAX], double B[NMAX][NMAX], double C[NMAX][NMAX]);
void mult_block(int mi, int mj, int mk,
                double WA[NBLC][NBLC], double WB[NBLC][NBLC], double WC[NBLC][NBLC]);
/* Prototype */

void multmats(int n, double A[NMAX][NMAX], double B[NMAX][NMAX], double C[NMAX][NMAX])
{
    int i, j, iloop, jloop, kloop, m, me, is, js, ks;
    double WA[NBLC][NBLC], WB[NBLC][NBLC], WC[NBLC][NBLC];

    /* Initialization */
    for(i=0; i<n; i++){
        for(j=0; j<n; j++) C[i][j] = 0.0;
    }

    m = n/NBLC;
    me = n%NBLC;
```

```
for(iloop=0; iloop<m; iloop++){
    is = NBLC*iloop;
    for(jloop=0; jloop<m; jloop++){
        js = NBLC*jloop;
        /* Copy A to WA */
        for(i=0; i<NBLC; i++){
            for(j=0; j<NBLC; j++) WA[i][j] = A[is+i][js+j];
        }
        for(kloop=0; kloop<m; kloop++){
            ks = NBLC*kloop;
            /* Copy B to WB */
            for(i=0; i<NBLC; i++){
                for(j=0; j<NBLC; j++) WB[i][j] = B[js+i][ks+j];
            }
            /* Block Matrix Multiplication(WC = WA*WB) */
            mult_block(NBLC, NBLC, NBLC, WA, WB, WC);
            /* Add WC to C */
            for(i=0; i<NBLC; i++){
                for(j=0; j<NBLC; j++) C[is+i][ks+j] += WC[i][j];
            }
        }
        ks = NBLC*m;
        /* Copy B to WB */
        for(i=0; i<NBLC; i++){
            for(j=0; j<me; j++) WB[i][j] = B[js+i][ks+j];
        }
        /* Block Matrix Multiplication(WC = WA*WB) */
        mult_block(NBLC, me, NBLC, WA, WB, WC);
        /* Add WC to C */
        for(i=0; i<NBLC; i++){
            for(j=0; j<me; j++) C[is+i][ks+j] += WC[i][j];
        }
    }
    js = NBLC*m;
    /* Copy A to WA */
    for(i=0; i<NBLC; i++){
        for(j=0; j<me; j++) WA[i][j] = A[is+i][js+j];
    }
    for(kloop=0; kloop<m; kloop++){
        ks = NBLC*kloop;
        /* Copy B to WB */
        for(i=0; i<me; i++){
            for(j=0; j<NBLC; j++) WB[i][j] = B[js+i][ks+j];
        }
        /* Block Matrix Multiplication(WC = WA*WB) */
        mult_block(NBLC, NBLC, me, WA, WB, WC);
        /* Add WC to C */
        for(i=0; i<NBLC; i++){
            for(j=0; j<NBLC; j++) C[is+i][ks+j] += WC[i][j];
        }
    }
    ks = NBLC*m;
    /* Copy B to WB */
    for(i=0; i<me; i++){
        for(j=0; j<me; j++) WB[i][j] = B[js+i][ks+j];
    }
    /* Block Matrix Multiplication(WC = WA*WB) */
    mult_block(NBLC, me, me, WA, WB, WC);
    /* Add WC to C */
    for(i=0; i<NBLC; i++){
```

```
        for(j=0; j<me; j++) C[is+i][ks+j] += WC[i][j];
    }
}

is = NBLC*m;
for(jloop=0; jloop<m; jloop++){
    js = NBLC*jloop;
    /* Copy A to WA */
    for(i=0; i<me; i++){
        for(j=0; j<NBLC; j++) WA[i][j] = A[is+i][js+j];
    }
    for(kloop=0; kloop<m; kloop++){
        ks = NBLC*kloop;
        /* Copy B to WB */
        for(i=0; i<NBLC; i++){
            for(j=0; j<NBLC; j++) WB[i][j] = B[js+i][ks+j];
        }
        /* Block Matrix Multiplication(WC = WA*WB) */
        mult_block(me, NBLC, NBLC, WA, WB, WC);
        /* Add WC to C */
        for(i=0; i<me; i++){
            for(j=0; j<NBLC; j++) C[is+i][ks+j] += WC[i][j];
        }
    }
    ks = NBLC*m;
    /* Copy B to WB */
    for(i=0; i<NBLC; i++){
        for(j=0; j<me; j++) WB[i][j] = B[js+i][ks+j];
    }
    /* Block Matrix Multiplication(WC = WA*WB) */
    mult_block(me, me, NBLC, WA, WB, WC);
    /* Add WC to C */
    for(i=0; i<me; i++){
        for(j=0; j<me; j++) C[is+i][ks+j] += WC[i][j];
    }
}
js = NBLC*m;
/* Copy A to WA */
for(i=0; i<me; i++){
    for(j=0; j<me; j++) WA[i][j] = A[is+i][js+j];
}
for(kloop=0; kloop<m; kloop++){
    ks = NBLC*kloop;
    /* Copy B to WB */
    for(i=0; i<me; i++){
        for(j=0; j<NBLC; j++) WB[i][j] = B[js+i][ks+j];
    }
    /* Block Matrix Multiplication(WC = WA*WB) */
    mult_block(me, NBLC, me, WA, WB, WC);
    /* Add WC to C */
    for(i=0; i<me; i++){
        for(j=0; j<NBLC; j++) C[is+i][ks+j] += WC[i][j];
    }
}
ks = NBLC*m;
/* Copy B to WB */
for(i=0; i<me; i++){
    for(j=0; j<me; j++) WB[i][j] = B[js+i][ks+j];
}
/* Block Matrix Multiplication(WC = WA*WB) */
mult_block(me, me, me, WA, WB, WC);
```

```
    /* Add WC to C */
    for(i=0; i<me; i++){
        for(j=0; j<me; j++) C[is+i][ks+j] += WC[i][j];
    }
}

void mult_block(int mi, int mj, int mk,
                double WA[NBLC][NBLC], double WB[NBLC][NBLC], double WC[NBLC][NBLC])
{
    int i, j, k;
    double s;

    /* (j,i,k); Inner Product Version */
    for(j=0; j<mj; j++){
        for(i=0; i<mi; i++){
            s = 0.0;
            for(k=0; k<mk; k++){
                s += WA[i][k]*WB[k][j];
            }
            WC[i][j] = s;
        }
    }
}
```

mltmatst.c　mltmats.c の性能をテストするためのプログラム。

```
/* Test Program of Function: multmats */

#include "dimmats.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define Timer clock()

double A[NMAX][NMAX], B[NMAX][NMAX], C[NMAX][NMAX], D[NMAX][NMAX];

/* Prototype */
int mflops(double flops, double time);
void set_mat(int n, double A[NMAX][NMAX], double B[NMAX][NMAX]);
void multmats0(int n, double A[NMAX][NMAX], double B[NMAX][NMAX], double C[NMAX][NMAX]);
void multmats(int n, double A[NMAX][NMAX], double B[NMAX][NMAX], double C[NMAX][NMAX]);
double check(int n, double C[NMAX][NMAX], double D[NMAX][NMAX]);
/* Prototype */

int main()
{
    int n;
    double sec, flops;
    clock_t tic, toc;

    printf("input n (<= %d)\n", NMAX);
    printf(" n = "); scanf("%d", &n);

    set_mat(n, A, B);    /* Matrices Setting */
    flops = 2.0*n*n*n;

    printf("*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]\n");
```

```
    /* Normal(Definition) Matrix Multiplication */
    tic = Timer; multmats0(n, A, B, D); toc = Timer;
    sec = (double)(toc - tic)/CLOCKS_PER_SEC;
    printf(" definition  %10.2f %13d            ***\n", sec, mflops(flops, sec));

    /* Block Matrix Multiplication */
    tic = Timer; multmats(n, A, B, C); toc = Timer;
    sec = (double)(toc - tic)/CLOCKS_PER_SEC;
    printf(" multmats    %10.2f %13d %17.1e\n", sec, mflops(flops, sec), check(n, C, D));

    return 0;
}

int mflops(double flops, double sec)
{
    int mf;

    if (sec > 1.0e-14) mf = (int)(flops/sec/1000000);
    else mf = 0;

    return mf;
}

void set_mat(int n, double A[NMAX][NMAX], double B[NMAX][NMAX])
{
    int i, j;

    srand( (unsigned)time( NULL ) );

    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            A[i][j] = (double)rand()/1000.0;
            B[i][j] = (double)rand()/1000.0;
        }
    }
}

double check(int n, double C[NMAX][NMAX], double D[NMAX][NMAX])
{
    int i, j;
    double temp, max_dif;

    max_dif = 0.0;
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            temp = fabs(C[i][j] - D[i][j]);
            if(temp > max_dif) max_dif = temp;
        }
    }

    return max_dif;
}

void multmats0(int n, double A[NMAX][NMAX], double B[NMAX][NMAX], double C[NMAX][NMAX])
{
    int i, j, k;

    /* Initialization */
    for(i=0; i<n; i++){
        for(j=0; j<n; j++) C[i][j] = 0.0;
    }
```

```
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            for(k=0; k<n; k++){
                C[i][j] += A[i][k]*B[k][j];
            }
        }
    }
}
```

4.2 結果

結果は次の通り。

ここではそれぞれ一回しか載せていないが、実際にやった時は同じのを何度もやったりして、特に 90〜100 はどれも 3 回以上実行している。他のプロセスの動きにも気を使ったせいか、0.1 秒以内に差がおさまったので、あまりこだわるのは無意味と判断し最速値を掲載した。(※注：他の OS ではどうか調べていないか、clock() 関数が返す CPU 時間には他のプロセスの動作によるキャッシュヒット率低下の影響か、多くのプロセスを動作させると明らかに差が出る。)

なお、Check[Error] が一部を除いてとても大きな数字になっているが、これは definition(定義通りの乗算の計算) の計算の行の実行を省いたため、それとの比較ができていないからである。実際にこんなことが起こっているわけではない。

```
ORIGINAL EDITION:

NBLC = 50
input n (<= 1000)
 n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 definition       44.38            45           ***
 multmats         12.36           161          1.2e-009

NBLC = 75
input n (<= 1000)
 n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 multmats         12.30           162          3.0e+005

NBLC = 80
input n (<= 1000)
 n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 multmats         12.48           160          3.1e+005

NBLC = 85
input n (<= 1000)
 n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 multmats         12.20           163          3.0e+005

NBLC = 90
input n (<= 1000)
 n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 multmats         12.39           161          3.0e+005

NBLC = 91
```

```
input n (<= 1000)
 n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 multmats         13.09           152          3.0e+005

NBLC = 92
input n (<= 1000)
 n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 multmats         12.09           165          3.1e+005

NBLC = 93
input n (<= 1000)
 n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 multmats         13.05           153          3.0e+005

NBLC = 94
input n (<= 1000)
 n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 multmats         12.19           164          3.0e+005

NBLC = 95
input n (<= 1000)
 n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 definition       44.83            44             ***
 multmats         12.06           165          1.1e-009

NBLC = 96
input n (<= 1000)
 n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 multmats         19.59           102          3.0e+005

NBLC = 97
input n (<= 1000)
 n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 multmats         12.08           165          3.0e+005

NBLC = 98
input n (<= 1000)
 n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 multmats         12.94           154          3.0e+005

NBLC = 99
input n (<= 1000)
 n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 multmats         12.17           164          3.1e+005

NBLC = 100
input n (<= 1000)
 n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 multmats         12.20           163          3.1e+005

NBLC = 105
```

```
input n (<= 1000)
 n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 multmats         14.30           139          3.0e+005

NBLC = 110
input n (<= 1000)
 n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 multmats         12.66           158          3.0e+005

NBLC = 125
input n (<= 1000)
 n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 multmats         17.34           115          3.0e+005

NBLC = 150
input n (<= 1000)
 n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 multmats         30.16            66          3.0e+005
```

以上より、NBLC=95 の時が最速であると分かる。

5 マルチスレッド版

こちらが今回挙げるプログラムのうち、最速である。

5.1 プログラム

dimmats.h 設定のためのヘッダファイル。

スレッドの数を表す NTHR が追加されている。

```
/* Header File for Function: mult_mats1 */
#define NMAX 1000   /* Maximum Matrix Size */
#define NBLC 55     /* Block Matrix Size */
#define NTHR 2      /* Number of Thread  */
```

mltmats.c 行列高速乗算関数の本体。

全ての変更はマルチスレッド化のためだけである。手法は [3] の WindowsSDK 編,[4] のマルチスレッド関連記事を参考にした。

また、スレッド起動時に多次元配列を渡すのが難しい（左辺値にならない）ので、ポインタを直接的に使うよう書き換えた。その際、[2] が参考になった。

```
/*
        Function multmats(int n, double *A, double *B, double *C)
        Purpose          : Matrix Multiplication for Full Square Matrices
        File Name        : multmats.c
        Version (Date)   : 1.0 (2001/1/22) by OGITA
                         : 1.0m(2001/6/ 1) by MAKINO
        Written by       : Takeshi OGITA (luther@geocities.co.jp)
        Modified by      : Tomohito MAKINO (g99p1261@mn.waseda.ac.jp)
```

```
        Platform        : Any
        CPU             : Any (Recommendation : 2 CPUs)
        OS              : Windows (Recommendation : NT/2000)
        Compiler        : Windows;Visual C++
        Remarks         : Not Any.
*/

#include "dimmats.h"
#include <process.h>
#include <stdlib.h>

#ifdef _DEBUG
#include <stdio.h>
#endif

typedef struct{
    int ThreadID;
    int n;
    int volatile *RunningThreads;
}*VALS,dVALS;

static double *A,*B,*C;

/* Prototype */
void multmats(int n,double *A0,double *B0,double *C0);
void mult_blocks_thread(void *data);
void mult_block(int mi, int mj, int mk,
                double WA[NBLC][NBLC],double WB[NBLC][NBLC],double WC[NBLC][NBLC]);
/* Prototype */

void multmats(int n,double *A0,double *B0,double *C0)
{
    int i,j;
    int volatile RunningThreads;
    dVALS data[NTHR];

    A=A0;
    B=B0;
    C=C0;

    RunningThreads=0;

    /* Initialization */
    for(i=0; i<n; i++){
        for(j=0; j<n; j++) C[i*n+j] = 0.0;
    }

    data[0].ThreadID = 0;   data[0].n=n;
    data[0].RunningThreads = &RunningThreads;

    for(i=1;i<NTHR;i++){
        data[i].ThreadID = i;   data[i].n=n;
        data[i].RunningThreads = &RunningThreads;
        _beginthread(mult_blocks_thread,0,(void*)&data[i]);
    }

    mult_blocks_thread((void*)&data[0]);

    while(RunningThreads>0);
}
```

```
void mult_blocks_thread(void *arg){
    int iloop,jloop,kloop,is,js,ks,i,j;
    double WA[NBLC][NBLC], WB[NBLC][NBLC], WC[NBLC][NBLC];
    int n,m,me,id;
    VALS data=(VALS)arg;

    (*(data->RunningThreads))++;

    n=data->n;
    id=data->ThreadID ;

    m = n/NBLC;
    me = n%NBLC;

#ifdef _DEBUG
    printf("ID=%d: n=%d m=%d me=%d runninng=%d\n",id,n,m,me,(*(data->RunningThreads)));
#endif

    for(iloop = m*id/NTHR ; iloop < m*(id+1)/NTHR ; iloop++){
        is = NBLC * iloop;
        for(jloop=0; jloop<m; jloop++){
            js = NBLC*jloop;
            /* Copy A to WA */
            for(i=0; i<NBLC; i++){
                for(j=0; j<NBLC; j++)
                    WA[i][j]= A[(is+i)*n+(js+j)];
            }
            for(kloop=0; kloop<m; kloop++){
                ks = NBLC*kloop;
                /* Copy B to WB */
                for(i=0; i<NBLC; i++){
                    for(j=0; j<NBLC; j++) WB[i][j] = B[(js+i)*n+(ks+j)];
                }
                /* Block Matrix Multiplication(WC = WA*WB) */
                mult_block(NBLC, NBLC, NBLC, WA, WB, WC);
                /* Add WC to C */
                for(i=0; i<NBLC; i++){
                    for(j=0; j<NBLC; j++) C[(is+i)*n+(ks+j)] += WC[i][j];
                }
            }
            ks = NBLC*m;
            /* Copy B to WB */
            for(i=0; i<NBLC; i++){
                for(j=0; j<me; j++) WB[i][j] = B[(js+i)*n+(ks+j)];
            }
            /* Block Matrix Multiplication(WC = WA*WB) */
            mult_block(NBLC, me, NBLC, WA, WB, WC);
            /* Add WC to C */
            for(i=0; i<NBLC; i++){
                for(j=0; j<me; j++) C[(is+i)*n+(ks+j)] += WC[i][j];
            }
        }
        js = NBLC*m;
        /* Copy A to WA */
        for(i=0; i<NBLC; i++){
            for(j=0; j<me; j++) WA[i][j] = A[(is+i)*n+(js+j)];
        }
        for(kloop=0; kloop<m; kloop++){
            ks = NBLC*kloop;
            /* Copy B to WB */
            for(i=0; i<me; i++){
```

```
            for(j=0; j<NBLC; j++) WB[i][j] = B[(js+i)*n+(ks+j)];
        }
        /* Block Matrix Multiplication(WC = WA*WB) */
        mult_block(NBLC, NBLC, me, WA, WB, WC);
        /* Add WC to C */
        for(i=0; i<NBLC; i++){
            for(j=0; j<NBLC; j++) C[(is+i)*n+(ks+j)] += WC[i][j];
        }
    }
    ks = NBLC*m;
    /* Copy B to WB */
    for(i=0; i<me; i++){
        for(j=0; j<me; j++) WB[i][j] = B[(js+i)*n+(ks+j)];
    }
    /* Block Matrix Multiplication(WC = WA*WB) */
    mult_block(NBLC, me, me, WA, WB, WC);
    /* Add WC to C */
    for(i=0; i<NBLC; i++){
        for(j=0; j<me; j++) C[(is+i)*n+(ks+j)] += WC[i][j];
    }
}
is = NBLC*m;
for(jloop=m*id/NTHR; jloop<m*(id+1)/NTHR; jloop++){
    js = NBLC*jloop;
    /* Copy A to WA */
    for(i=0; i<me; i++){
        for(j=0; j<NBLC; j++) WA[i][j] = A[(is+i)*n+(js+j)];
    }
    for(kloop=0; kloop<m; kloop++){
        ks = NBLC*kloop;
        /* Copy B to WB */
        for(i=0; i<NBLC; i++){
            for(j=0; j<NBLC; j++) WB[i][j] = B[(js+i)*n+(ks+j)];
        }
        /* Block Matrix Multiplication(WC = WA*WB) */
        mult_block(me, NBLC, NBLC, WA, WB, WC);
        /* Add WC to C */
        for(i=0; i<me; i++){
            for(j=0; j<NBLC; j++) C[(is+i)*n+(ks+j)] += WC[i][j];
        }
    }
    ks = NBLC*m;
    /* Copy B to WB */
    for(i=0; i<NBLC; i++){
        for(j=0; j<me; j++) WB[i][j] = B[(js+i)*n+(ks+j)];
    }
    /* Block Matrix Multiplication(WC = WA*WB) */
    mult_block(me, me, NBLC, WA, WB, WC);
    /* Add WC to C */
    for(i=0; i<me; i++){
        for(j=0; j<me; j++) C[(is+i)*n+(ks+j)] += WC[i][j];
    }
}
js = NBLC*m;
/* Copy A to WA */
for(i=0; i<me; i++){
    for(j=0; j<me; j++) WA[i][j] = A[(is+i)*n+(js+j)];
}
for(kloop=m*id/NTHR; kloop<m*(id+1)/NTHR; kloop++){
    ks = NBLC*kloop;
    /* Copy B to WB */
```

```
        for(i=0; i<me; i++){
            for(j=0; j<NBLC; j++) WB[i][j] = B[(js+i)*n+(ks+j)];
        }
        /* Block Matrix Multiplication(WC = WA*WB) */
        mult_block(me, NBLC, me, WA, WB, WC);
        /* Add WC to C */
        for(i=0; i<me; i++){
            for(j=0; j<NBLC; j++) C[(is+i)*n+(ks+j)] += WC[i][j];
        }
    }
    ks = NBLC*m;
    /* Copy B to WB */
    for(i=0; i<me; i++){
        for(j=0; j<me; j++) WB[i][j] = B[(js+i)*n+(ks+j)];
    }
    /* Block Matrix Multiplication(WC = WA*WB) */
    mult_block(me, me, me, WA, WB, WC);
    /* Add WC to C */
    for(i=me*id/NTHR; i<me*(id+1)/NTHR; i++){
        for(j=0; j<me; j++) C[(is+i)*n+(ks+j)] += WC[i][j];
    }
    (*(data->RunningThreads))--;
#ifdef _DEBUG
    printf("ID=%d: END runninng=%d\n",id,(*(data->RunningThreads)));
#endif
}

void mult_block(int mi, int mj, int mk,
                double WA[NBLC][NBLC],double WB[NBLC][NBLC],double WC[NBLC][NBLC])
{
    int i, j, k;
    double s;

    /* (j,i,k); Inner Product Version */
    for(j=0; j<mj; j++){
        for(i=0; i<mi; i++){
            s = 0.0;
            for(k=0; k<mk; k++){
                s += WA[i][k]*WB[k][j];
            }
            WC[i][j] = s;
        }
    }
}
```

mltmatst.c　mltmats.c をテストするためのメインプログラム及び諸ルーチン。

元プログラムとの変更点は、プリプロセッサ ディレクティブを利用して、さまざまなテスト・観測を行いやすくした点である。

_MODE_VERIFY が define されていれば定義どおりの行列演算と比較を行い、define されていなければ定義どおりの行列演算を省いて時間を節約する。

DEFAULT_OF_N は定義されていれば、その大きさの行列の演算を行う。定義されていなければ従来どおり標準入力から入力を受け付ける。

NTEST は、ここで指定された回数だけテストを繰り返すというものである。

あとは、最初に define で指定された値や n が表示されたり、最後に音が出るようになっているが、これもテスト・観測の都合である。

```
/* Test Program of Function: multmats */

/* _MODE  */
#undef _MODE_VERIFY
#define DEFAULT_OF_N 1000
#define NTEST 3


#include "dimmats.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define Timer clock()

double A[NMAX][NMAX], B[NMAX][NMAX], C[NMAX][NMAX], D[NMAX][NMAX];

/* Prototype */
int mflops(double flops, double time);
void set_mat(int n, double A[NMAX][NMAX], double B[NMAX][NMAX]);
void multmats0(int n, double A[NMAX][NMAX], double B[NMAX][NMAX], double C[NMAX][NMAX]);
void multmats(int n,double *A0,double *B0,double *C0);
double check(int n, double C[NMAX][NMAX], double D[NMAX][NMAX]);
/* Prototype */

int main()
{
    int n,i;
    double sec, flops;
    clock_t tic, toc;

    printf("NBLC = %d , NTHR = %d",NBLC,NTHR);

#ifndef DEFAULT_OF_N
    printf("\ninput n (<= %d)\n", NMAX);
    printf(" n = "); scanf("%d", &n);
#else
    printf(" , n = %d\n",DEFAULT_OF_N);
    n=DEFAULT_OF_N;
#endif

    flops = 2.0*n*n*n;

#ifdef _MODE_VERIFY
    set_mat(n, A, B);   /* Matrices Setting */
    printf("*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]\n");

    /* Normal(Definition) Matrix Multiplication */
    tic = Timer;
    multmats0(n, A, B, D);
    toc = Timer;
    sec = (double)(toc - tic)/CLOCKS_PER_SEC;
    printf(" definition  %10.2f %13d            ***\n", sec, mflops(flops, sec));
    /* Block Matrix Multiplication */
    tic = Timer; multmats(n,&A[0][0],&B[0][0],&C[0][0]); toc = Timer;
    sec = (double)(toc - tic)/CLOCKS_PER_SEC;
    printf(" multmats    %10.2f %13d %17.1e\n", sec, mflops(flops, sec), check(n, C, D));
#else
    printf("*** Type ***   Time[sec]    Speed[MFLOPS]\n");
    for(i=0;i<NTEST;i++){
```

```
        set_mat(n, A, B);   /* Matrices Setting */
        /* Block Matrix Multiplication */
        tic = Timer; multmats(n,&A[0][0],&B[0][0],&C[0][0]); toc = Timer;
        sec = (double)(toc - tic)/CLOCKS_PER_SEC;
        printf(" multmats    %10.2f %13d\n", sec, mflops(flops, sec));
    }
    printf("\a\n");
#endif

    return 0;
}

int mflops(double flops, double sec)
{
    int mf;

    if (sec > 1.0e-14) mf = (int)(flops/sec/1000000);
    else mf = 0;

    return mf;
}

void set_mat(int n, double A[NMAX][NMAX], double B[NMAX][NMAX])
{
    int i, j;

    srand( (unsigned)time( NULL ) );

    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            A[i][j] = (double)rand()/1000.0;
            B[i][j] = (double)rand()/1000.0;
        }
    }
}

double check(int n, double C[NMAX][NMAX], double D[NMAX][NMAX])
{
    int i, j;
    double temp, max_dif;

    max_dif = 0.0;
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            temp = fabs(C[i][j] - D[i][j]);
            if(temp > max_dif) max_dif = temp;
        }
    }

    return max_dif;
}

void multmats0(int n, double A[NMAX][NMAX], double B[NMAX][NMAX], double C[NMAX][NMAX])
{
    int i, j, k;

    /* Initialization */
    for(i=0; i<n; i++){
        for(j=0; j<n; j++) C[i][j] = 0.0;
    }
```

```
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            for(k=0; k<n; k++){
                C[i][j] += A[i][k]*B[k][j];
            }
        }
    }
}
```

5.2 結果

精度

```
NBLC = 55 , NTHR = 2 , n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]    Check[Error]
 definition       44.23            45            ***
 multmats          6.63           301          1.2e-009
```

ここでいう Error は本当の誤差ではなく、definition に従った計算と比較した数値にすぎないが、どちらにせよ特に問題は無い範囲といえる。

広範囲で NBLC,NTHR を変更した時の速度

#define NTEST 3 として行ったテストの平均である。最速を求めるのが目的であるので、これ以外の部分のテストはとりあえず省いた。

NBLC \ NTHR	1	2	3	4	7	8
40				7.03		
45				6.97		
50		7.05	6.89	6.61		
55	12.66	6.57	6.72	7.00	7.09	7.33
60		6.69	6.95			
65		6.97				
70		6.76				
75		7.22				
80		7.08				
90		7.67		11.18		10.72

最速値の追及

上の結果から、最速値は NBLC=50〜65,NTHR=2 にあると考えられる。もう一度全て実行しなおしてやってみる。

NBLC	50	51	52	53	54	55	56	57	58	59	60
1	6.61	6.89	6.77	6.41	6.64	6.58	6.81	7.17	6.88	6.69	6.44
2	6.55	6.77	6.67	6.45	6.55	6.52	6.72	7.08	6.75	6.55	6.42
3	6.50	6.75	6.69	6.42	6.55	6.45	6.69	7.08	6.72	6.56	6.44
Ave.	6.55	6.80	6.71	6.43	6.58	6.52	6.74	7.11	6.78	6.60	6.43

NBLC	61	62	63	64	65

```
----+-----+-----+-----+-----+-----
1   | 6.56| 6.78| 7.44|10.69| 7.03
2   | 6.44| 6.64| 7.03|10.64| 7.03
3   | 6.41| 6.66| 7.05|10.66| 7.06
----+-----+-----+-----+-----+-----
Ave.| 6.47| 6.69| 7.17|10.66| 7.04
```

最速値の測定

上の表より、NBLC=53 とした。60 の時は広範囲テストでなぜか成績が悪かったので避ける事にした。

```
G:\repo2\MLTMAT~2\Release>mltmatst
NBLC = 53 , NTHR = 2 , n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]
 multmats          6.48           308
 multmats          6.41           312
 multmats          6.42           311
 multmats          6.44           310
 multmats          6.42           311
 multmats          6.44           310
 multmats          6.41           312
 multmats          6.42           311
 multmats          6.41           312
 multmats          6.41           312

G:\repo2\MLTMAT~2\Release>
```

よって、6.43 秒

6 おまけ

```
G:\repo2\MLTMAT~2\Release>mltmatst
NBLC = 70 , NTHR = 2 , n = 1000
*** Type ***   Time[sec]    Speed[MFLOPS]
 multmats          5.41           369
 multmats          5.36           373
 multmats          5.36           373
G:\repo2\MLTMAT~2\Release>
```

ちょっとした反則をつかってみた。

```
#define double float
```

を入れてみたら、この結果が出た。精度を落とせば速度が上がる、あるいはキャッシュに入る量が増える、ということだろうか。

7 考察

7.1 オリジナル版

NBLC=95 が最速である事は間違いないが、NBLC=96 の時、19 秒台ととても遅くなっている。何度試しても 19 秒台だったのでそういうものなのだろうが、原因は分からない。他のところでも、

ところどころ速かったり遅かったりで謎が多い。なお、NBLC によるばらつきの方が同じ NBLC でのばらつきより明らかに大きかったので、試行回数が少ないのが原因ではない。

このような実験的立場からの高速化もできるが、理論的立場からの高速化を考えると、パイプラインやキャッシュの仕様やアルゴリズムまで踏み込んで調べることになる。しかし、パイプラインやキャッシュの仕様やアルゴリズムを知っても、C 言語でそれを使うにはコンパイラがどんな機械語命令を生成するか予測する必要がある。これも実験的に調べるか、コンパイラのソースを調べるかしないと分からないことである。実験的に調べたら理論的立場からの高速化でなくなるので、コンパイラのソースを調べることになる。

以上より　授業後に質問した、コンパイラのソースから機械語命令を予測する手間をかけるより、直接アセンブリ言語から書いた方が効率的かつ理論的、あるいは科学的なのではないか（質問した時は単に「アセンブリ言語から書いた方がいい」と言ったと思います）、ということなのですが、やはり納得できない点が残ります。

それは、このレベルでの速度の追及はアーキテクチャをフルに活かし切る事であるわけだから、移植性などとは矛盾があるのではないか、とのことと、実際にテストデータをもらって、実行速度を自分で計測して自動的にループアンローリングやループ変換を行うコンパイラが作れるのではないか、ということです。

あるいは、自動化できなくても、以下のように書けると美しく速いと思うのですが、どうでしょう？

```
#define iMAX 4
unlooped for(i=0;i=iMAX;i++){
    someroutine(i);
}
```

7.2　マルチスレッド版

広範囲テストの結果　最大速度は予想通りマルチスレッド化で 2 倍弱になるようだ。スレッドの本数が 4 本程度までなら効率はさして変わらない。奇数の 3 の時ですら意外と変わらなかったのでちょっと驚いた。CPU は 2 つなのにである。Windows2000 のスケジューリングの能力だろうか。

一方、最良な NBLC はスレッド数が増えるほど小さくなるようだ。これは、一スレッドあたりが使うべきキャッシュの量の問題だろう。

ここには表は掲載していないが、NBLC を増やしすぎると、遅くなる一方で数値にばらつきが出ることも確かめている。例えば、NBLC=90,NTHR=4 の時、14.88, 8.89, 9.77 であった。キャッシュの入りきらなくなって、ヒット率が不安定になるのであろう。また、全体に 1 回目が遅いのはプログラムキャッシュなどが考えられる。

参考文献

[1] 荻田武史: 行列乗算高速化プロジェクト
http://www.geocities.co.jp/Technopolis/6872/math/multmat.htm

[2] 山下 淳: C 言語を学習する上でよく聞かれる質問集
http://www.jks.is.tsukuba.ac.jp/free/students/junkun/FAQ/C/

[3] 粂井康孝: 猫でもわかるプログラミング
http://www.kumei.ne.jp/c_lang/

[4] MSDN ライブラリ