Cornell University

Department of Physics

Phys 480/680, Astro 690 March 31, 2002

Computational Physics, Spring 2002

Homework Assignment # 7

(Due Tuesday, April 9 at 8:40 am in Computer Lab.)

Agenda and readings for the weeks of April 2 & 9:

Goals: Test timings of BLAS operations and blocked matrix multiplies; gain experience calling LAPACK.

Readings marked NR are from *Numerical Recipes: The Art of Scientific Computing*, 2nd edition (in C). Readings marked LN are from the course lecture notes to be found at http://www.ccmr.cornell.edu/~muchomas/P480.

Recap of last week:

- Lec16, 03/26 (Tue: mini-lecture, 25mins): Optimization — Amdahl's law, inner loop optimization, rates of common operations
- Lab5, 03/26 (Tue): Full, self-consistent solutions for density functional theory of atoms; profiling and practical optimization.
- Lec17, 03/28 (Thu): Cache optimization; BLAS1-3; blocked matrix multiply; "Expressive software" concept.

Preview of coming two weeks:

- Lec18, 04/02 (Tue: mini-lecture, 25mins): Solution of Poisson's equation in $d > 1$ dimension in a single line of expressive software!
- Lab6, 04/02 (Tue): Fully optimized atomic software, FLOP rates of common operations
- Lec19, 04/04 (Thu): Implemenation of expressive software for Poisson's equation: planewave basis and coding in C++
- Lec20, 04/09 (Tue: mini-lecture, 25mins): C++ continued.
- Lab7, 04/09 (Tue): Timing computer FLOP performance for BLAS/LAPACK operations

Contents

1 Basic Linear Algebra Subroutines (BLAS) 3

1.1 BLAS2 3

1.2 BLAS3 3

2 Linear Algebra Package (LAPACK) 4

A "lapack.c" **5**

B "lapack.c" for Windows **7**

C LAPACK **10**

1 Basic Linear Algebra Subroutines (BLAS)

This problem develops experience using cache in the context of basic linear algebraic subroutines (BLAS).

1.1 BLAS2

As a typical BLAS2 operation, we here consider vector-matrix multiplication (the example from lecture),

$$v_2[i] = \sum_k v_1[j] M_1[j][i]. \tag{1}$$

To assess the importance of the ordering of the loops, write the following routines (same prototype as mult()), uncomment their calls through timer in the main program in "timer.c" from Homework Assignment # 6, comment out preceding calls to timer, and run the resulting program.

1. vecmat1(): code the multiplication (1) using the naïve sequence of loops (first i, then j)
2. vecmat2(): code the multiplication (1) reversing the sequence of the loops

1.2 BLAS3

The BLAS3 routines, which involve n^3 operations (and only n^2 data) are where the great gains from cache may be had. Write the routines below (same prototype as mult()), and uncomment the rest of the main program. Note that, for debugging, the main program checks your two matrix-matrix multiplications routines against each other. This is useful because block-matrix multiply is somewhat intricate and prone to coding bugs.

1. matmat1(): standard implementation of the matrix multiplication

$$M_3[i][j] = \sum_{k=1}^{n} M_1[i][k] M_2[k][j]$$

2. matmat3(): matrix multiplication $M_3 = M_1 M_2$ via blocks of size Nb×Nb, where Nb is set in a #define statement. (Nb=32 is a reasonable number.)

   Note: To simplify the exercise, you may assume that Nb divides the total matrix size evenly.

Hint: Pseudo-code for the block-matrix multiplication follows:

```
double b1[Nb][Nb],b2[Nb][Nb],b3[Nb][Nb]; /* Regular C arrays are faster
                                             -- use them! */

for (I=0; I<n/Nb; I++)
  for (J=0; J<n/Nb; J++) { /* Compute IJ block of M3: */


    -> YOUR CODE: zero out b3, an Nb x Nb array to store IJ block of M3

    for (K=0; K<n/Nb; K++) { /* Add to b3 product of IK block of M1
                                with KJ block of M2 */
      /* Copy IK block of M1 into b1, an Nb x Nb matrix */
      for (i=0; i<Nb; i++)
        for (k=0; k<Nb; k++)
         b1[i][k]=M1[I*Nb+i][K*Nb+k];
```

```
      /* Copy KJ block of M2 into b2, an Nb x Nb matrix */
      -> YOUR CODE

      /* Do b1*b2, adding result into b3 */
      for (i=0; i<Nb; i++)
        for (j=0; j<Nb; j++)
          for (k=0; k<Nb; k++)
            b3[i][j]+=b1[i][k]*b2[k][j];
    }
  -> YOUR CODE: Copy b3 into IJ block of M3
  }
}
```

2 Linear Algebra Package (LAPACK)

This problem demonstrates the power of a highly optimized package such as LAPACK over *Numerical Recipes*. Compile and run the main program in Appendix A(Appendix B if you are using Windows), which compares the MFLOPS rates of Numerical Recipes routines ludcmpp480() and lubksbp480() for solving a linear system of N=512 equations in 512 unknown against the LAPACK routine dgesv(). The program also checks that the equations are solved correctly. You may wish to set the problem size N=512 (#define at top of code) to a smaller number while you are debugging.

Because LAPACK is written in Fortran 77, much of the work in calling LAPACK is to get the code to compile. You will note, for instance, that because Fortran 77 accepts all arguments as pointers to the corresponding data, the arguments in the call to dgesv() each represent the *address* of the first element of a set of data. (The &'s to the left of each argument accomplish this.)

Because of the above subtleties, Appendix A/B gives a completed program for you. You still likely will have to spend some time hacking with the code to get it to compile. Compilers on UNIX systems, for instance, usually put underscore characters ('_') to the right of the names of Fortran 77 routines (as we have done in the appendix). Your system may do something different and you may have to experiment with moving the '_' or, perhaps, eliminating it!

If you cannot find the LAPACK or BLAS libraries for your machine, Appendix C gives links to the P480 home page where you can down-load C code for the LAPACK and BLAS routines that you will need, and some hints for compiling. (Thanks to Nick Bailey, the grader for Spring 2000, for getting this for you!) Note that the code will run a lot faster if you can use the vendor-supplied BLAS provided there for Pentium/Linux machines.

Finally, if you are compiling under Linux, something like the following compile statement (all on one command line!) will *probably* get the code to compile:

```
cc -O3 -fomit-frame-pointer lapack.c p480.o nrutil.o -o lapack -L... -llapack -lblas -lf2c -lm
```

where "..." represents the full path name of the directory where the "liblapack.a" and "libblas.a "files appears. Don't hesitate to ask for help in getting this code to compile!

Important Note: To use LAPACK from C in real applications, it is important to keep in mind that LAPACK is written in Fortran and expects matrices to be stored in memory in the opposite order from C. This means that, *if you start in C, you should send LAPACK the transposes of any matrices (and expect to get transposes back).* For the test program in this example, we work with symmetric matrices — so this transposing isn't needed!

A "lapack.c"

```
#include <stdio.h>
#include <sys/time.h>
#include "nrutil.h"
#include <math.h>
#define N 512

main()
{
  /* Timer stuff */
  struct timeval time1,time2;
  double rtime,mflop;

  /* Stuff for LAPACK call */
  int NL=N,NRHS=1,LDA=N,*IPIV,LDB=N,INFO;

  /* Declare arrays and variables */
  double **M1,*v1;
  double **M2,*v2;
  double **M3,*v3;

  double tmp,error;
  int *indx;
  int i,j;

  M1=dmatrix(1,N,1,N);
  M2=dmatrix(1,N,1,N);
  M3=dmatrix(1,N,1,N);

  v1=dvector(1,N);
  v2=dvector(1,N);
  v3=dvector(1,N);

  indx=ivector(1,N);
  IPIV=ivector(1,N);

  /* Matrices and vectors */
  for (i=1; i<=N; i++)
    for (j=1; j<=N; j++) {
      M1[i][j]=1./(1+i+j);
      M2[i][j]=1./(1+i+j);
      M3[i][j]=1./(1+i+j);
    }
  for (i=1; i<=N; i++) {
    v1[i]=1./(1+i);
    v2[i]=1./(1+i);
    v3[i]=1./(1+i);
  }

  /* Numerical Recipes */
  gettimeofday(&time1,NULL);
  ludcmpp480(M1,N,indx,&tmp); /* Get LU decomposition */
```

```
  lubksbp480(M1,N,indx,v1); /* Use LU decomposition to solve equations */
  gettimeofday(&time2,NULL);

  /* Output timing */
  mflop=(double) 2*N*N*N/3./1e6;
  rtime=
    (time2.tv_sec-time1.tv_sec)+(time2.tv_usec-time1.tv_usec)/1.e6;

  printf("\n");
  printf("Numrec:  %.0lf MFLOP in %.3f sec ==> %.0lf MFLOPS.\n",
  mflop,rtime,mflop/rtime);

  /* LAPACK */
  gettimeofday(&time1,NULL);
  dgesv_(&NL,&NRHS,&M2[1][1],&LDA,&IPIV[1],&v2[1],&LDB,&INFO);
  gettimeofday(&time2,NULL);
  if (INFO!=0) {
    printf("Error in LAPACK call: %d",INFO);
    exit(1);
  }

  /* Output timing info */
  mflop=(double) 2*N*N*N/3./1e6;
  rtime=
    (time2.tv_sec-time1.tv_sec)+(time2.tv_usec-time1.tv_usec)/1.e6;
  printf("LAPACK:  %.0lf MFLOP in %.3f sec ==> %.0lf MFLOPS.\n",
  mflop,rtime,mflop/rtime);

  /* Check results */
  printf("\n");
  for (i=1; i<=N; i++) {
    tmp=-v3[i];
    for (j=1; j<=N; j++)
      tmp+=M3[i][j]*v1[j];
    error+=tmp*tmp;
  }
  printf("Numerical Recipes rms error: %e\n",sqrt(error/N));

  for (i=1; i<=N; i++) {
    tmp=-v3[i];
    for (j=1; j<=N; j++)
      tmp+=M3[i][j]*v2[j];
    error+=tmp*tmp;
  }
  printf("LAPACK            rms error: %e\n",sqrt(error/N));
  printf("\n");
}

MAIN__()
{
  printf("Who am I?  And, why am I here?\n");
  exit(1);
}
```

B "lapack.c" for Windows

```
#include <stdio.h>
#include <sys/timeb.h>
#include "nrutil.h"
#include "p480.h"
#include <math.h>
#define N 512


main()
{
  /* Timer stuff */
  struct _timeb time1,time2;
  double rtime,mflop;

  /* Stuff for LAPACK call */
  int NL=N,NRHS=1,LDA=N,*IPIV,LDB=N,INFO;

  /* Declare arrays and variables */
  double **M1,*v1;
  double **M2,*v2;
  double **M3,*v3;

  double tmp,error=0;
  int *indx;
  int i,j;

  M1=dmatrix(1,N,1,N);
  M2=dmatrix(1,N,1,N);
  M3=dmatrix(1,N,1,N);

  v1=dvector(1,N);
  v2=dvector(1,N);
  v3=dvector(1,N);

  indx=ivector(1,N);
  IPIV=ivector(1,N);

  /* Matrices and vectors */
  for (i=1; i<=N; i++)
    for (j=1; j<=N; j++) {
      M1[i][j]=1./(1+i+j);
      M2[i][j]=1./(1+i+j);
      M3[i][j]=1./(1+i+j);
    }
  for (i=1; i<=N; i++) {
    v1[i]=1./(1+i);
    v2[i]=1./(1+i);
    v3[i]=1./(1+i);
  }

  /* Numerical Recipes */
```

```
  _ftime(&time1);
  ludcmpp480(M1,N,indx,&tmp); /* Get LU decomposition */
  lubksbp480(M1,N,indx,v1); /* Use LU decomposition to solve equations */
  _ftime(&time2);


  /* Output timing */
  mflop=(double) 2*N*N*N/3./1e6;
  rtime=
    (time2.time-time1.time)+(time2.millitm-time1.millitm)/1.e6;

  printf("\n");
  printf("Numrec:  %.0lf MFLOP in %.3f sec ==> %.0lf MFLOPS.\n",
          mflop,rtime,mflop/rtime);

  /* LAPACK */
  _ftime(&time1);
  //dgesv_(&NL,&NRHS,&M2[1][1],&LDA,&IPIV[1],&v2[1],&LDB,&INFO);
  _ftime(&time2);
  /*if (INFO!=0) {
    printf("Error in LAPACK call: %d",INFO);
    exit(1);
  }*/

  /* Output timing info */
  mflop=(double) 2*N*N*N/3./1e6;
  rtime=
    (time2.time-time1.time)+(time2.millitm-time1.millitm)/1.e3;
  printf("LAPACK:  %.0lf MFLOP in %.3f sec ==> %.0lf MFLOPS.\n",
          mflop,rtime,mflop/rtime);

  /* Check results */
  printf("\n");
  for (i=1; i<=N; i++) {
    tmp=-v3[i];
    for (j=1; j<=N; j++)
      tmp+=M3[i][j]*v1[j];
    error+=tmp*tmp;
  }
  printf("Numerical Recipes rms error: %e\n",sqrt(error/N));

  for (i=1; i<=N; i++) {
    tmp=-v3[i];
    for (j=1; j<=N; j++)
      tmp+=M3[i][j]*v2[j];
    error+=tmp*tmp;
  }
  printf("LAPACK            rms error: %e\n",sqrt(error/N));
  printf("\n");
}

MAIN__()
{
```

```
  printf("Who am I?  And, why am I here?\n");
  exit(1);
}
```

C LAPACK

If you cannot locate the LAPACK library on your machine, please see
http://www.ccmr.cornell.edu/~muchomas/P480/LAPACK for instructions as to how to proceed.

Please let us (muchomas@ccmr.cornell.edu, freedman@physics.cornell.edu) know if you have any difficulties!