COMP-40730 HPC

Report for Assignment 1

Author:	Paula Dwan
Due date:	30-June-2014
Lecturer:	Alexey Lastovetsky
Subject:	COMP-40730 High Performance Computing
College:	University College Dublin

Contents

Exercise

Write C programs implementing the following three algorithms of multiplication of two n×n dense matrices:

1) Straightforward non-blocked IJK algorithm.
2) Blocked IJK algorithm using square b×b blocks.
3) Blocked KIJ algorithm using square b×b blocks.

Experiment with the programs and build/plot:

1) The dependence of the execution time of each program on the matrix size n and the block size b .
2) The speedup of the Blocked algorithms over the non-blocked one as a function of the matrix size and the block size.
3) Compare the fastest program with the BLAS/ATLAS routine **dgemm** implementing the same operation.

Explain the results.

Variants of the assignment:

1) Multiplication of matrix blocks in the implementation of the Blocked IJK algorithm:
   a. manually written
   b. BLAS calls
   c. ATLAS calls
2) Multiplication of matrix blocks in the implementation of the Blocked KIJ algorithm:
   a. manually written
   b. BLAS calls
   c. ATLAS calls
3) Comparison with BLAS/ATLAS routine;
   a. BLAS
   b. ATLAS

Observations

I had started working on the three different implementations and decided to complete each in order to learn better the theory behind the assignments. I also used three separate programs to complete each section.

Overview of Computations Obtained and How

Assignment 1 basically involved (for me) writing three programs :

1. **A1-Sijk-1D.c**

   Implementation of a straight forward matrix nxn multiplication using both a manual (dot product calculation) and BLAS application.

   Algorithm

   no additional variable

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

   This was calculated using a temporary variable to reduce the number of pointers in use and thus improve performance and also without for comparison.

   This is due to the fact that the variable value is held in

   Algorithm

   using additional variable

   ```
   for (i=0; i<rows; i++)
   {
       for (j=0; j<cols; j++)
       {
           double sum = 0.0;
           for (k=0; k<rows; k++)
           {
               sum+= (A [i][k]) * (B [k][j]);
           }
           C [i][j] = sum;
       }
   }
   ```

2. **A1-Bijk-1D.c**

   Blocked IJK; again I calculated the results using both manual and BLAS computations.

   Blocked IJK; this was calculated the results using both manual (complex : six loops and simplified : three loops) and BLAS computations.

   So I had two types of Blocked KIJ : which theoretically improved performance by applying temporal improvement using blocks. Blocked IJK was implemented using simplified algorithm of three inner loops and adding block size to each and complex algorithm of six inner loops.

   There was only one type of Straightforward KIJ calculated, and an additional variable was used to reduce the number of pointers thus again theoretically improving performance. Finally, calculations were completed using BLAS/ATLAS for comparison.

   Algorithm

   Complex

   ```
   for (bi=0; bi<rows; bi+=block)
   {
       for (bj=0; bj<cols; bj+=block)
       {
           for (bk=0; bk<rows; bk+=block)
           {
               for (ni=0; ni<block; ni++)
               {
                   for (nj=0; nj<block; nj++)
                   {
                       double sum = 0.0;
                       for (nk=0; nk<block; nk++)
                       {
                           sum += A [bi+ni][bk+nk] * B [bk+nk][bj+nj];
                       }
                       C [bi+ni][bj+nj] = sum;
                   }
               }
           }
       }
   }
   ```

Simplified IJK – included for comparison

Algorithm
Simplified

```
for (ni=0; ni<(n/b); ni++)
{
    for (nj=0; nj<(n/b); nj++)
    {
        double sum = 0.0;
        for (nk=0; nk<(n/b); nk++)
        {
            sum += A [ni][nk] * B [nk][nj];
        }
        C [ni][nj] = sum;
    }
}
```

3. **A1-Bkij-1D.c**

Blocked KIJ; again I calculated the results using both manual (complex : six loops and simplified : three loops) and BLAS computations.

So I had two types of Blocked KIJ : which theoretically improved performance by applying temporal improvement using blocks. Blocked KIJ which was implemented using simplified algorithm of three inner loops and adding block size to each.

There was only one type of Straightforward KIJ calculated, and an additional variable was used to reduce the number of pointers thus again theoretically improving performance. Finally, calculations were completed using BLAS/ATLAS for comparison.

Algorithm
Complex

```
for (bk=0; bk<rows; bk+=block)
{
    for (bi=0; bi<cols; bi+=block)
    {
        for (bj=0; bj<rows; bj+=block)
          {
            for (nk=0; nk<block; nk++)
            {
                for (ni=0; ni<block; ni++)
                {
                    double sum = A [bi+ni][bk+nk];
                    for (nj=0; nj<block; nj++)
                    {
                        C [bi+ni][bj+nj] += sum * B [bk+nk][bj+nj];
                    }
                }
            }
        }
    }
}
```

Simplified Blocked KIJ – included for comparision

Algorithm
Simplified

```
for (nk=0; nk<(n/b); nj++)
{
    for (ni=0; ni<(n/b); ni++)
    {
        double sum = B [ni][nk];
        for (nj=0; nj<(n/b); nj++)
        {
            C [ni][nj] += sum * B[nk][nj];
        }
    }
}
```

Assignment Execution

Each program was executed multiple times using the script *./runAssignment1.sh*. This has multiple options and the syntax and usage follows:

```
USAGE : ./runAssignment1.sh -a|--all -1|--simple -2|--ijk -3|--kij -r|--random -i|--increment \
                    -m|--matrix<n> -b|--block <b> -v|--values -?|-h|--help
  where:
  -a | --all :        calculate data for all algorithms via separate .c programs to multiply
                      |A|x|B| -> |C|
                      Straightforward non-blocked ijk algorithm :     A1-Sijk-1D.c
                      Blocked ijk algorithm using square bxb blocks : A1-Bijk-1D.c
                      Blocked kij algorithm using square bxb blocks : A1-Bkij-1D.c
  -1|--simple :       calculate data for only the algorithm
                      Straightforward non-blocked ijk algorithm :     A1-Sijk-1D.c
  -2|--bijk :         calculate data for only the algorithm
                      Blocked ijk algorithm using square bxb blocks : A1-Bijk-1D.c
  -3|--bkij :         calculate data for only the algorithm
                      Blocked kij algorithm using square bxb blocks : A1-Bkij-1D.c

  -r|--random :       initialization A| & |B| with random numbers and |C| with '0'
  -i|--increment :    initialize |A| & |B| incrementally with <row> value and |C| with '0'
                      '-i|--increment' & '-r|--random' are mutually exclusive.

  -m|--matrix <n> :   matrix dimension, set to maximum of '1,000' if invalid or not provided
  -b|--block <b> :    block size, with '<nb> < <nx>' and '<nx> ÷ <nb> = zero',
                      defaults to '50' if invalid or not provided and <nx> will be reset to 1,000.
                      Mutually exclusive with '-v|--values'

  -v|--values :       use predefined range of valid values for <nx> and <nb> as follows :
                      <nx> = { 50, 50, 50, 100, 100, 100, 500, 500, 500, 1000, 1000, 1000 }
                      <nb> = { 2, 5, 10, 5, 10, 20, 10, 20, 50, 10, 50, 100 }
                      Mutually exclusive with '-m|--matrix <n>' and '-b|--block <b>'

  -?|-h|--help :      usage
```

Execute this script in the home directory of Assignment 1, retaining the overall directory structure when unzipping.

Data Results Obtained

Data text files suitable containing the values of the computation used for matrices |A| and |B| and the results stored in |C| are saved in the sub-directory /Results. File naming convention is :

<data log file name>	`pdwan-<A1.Sijk-|A1.Bijk-|A1.Skij->-<current time>-values<iteration>.txt`
example:	`pdwan-A1.Sijk-20140628-022426-values-0.txt`

Note that if predefined range of valid values is not used then there is only one iteration. Single iteration also applies where the user enters arbitrary, valid values for matrix size and block size for each individual .c program and does not use the scripts

. *(This was mainly used to validate the results using a 10x10 matrix.)*

Results Evaluated

A summary file containing processing time for each computation (manual and BLAS) for each selected algorithm is also saved. This is in a format suitable for us with GNUplot.

<timing log file name>	`pdwan-<A1.Sijk-| A1.Bijk-	A1.Skij->-timing<iteration>.dat`
example:	`pdwan-A1.Sijk-20140628-022426-timing-0.dat`

I wished to keep each .c program as clean as possible and so all production setup was completed in the script for each assignment. Thus file creation and validation for each iteration was completed before the .c program was even called. Simple validation of the arguments passed to each .c program is also completed.

I also spot-checked the results for simple IJK computation : manual, BLAS and ATLAS. Results obtained are detailed in Appendix I – Validate Results.

GNUplot Execution

I followed the same convention for each .dat file as produced by each .c program, an example follows :

Sample .dat file

```
# Program running :    A1-Bijk-1Dmatrix.c
# File name is:        pdwan-A1.Bijk-20140628-022426-timing-0.dat
#
# Computation of Blocked IJK using square bxb block for matrices
# |A| & |B| storing results in |C|
#
# matrix size         block size      Time/manual      Time / dgemm
50                    2
50                    5
50                    10
100                   5
100                   10
100                   20
500                   10
500                   20
500                   50
1000                  10
1000                  50
1000                  100
```

Each was then presented in graphical format using GNUplot, comparing times taken for manual and for BLAS/ATLAS computations. A generic GNUplot program was written to output the data to the screen.

Sample GNUplot program execution

```
# To execute, launch GNUpopt and run :
# gnuplot> load <filename.gp>
# making sure that the data file name used is updated if needed.

unset log
unset label
set xtic auto
set ytic auto
set grid
set title “Comparison of time taken for manual and dgemm computation \n
    for matrix size and block size”
set xlabel “Time taken / ms”
set ylabel “size of block / matrix”
plot
    'pdwan-A1.Bijk-20140628-022426-timing-0.dat'
        u 1:3 t 'Matrix : manual' w l lw 0.5 lc rgb 'blue'

    'pdwan-A1.Bijk-20140628-022426-timing-0.dat'
        u 2:3 t 'Block: manual' w l lw 0.5 lc rgb 'green'

    'pdwan-A1.Bijk-20140628-022426-timing-0.dat'
        u 1:4 t 'Matrix : dgemm' w l lw 0.5 lc rgb 'black'

    'pdwan-A1.Bijk-20140628-022426-timing-0.dat'
        u 2:4 t 'Block: manual' w l lw 0.5 lc rgb 'red'
```

Thankfully for linux (Ubuntu) – I could install and run GNUplot locally.

Screen shots of each were taken and added to the sections GNUplot graphs - Straight-forward IJK, GNUplot graphs - Blocked IJK and GNUplot graphs - Blocked KIJ.

Summary Results : Part 1B – BLAS Calls

Build/plot:	The dependence of the execution time of each program on the matrix size n and the block size b.
Variant :	*Multiplication of matrix blocks in the implementation of the Blocked IJK algorithm using BLASS calls.*

GNUplot Graphs

Conclusions

Summary Results : Part 2A – Manually Calculated

Build/plot:	The speedup of the Blocked algorithms over the non-blocked one as a function of the matrix size and the block size.
Variant :	*Multiplication of matrix blocks in the implementation of the Blocked IJK algorithm, which is written manually.* *I also ran the same values for Straight-forward KIJ and for BLAS for comparison of speed for each.*

GNUplot Graphs

Conclusions

Summary Results : Part 3B ATLAS

Build/plot:	Compare the fastest program with the BLAS/ATLAS routine dgemm implementing the same operation.
Variant :	*Comparison with BLAS/ATLAS routine – use ATLAS routine.*

GNUplot Graphs

Conclusions

Appendix I – Validate Results

Spot check only using 10x10 matrix for random number generation to initialize matrices |A| and |B| and also row values for each.