COMP-40730 HPC

REPORT FOR ASSIGNMENT 2

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

CONTENTS

EXERCISE

Write a parallel Pthreads program computing the norm of the product of two n×n dense matrices on a p-processor SMP so that

- p threads are involved in the parallel computations.
- The 1-dimensional parallel algorithm of matrix multiplication is employed:
  - one of matrices is partitioned in one dimension into p equal slices
  - there is one-to-one mapping between the partitions and threads
  - each thread is responsible for computation of the corresponding slice of the resulting matrix
- Computation of the norm of the resulting matrix employs the mutex synchronization mechanism.

You can use BLAS or ATLAS for local computations.

Experiment with the program and build:

- The dependence of the execution time of the program on the matrix size n.
- The speedup over a serial counterpart of the program.

Explain the results.

Variants of the assignment:

1. Granularity of the program:
   (a) Two successive steps:
      i. Parallel matrix multiplication
      ii. Parallel computation of the norm of the resulting matrix
   (b) One-step algorithm. No intermediate resulting matrix.

2. Partitioning scheme:
   (a) Left matrix is horizontally partitioned
   (b) Right matrix is vertically partitioned

3. Matrix norm to be computed:
   (a) The maximum absolute column sum norm (aka one-norm):

$$\|A\|_1 = \max_{0 \le j < n} \sum_{i=0}^{n-1} |a_{ij}|$$

   (b) The maximum absolute row sum norm (aka infinity-norm):

$$\|A\|_\infty = \max_{0 \le i < n} \sum_{j=0}^{n-1} |a_{ij}|$$

Overview of Computations Obtained and How

Assignment 2 basically involved (for me) writing one programs :

1. **manual straight-forward IJK computation**

   Implementation of a straight forward matrix nxn multiplication :

Code

```
for (ni=0 ; ni<rows ; ni++)
{
   for (nj=0 ; nj<cols ; nj++)
   {
      double sum = 0.0 ;
      for (nk=0 ; nk<rows ; nk++)
      {
         sum+= (matrix_a[(ni*rows)+nk]) * (matrix_b[(nk*rows)+nj]) ;
      }
      matrix_c[(ni*rows)+nj] = sum ;
   }
}
```

2. **Dgemm for straight-forward IJK computation**

   Blocked IJK : This was calculated using manual algorithm blocked IJK and DGEMM. Also matrix |C| was evaluated using a straight-forward IJK/KIJ algorithm in order to best compare the time taken.

   So I had two types of Blocked KIJ : manual and DGEMM as well as the simplified straight-forward calculation. Blocked IJK was implemented using simplified algorithm of three inner loops and adding block size to each.

Code

```
   int ni, nj ;
// m, n, k :    local integers indicating the size of the matrices for
// rows x columns :: A :  m x  k, B :  k x n, C:  m x n
// Here, m = n = k = rows = columns = <nx> = <ny> as supplied
   int lm = rows, ln = rows ;
// la_offset, lb_offset, lc_offset :
// Leading dimension of matrix A, B or C respectively, or the number of elements
// between successive rows for row-major storage or columns for column-major
// storage.
   int la_offset = rows, lb_offset = cols, lc_offset = rows ;
   int ALPHA=1.0 ;
   int BETA=0.0 ;

   cblas_dgemm( CblasRowMajor, CblasNoTrans, CblasNoTrans, lm, ln, ln, ALPHA, \
      matrix_a, la_offset, matrix_b, lb_offset, BETA, matrix_c, lc_offset) ;
```

3. **Pthreads calclation**

   This was completed using a data structure, creating a new thread to correspond to the each slice. Infinity norm was mutes'ed to ensure only one thread could access it for updates at any one time.

Code

For options 2. and 3., the matrix |C| was calculated (cblas by default otherwise the user could decide to re-build and execute using cblas or atlas).

Thus results were obtained for each optionm 1., 2. and 3. using the same source |A| and |B| to ensure like was compared to like.

Multiple implementation of each .c program was enabled using ***./runAssignment2.sh,*** for example :

```
$ ./runAssignment2.sh –a –i –v
```

This runs all three algorithm c programs using cblas for incremental column values, using predefined settings for matrix and block sizes for each implementation.

Sample CLI output follows :

Single implementation was completed using **A2-pthreads-1D.c.**

This is compiled using *gcc* for atlas and cblas, as follows :

```
gcc  -I/home/cs/khasanov/libs/CBLAS/src A2-pthreads-1D.c -o A2-pthreads-1D-cblas
/home/cs/khasanov/libs/cblas_LINUX.a  /usr/lib/libblas.a -lgfortran

gcc -o A2-pthreads-1D-atlas A2-pthreads-1D.c -I/home/cs/khasanov/libs/ATLAS/include/
-L/home/cs/khasanov/libs/ATLAS/lib/Linux_UNKNOWNSSE2_4/ -lcblas -latlas -lm -O3
```

ASSIGNMENT EXECUTION

The compiled .c program ./A2-pthreads-1D was executed multiple times standalone or using the script *./runAssignment2.sh* to obtain as wide a range of time taken to calculate |C| using each algorithm.

This has multiple options and the syntax and usage follows :

Execute this script in the home directory of Assignment 2.

Sample execution follows for :

```
$ ./runAssignment2.sh -1 -a -v
```

Note: Please retain the overall directory structure when unzipping.

Note that the script `./runAssignment2.sh` allows two types of implementation

- Multiple iteration : use the switch <-v|--values>, when a predefined range applies for [N] : matrix size and [T] : number of threads applicable.
- Single iteration : use the switch <-m|--matrix> [N] where the user specifies the values for [N] : matrix size and [T] : number of threads applicable.

RUNNING A2-PTHREADS-1D-<ATLAS | CBLAS> *: STANDALONE*

The compiled .c program may also be run standalone. Usage and sample execution follows :

LOG FILES 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 appropriate log files. File naming convention via the script is :

<data log file name>	`Values-<time>-<A2-pthreads-1D>-<iteration>.txt`
example:	`Values-20140715.170928-A2-pthreads-1D-0.txt`

Single iteration also applies where the user enters arbitrary, valid values for matrix size and does not use the scripts and the other required parameters. Each new matrices |A| and |B| and the results in |C| were saved to the data file, thus simple validation using *LibreOffice Calc*.

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

<timing log file name>	`Data-<time>-<A2-pthreads-1D>.dat`
example:	`Data-20140715.171337-A2-pthreads-1D.dat` `Data-20140715.172124-A2-pthreads-1D.dat`

I did not save a separate .dat file for each run of the script for each algorithm. Instead each .dat file contains the time taken for each matrix size (and block size, if applicable) for the preset range of values. **./runAssignment2.sh** may be updated with more if needed but the following are those in use at the moment.

```
#  Matrix & Thread size - range 1
   declare -a NXArray=( 50 50 50 50 50 50 100 100 100 100 100 100 )
   declare -a NPArray=( 10 10 10 10 10 10 20 20 20 20 20 20 )
#  Matrix & Thread size - range 2
#  declare -a NXArray=( 50 50 50 100 100 100 500 500 500 500 1000 1000 1000 1000 )
#  declare -a NPArray=( 2 5 10 5 10 20 5 10 20 50 5 10 50 100 )
```

For compilation using the script, a suffix of ***-atlas*** indicates compilation for atlas and a suffix of ***-cblas*** indicates that the c program was compiled via cblas.

Finally a log file containing a listing of each algorithm used for that iteration.

After each run, all .log, .txt, .dat and .bup files are copied to the directory *logDir/*.

If ***./A2-pthreads-1D-cblas*** or any of the other algorithm files is used without the script then .dat and .txt files may be named whatever the user wishes and no .log file applies.

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 if ran standalone.

I also spot-checked the results as practical. Results spot-check are detailed in Appendix I – Validate Results, while sample results are listed in Appendix II – summary of time taken using predefined values.

GNUplot Execution

I followed the same structure for each .dat file as produced, an example follows :

Sample .dat file	

If wished, the .txt file contains the matrices |A| and |B| used to calculate |C| and the type of computation applicable and the time taken to complete. The .dat file is just a summary of the matrix and block sizes (when the later is applicable) as well as time taken for each type of computation.

The contents of each .dat was then presented in graphical format using GNUplot, comparing times taken for manual and for BLAS/ATLAS computations.

Sample GNUplot program execution

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

# Paula Dwan : Assignment 2
reset
set xtic auto
set ytic auto
set size 1,1
set grid
set key outside
#
set title 'pthreads : Matrix size -v- Time taken'
set ylabel 'Time taken / seconds'
set xlabel 'Matrix size'
set xrange [0:1100]
set yrange [0:34]
set xtics (100,200,300,400,500,600,700,800,900,1000,1100)
set origin 0,0
set key outside
plot 'logDir/Data-pthreads.dat' u 1:3 t 'manual' w l lw 0.8 lc rgb 'blue',
'logDir/Data-pthreads .dat' u 1:5 t 'dgemm' w l lw 0.8 lc rgb 'black',
'logDir/Data-pthreads .dat' u 1:7 t 'pthreads' w l lw 0.8 lc rgb 'red'
#
pause -1
```

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

Screen shots of each were taken and added to the section Summary Results.

SUMMARY RESULTS :

SUMMARY RESULTS :

Build/plot:	• The dependence of the execution time of the program on the matrix size n. • The speedup over a serial counterpart of the program.
Variant :	• *One-step algorithm. No intermediate resulting matrix.* • *Left matrix is horizontally partitioned* • *The maximum absolute row sum norm (aka infinity-norm):* $\|\|A\|\|_\infty = \max_{0 \le i < n} \sum_{j=0}^{n-1} \|a_{ij}\|$
Infinity norm	*Sum the absolute values along each row and then take the largest value as the answer.* *Example:* $A = \begin{vmatrix} 1 & -7 \\ -2 & -3 \end{vmatrix}$ *then matrix norm of A = max (1 + |-7|, |-2| + |-3|) = max (8, 5) =* ***8***

GNUPLOT GRAPHS – CBLAS USING MATRIX SIZES
(50 50 50 100 100 100 500 500 500 500 1000 1000 1000 1000)

$ ./RUNASSIGNMENT2.SH -A <-R|-I> -V -D2

manual

dgemm

pthreads

GNUPLOT GRAPHS – CBLAS USING MATRIX SIZES
(50 50 50 50 50 50 100 100 100 100 100 100)

$ ./RUNASSIGNMENT2.SH -A <-I|-R> -V -D2

manual

dgemm

pthreads

CONCLUSIONS

When applying the three options I used matrices |A| and |B| of varying sizes from [10x10] to [1000x1000]. The values in each matrix was Dependant on the switch -r (random from 1 to 10) and -r (column index + 1), so all cells in column 1000 had a value of 1001.

For the straight-forward algorithm of **|C|***[ij]* += **|A|***[ik]* * **|B|***[kj]* uses row major as follows (e.g.: [4x4] matrix:

0	*1*	*2*	*3*
4	*5*	*6*	*7*
8	*9*	*10*	*11*
12	*13*	*14*	*15*

APPENDICES

APPENDIX I – VALIDATE RESULTS

Spot check only using 10x10 matrices, initializing matrices |A| and |B| using successive column values.

Build using :

Resulting sample Matrix .txt file contains :

Resulting sample summary timing data file contains :

Validating results gives :

APPENDIX III – REFERENCES / ACKNOWLEDGEMENTS

www.stackoverflow.com

www.cs.indiana.edu