Package 'MagmaQR'

September 6, 2019

Type Package

Title Implemention of QR factorisation via the multi-GPU-based functionality in the MAGMA library

Version 0.1.0

Date 2019-07-01

Author Joshua Bowden [aut, cre], Andrew George [aut, cre]

Maintainer Andrew George <andrew.george@csiro.au>

Description Provides a client-server based interface to the QR factorisation (single-/multi-gpu routines) from the MAGMA library. Can be used as replacement for the qr.Q() function in R. We get the server to check how many GPUs are present therefore we do not need CUDA/other interface to be present for compilation of the MagmaQR package. Required is the MAGMA library available (http://icl.cs.utk.edu/magma/) compiled as position independent code with a multi-threaded high performance LAPACK and -DMAGMA_ILP64 defined (an example make.inc file should be available in MAGMA download package).

NeedsCompilation yes

License GPL-3 + file LICENCE

LazyLoad yes

Depends R (>= 3.5), Rcpp (>= 0.11.0)

Imports Rcpp

LinkingTo Rcpp

SystemRequirements GNU make

R topics documented:

`GetServerArgs` *Initialise shared memory on client and obtain server launch string.*

Description

This function will create a CSharedMemory object that initialises the shared memory region and the semaphore used for comms beteween client and server. If the object is already initialised it is removed and reinitialised. Returns a string of the form "-n 10000 -v 1 -g 3 -m /syevx_<PID_of_client> -s /sem_<PID_of_client> -p" that can be used to launch a qr_server process that will accept matrix data on which to perform QR factorisation and return the QR matrix.

Usage

```
GetServerArgs(matrixDimension, withVectors, numGPUsWanted, memName, semName,
  printDetails)
```

Arguments

`matrixDimension`	- type (integer) - the dimension of the (assumed square) matrix
`withVectors`	- type (boolean) - this is always set to true. It is a legacy parameter from Joshs Eigendecomposition code.
`numGPUsWanted`	- type (string) - The number of GPUs to use
`memName`	- type (string) - a name to give to the named shared memory region (will be created in /dev/shm/) and defaults to the user name if nothing specified
`semName`	- type (string) - a name to give to the semaphore (will be placed in /dev/shm) and defaults to the user name if nothing specified
`printDetails`	- type (integer 0|1|2) - 0 = don't print, 1 = print details of server progress to screen; 2 = print to log (not functional)

Value

- type (string) A string that can be used a command line arguments to run the qr_server executable

`MagmaQR` *MagmaQR - provides a fast replacement for the qr() function, using GPU based MAGMA library routine.*

Description

Implements the QR factorisation using the MAGMA library (http://icl.cs.utk.edu/magma/index.html) multi-GPU implementation with 64 bit integer interface. Currently only for square matricies. In this case it provides a direct replacement for R function qr.Q(). The package uses the 64 bit integer MAGMA library by a client-server shared memory architecture. This removes the problem that can arrise with larger datasets where R only provides the 32 bit BLAS/LAPACK interface. The server side code checks how many GPUs are present so the client side R package code does not require CUDA/other interface to be present. The server side code requires the MAGMA library to be available (http://icl.cs.utk.edu/magma/) compiled as position independent code (shared library) with a

multi-threaded high performance LAPACK and -DMAGMA_ILP64 defined (an example make.inc file should be available in MAGMA download package). The server code also requires an OpenCL library to be installed which is used to get the number of GPUs present on a system. This package can be used in conjunction with HiPLARb and HiPLARM that require high performance, but single-threaded BLAS routines, which will then degrade the performance of some routines (such as eigen() and possibly svd()) which require a multi threaded BLAS to operate effectively.

Compilation of the R package the 'client side' code: Optionally set MAGMA_HOME and CUDA_ROOT as per the server side instructions. This will alow the client to compile the server code during package install. Setting the OpenCL platform string and device type:

Compilation of the 'server side' code: Requires environment variables MAGMA_HOME to be set to where the MAGMA install is present. MAGMA_HOME=/usr/local/magma is the default. The CUDA version of MAGMA requires CUDA_ROOT to be set CUDA_ROOT=/usr/local/cuda is the default. The server code uses OpenCL to determine how many GPUs are present on the system it is being run on. The default platform string is stored in a file in <R library path>/MagamQR/extdata/platformstring.txt and contains the default platform string "NVidia", and the default device is set to "GPU". See the platformstring.txt file for other options. If these libraries and variables are set correctly then the server side code will be compiled automatically when the package is installed. If failing installation on install then whenerver the package is loaded into the R environment using library(MagmaQR) the system will attempt to compile the server code.

N.B. Calling the solve_mgpu() function with argument overwrite=TRUE will cause an *overwrite of the input matrix data* with the eigenvectors of the original matrix data (if they are requested). This is done to potentialy reduce the memory footprint of the function. If overwrite=TRUE then please ensure the original matrix is copied if the data needs to be used after the function is called. Using overwrite=FALSE will return the usual list of results in $vectors and $values list items of the result object.

Details

Package:	MagmaQR
Type:	Package
Version:	1.0.0
Date:	2019-09-01
License:	GPL-3 + file LICENCE
LazyLoad:	yes

Author(s)

Josh Bowden & Andrew George, CSIRO

Maintainer: Andrew George <andrew.george@csiro.au>

References

Stanimire Tomov, Jack Dongarra, Marc Baboulin, Towards dense linear algebra for hybrid GPU accelerated manycore systems, Parallel Computing, Volume 36, Issues 5-6, June 2010, Pages 232-240, ISSN 0167-8191, http://dx.doi.org/10.1016/j.parco.2009.12.005.

@article title = Towards dense linear algebra for hybrid GPU accelerated manycore systems, author = Stanimire Tomov and Jack Dongarra and Marc Baboulin, booktitle = Parallel Matrix Algorithms

and Applications, doi = 10.1016/j.parco.2009.12.005, issn = 0167-8191, journal = Parallel Computing, month = jun, number = 5-6, pages = 232–240, posted-at = 2010-12-17 09:48:58, priority = 2, volume = 36, year = 2010

Solca, Raffaele; Haidar, Azzam; Tomov, Stanimire; Schulthess, Thomas C.; Dongarra, Jack, "Abstract: A Novel Hybrid CPU-GPU Generalized Eigensolver for Electronic Structure Calculations Based on Fine Grained Memory Aware Tasks," in High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion: , vol., no., pp.1338-1339, 10-16 Nov. 2012 doi: 10.1109/SC.Companion.2012.173

@articleHaidar:2014:NHC:2747699.2747703, author = Haidar, Azzam and Tomov, Stanimire and Dongarra, Jack and Solc\'a, Raffaele and Schulthess, Thomas, title = A Novel Hybrid CPU-GPU Generalized Eigensolver for Electronic Structure Calculations Based on Fine-grained Memory Aware Tasks, journal = Int. J. High Perform. Comput. Appl., issue_date = May 2014, volume = 28, number = 2, month = may, year = 2014, issn = 1094-3420, pages = 196–209, numpages = 14, url = http://dx.doi.org/10.1177/1094342013502097, doi = 10.1177/1094342013502097, acmid = 2747703, publisher = Sage Publications, Inc., address = Thousand Oaks, CA, USA, keywords = Eigensolver, GPU, electronic structure calculations, generalized eigensolver, high performance, hybrid, multicore, two-stage,

See Also

eigen

Examples

```
# setup

# setup
set.seed(101)
n <- 200
ngpu <- 1
K <- matrix(sample(c(0,1), n*n, TRUE ), nrow=n)
res  <- tcrossprod(K)
print(res[1:5,1:5])

# CPU based
 library(MagmaQR)
 MagmaQR::RunServer( matrixMaxDimension=n,  numGPUsWanted=ngpu, memName="/syevd_mem", semName="/syevd_sem",

  qGPU  <- MagmaQR::qr_mgpu(res)

 ## CPU
 qCPU  <- qr.Q(qr(res))


   print(qGPU[1:5, 1:5])
   print("---------------------")
   print(qCPU[1:5, 1:5])


   print(c("Test Sum = ", sum(qGPU - qCPU) ))


  StopServer()  # Client signals to server to terminate
```

MakeServer — *Creates the server executable.*

Description

Creates qr_server executable using a call to 'make' and the makefile and make.inc information present in the <package root>/src directory. Users have to set the following variable in make.inc : MAGMALIB =$(MAGMA_HOME)/lib # The path to the MAGMA library

Users must ensure that MAGMA_HOME and possibly CUDA_ROOT environment variables have been set in the shells environment or the variables can be set using environmentSetup paramater. e.g. "environmentSetup="env MAGMA_HOME=/apps/magma/2.5.1a1-ipl64-cuda90 CUDA_ROOT=/apps/cuda/9.0.17 "

Usage

```
MakeServer(environmentSetup = "", target = "all")
```

Arguments

environmentSetup	- type (string) e.g. "env LD_LIBRARY_PATH=/usr/local/magma-1.7.1:$LD_LIBRARY_PATH "
target	- type (string) The make target e.g. all | clean | dist-clean | install

Value

A character vector containing output of the make process

qr_mgpu — *Obtain the Q matrix from QR factorisation of a square matrix using MAGMA multi-gpu routines*

Description

This function performs the QR factorisation of the input matrix and returns the Q matrix. The method involves the offload of the matrix data to a seperate syevd_server executable by copying data into a shared memory area and signalling to the server that the data is availble. This function will block until the server has completed the decomposition. The function checks that the input is square. N.B. The maximum size allowed of the input matrix is goverend by what was provided in the MagmaQR::RunServer() function. The server will automatically be restarted with a larger shared memory area if the user wants to perform QR on a larger matrix.

Usage

```
qr_mgpu(matrix, symmetric=TRUE, only_values=FALSE, overwrite=FALSE, printInfo=FALSE)
```

Arguments

matrix	- the input matrix to be used in QR factorisation. It must be square
symmetric	ignored
only_values	ignored
overwrite	ignored
printInfo	set to TRUE if additionaly output is needed

Value

The Q matrix is returned.

RunServer *Creates the R client side shared memory region and then launches a server process which is given access to the shared region. The server then waits for the R client to give it a matrix on which it will compute the eigenvalue decomposition of useing a syevdx_2stage MAGMA library function.*

Description

Creates the R client side shared memory region and then launches a server process which is given access to the shared region. The server then waits for the R client to give it a matrix on which it will compute the eigenvalue decomposition of useing a syevdx_2stage MAGMA library function.

Usage

```
RunServer(environmentSetup = "", numGPUsWanted = 0,
  matrixMaxDimension = 0, memName = "", semName = "", print = 0)
```

Arguments

environmentSetup	- type (string) - Environment variables that need to be set, such as library include paths
numGPUsWanted	- type (string) - The number of GPUs to use in for the symmetric eigenvalue (syevd) computation
matrixMaxDimension	- type (integer) - The maximum matrix size that this server instance can handle - sets the shared memory size
memName	- type (string) - A name to give to the named shared memory region (will be created in /dev/shm/) and defaults to the user name if nothing specified
semName	- type (string) - A name to give to the semaphore (will be placed in /dev/shm) and defaults to the user name if nothing specified
print	- type (integer 0|1|2) - 0 = don't print, 1 = print details of server progres to screen; 2 = print to log (not functional)

Details

This function creates a command line with which to call the syevd_server executable and then calls the executable with a non-blocking system() call to launch the server process. The server then waits for the client to send it matrix data via the syevdx_client() function. The matrixMaxDimension paramater specifies the largest size matrix that can be processed by this instance of the syevd_server().

Value

a vector character values containing output of the make process

---

`StopServer` *Function signals to the server through shared memory region to terminate from its main loop and then deletes the client CSharedMemory object*

Description

Function signals to the server through shared memory region to terminate from its main loop and then deletes the client CSharedMemory object

Usage

```
StopServer()
```

Index