Giapponese → Inglese

Programming with OpenACC directives

12 – Chapter 1 Using CUDA Library on OpenACC

1. Introduction to cuSOLVER library (Dense LAPACK, Sparse LAPACK)
2. cuSolverDN: Cholesky factorization of Hermitian positive definite matrices (equivalent to LAPACK Zpotrf)
   - CUDA Fortran
   - OpenACC Fortran
   - OpenACC C++
3. cuSolverDN: Solver using Cholesky decomposition of symmetric positive definite matrices (equivalent to LAPACK Dpotrs)
   - OpenACC Fortran
4. cuSolverDN: Linear system solver using LU decomposition (equivalent to LAPACK Dpotrs)
   - OpenACC Fortran
5. cuSolverDN: Eigenvalue solver for symmetric matrices (equivalent to LAPACK Dsyevd)
   - OpenACC Fortran
   - OpenACC C++
6. cuSolverDN: Dense matrix linear solver with QR decomposition (equivalent to LAPACK Dgeqrf)
   - OpenACC Fortran
7. cuSolverDN: Orthogonalization by QR decomposition (equivalent to LAPACK Dgeqrf,Dorgqr)
   - OpenACC Fortran
8. cuSolverDN: Singular value decomposition including singular vector calculation (by QR) (equivalent to LAPACK Dgesvd)
   - OpenACC Fortran
9. cuSolverDN: Singular value decomposition including singular vector calculation (By Jacobi) (equivalent to LAPACK Dgesvd)
   - OpenACC Fortran
10. cuSolverSP: Parallel direct sparse solver (CSR QR decomposition)
    - Batched Sparse QR: CUDA Fortran
    - Batched Sparse QR : OpenACC Fortran
11. cuFFT example with OpenACC Fortran (FFT)
12. cuRAND example with OpenACC Fortran (random number generation)
    - OpenACC Host Code
    - OpenACC device code
13. cuSPARSE example from OpenACC Host code

1. Introduction to cuSOLVER library (Dense LAPACK, Sparse LAPACK)

The cuSOLVER library is a high-level package based on the cuBLAS and cuSPARSE libraries. Each library can be used alone or in conjunction with other toolkit libraries. The purpose of cuSolver is to provide useful features of the CUDA version of **LAPACK , such as least squares solver for dense matrices, sparse matrices, general matrix factorization and trigonometry for eigenvalue solvers.** In addition, cuSOLVER also provides a new refactoring library that helps solve sequences of matrices that share sparse patterns.

As of PGI 17.10, the only licensed product that provides a Fortran Interface to the cuSOLVER library (cusolverDn.mod) is **for Linux** . Please note that the Fortran Interface Module is not provided for the Windows version at this time. Note that if you create a Fortran interface module, it can also be used on Windows.

The first part of cuSOLVER , called **cuSolverDN** , provides useful utilities such as dense matrix factorization, solution routines such as LU, QR, SVD, and LDLT, and matrix and vector permutations. Provides functionality similar to so-called LAPACK routines.

Second, **cuSolverSP** provides a new set of sparse routines based on sparse QR decomposition. Since not all matrix factorizations have parallel sparsity patterns, the cuSolverSP library also provides a CPU path for handling sequential-like matrices. For matrices with high parallelism, GPU paths provide higher performance. The library is designed to be called from C and C++, but is also easily available from Fortran via the OpenACC + PGI Fortran Module.

The third one is **cuSolverRF** . This is a refactorization package for sparse matrices that can provide very good performance when solving sequences of matrices where only the coefficients are changed and the sparsity pattern remains the same. (Not explained on this page)

cuSolverDN library (Dense LAPACK)

The cuSolverDN library is designed to solve dense linear systems.

$$Ax = b$$

Here, coefficient matrix A∈R ^{nxn} , right-hand side vector b∈R ^{n} , solution vector x∈R ^{n}

The cuSolverDN library provides partial pivoting on QR decomposition and LU to handle general matrices A that may be asymmetric, and Cholesky decomposition routines are also provided for symmetric/Hermitian matrices. In addition, for symmetric indefinite matrices, Bunch-Kaufman (LDL) decomposition is provided. The cuSolverDN library also provides useful bidiagonalization routines and singular value decomposition (SVD).

The cuSolverDN library targets LAPACK's computationally intensive and common routines and provides an API compatible with LAPACK, making it easy to migrate existing LAPACK-based programs. A user can use his cuSolverDN to accelerate these time-consuming routines and keep existing code compatible with her LAPACK usage without significant changes.

cuSolverSP library (Sparse LAPACK)

The cuSolverSP library is primarily designed to solve sparse linear systems and least squares problems.

$$Ax = b$$
$$x = \text{argmin}||A * z - b||$$

Here, coefficient matrix A∈R ^{nxn} , right-hand side vector b∈R ^{n} , and solution vector x∈R ^{n} . For linear systems, m = n is required.

The core algorithm is based on sparse QR decomposition. Enter matrix A in CSR format. If matrix A is symmetric/Hermitian, the user must provide the full matrix. That is, you need to fill in the missing bottom or top. If matrix A is symmetric positive definite and the user only needs to solve for his Ax = b, the Cholesky decomposition will work, but the user can substitute can.

On top of the linear and least squares solvers, the cuSolverSP library provides a simple eigenvalue solver based on the shift-inverse power method and a function to count the number of eigenvalues contained in a box in the complex plane.

2. cuSolverDN: Cholesky decomposition of Hermitian positive definite matrix (equivalent to LAPACK zpotrf)

The cuSOLVER library is based on the cuBLAS and cuSPARSE libraries, supports matrix factorization and triangulation routines for dense matrices, sparse least squares solvers and eigenvalue solvers, and provides a refactoring library to help solve sequences of matrices with covariance matrices. provide. **Optimized cuSolverDN type** routines (cuSolverSP and cuSolverRF type (will be supported in a future release). This same cuSolver library is built using the PGI compiler and is also compatible with the PGI OpenMP runtime, so it can also be called from PGI OpenACC C/C++.

The following example is an example of using cuSolverDN: dense LAPACK Function from OpenACC Fortran/C++ and CUDA Fortran.

The programming response differs depending on which programming model is used to manage device memory for arrays and variables, but here we will explain how to declare a device array on the CUDA Fortran side, and how to declare an array on the OpenACC side. An example of this is shown below.

The example below is the result using **PGI 17.10 version .** If you want to check the version you are using, specify the -V command option as follows.

```
$ pgfortran -V
pgfortran 17.10-0 64-bit target on x86-64 Linux -tp haswell
PGI Compilers and Tools
Copyright (c) 2017, NVIDIA CORPORATION. All rights reserved.
```

Dense Linear Solver - cusolverDnZpotrf() with CUDA Fortran

An example of the Cholesky decomposition of a Hermitian positive definite matrix is shown. It is equivalent to **LAPACK's ZPOTRF routine .** A is an n×n Hermitian matrix, and only the lower or upper parts are meaningful. The following example is a program using cuSOLVER from CUDA Fortran. As a PGI Fortran MODULE interface, it is necessary to specify cublas_v2 and cusolverDn in the USE statement. For input parameter CUBLAS_FILL_MODE_LOWER, only the lower triangular part of A is processed and replaced by the lower triangular Cholesky coefficient L. ($A = L * L^H$) The workspace pointed to by the input parameter Workspace must be provided. The input parameter Lwork is the size of the work area, returned by potrf_bufferSize(). Note that the dense matrix A must be input in column-major order .

CUDA Fortran + cuSOLVER(cusolverDnZpotrf)

```
! Copyright (c) 2017, NVIDIA CORPORATION. All rights reserved.
!
!
! Permission is hereby granted, free of charge, to any person obtaining a
! copy of this software and associated documentation files (the "Software"),
! to deal in the Software without restriction, including without limitation
! the rights to use, copy, modify, merge, publish, distribute, sublicense,
! and/or sell copies of the Software, and to permit persons to whom the
! Software is furnished to do so, subject to the following conditions:
!
! The above copyright notice and this permission notice shall be included in
! all copies or substantial portions of the Software.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
! IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
! FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
! THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
! LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
! FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
! DEALINGS IN THE SOFTWARE.
!
program main
  use cublas_v2
  use cusolverDn
  use cudafor
  implicit none
  integer, parameter :: n=3
  complex(8) :: a(n,n)
  complex(8), device :: a_d(n,n)
  complex(8), device , allocatable :: workspace_d(:)
  integer, device :: devInfo_d
  integer :: istat, Lwork
  type(cusolverDnHandle) :: h

  a(1,1) = 25.0; a(1,2) = 15.0; a(1,3) = -5.0
  a(2,1) = a(1,2); a(2,2) = 18.0; a(2,3) = 0.0
  a(3,1) = a(1,3); a(3,2) = a(2,3); a(3,3) = 11.0

  a_d = a

!handle creation
  istat = cusolverDnCreate(h)
  if (istat /= CUSOLVER_STATUS_SUCCESS) &
       write(*,*) 'handle creation failed'
! Working buffer size calculation
  istat = cusolverDnZpotrf_bufferSize(h, &
       CUBLAS_FILL_MODE_LOWER, n, a_d, n, Lwork)

  if (istat /= CUSOLVER_STATUS_SUCCESS) &
       write(*,*) 'cusolverDnZpotrf_buffersize failed'

  allocate(workspace_d(Lwork))
! Cholesky decomposition
  istat = cusolverDnZpotrf(h, CUBLAS_FILL_MODE_LOWER, &
       n, a_d, n, workspace_d, Lwork, devInfo_d)

  if (istat /= CUSOLVER_STATUS_SUCCESS) &
       write(*,*) 'cusolverDnZpotrf failed'

  istat = devInfo_d
  if (istat /= 0) write(*,*) 'Cholesky factorization failed'

  istat = cusolverDnDestroy(h)
  if (istat /= CUSOLVER_STATUS_SUCCESS) &
       write(*,*) 'handle destruction failed'

  a = a_d

  write(*,"(3(f0.0,SP,f0.0,'i',2x))") a(1,:)
  write(*,"(3(f0.0,SP,f0.0,'i',2x))") a(2,:)
  write(*,"(3(f0.0,SP,f0.0,'i',2x))") a(3,:)
end program main
```

It is necessary to compile and link with -acc -Mcudalib=cusolver,cublas and -Mcuda options as compile options.

Compile & run

```
$ pgfortran -fast -Minfo -Mcuda=cc60,cc35,cuda8.0 -Mcudalib=cusolver,cublas   testDn.
main:
     60, Loop unrolled 3 times (completely unrolled)
     61, Loop unrolled 3 times (completely unrolled)
     62, Loop unrolled 3 times (completely unrolled)
$ ./a.out
5.+0.i +15.+0.i -5.+0.i
3.+0.i +3.+0.i +0.+0.i
-1.+0.i +1.+0.i +3.+0.i
```

Dense Linear Solver - cusolverDnZpotrf() with OpenACC Fortran

Rewrite the above CUDA Fortran program using OpenACC. As a PGI Fortran MODULE interface, it is necessary to specify cublas_v2 and cusolverDn in the USE statement. You can use the CUDA math library just by using OpenACC directives, without having to worry about memory allocation on the device side. This means that it is easy to port existing Fortran programs.

One thing to keep in mind with the OpenACC directive is that if the actual argument passed when calling a (library) routine written in CUDA Fortran or CUDA C is a "device pointer", **host_data use_device** is used to inform the compiler of this. **()** use. If you use this directive correctly, porting an existing program-based program using OpenACC should not be a big burden.

Top of page

OpenACC Fortran + cuSOLVER(cusolverDnZpotrf)

```
! Copyright (c) 2017, NVIDIA CORPORATION. All rights reserved.
!
!
! Permission is hereby granted, free of charge, to any person obtaining a
! copy of this software and associated documentation files (the "Software"),
! to deal in the Software without restriction, including without limitation
! the rights to use, copy, modify, merge, publish, distribute, sublicense,
! and/or sell copies of the Software, and to permit persons to whom the
! Software is furnished to do so, subject to the following conditions:
!
! The above copyright notice and this permission notice shall be included in
! all copies or substantial portions of the Software.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
! IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
! FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
! THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
! LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
! FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
! DEALINGS IN THE SOFTWARE.
!
program main
  use cublas_v2
  use cusolverDn
  use cudafor
  implicit none
  integer, parameter :: n=3
  complex(8) :: a(n,n)
!!! complex(8), device :: a_d(n,n)              ! No need to declare device variables
!!! complex(8), device, allocatable :: workspace_d(:)
!!! integer, device :: devInfo_d
  complex(8), allocatable :: workspace_d(:)
  integer :: devInfo_d
  integer :: istat, Lwork
  type(cusolverDnHandle) :: h

  a(1,1) = 25.0; a(1,2) = 15.0; a(1,3) = -5.0
  a(2,1) = a(1,2); a(2,2) = 18.0; a(2,3) = 0.0
  a(3,1) = a(1,3); a(3,2) = a(2,3); a(3,3) = 11.0

!!! a_d = a

  istat = cusolverDnCreate(h)
  if (istat /= CUSOLVER_STATUS_SUCCESS) &
       write(*,*) 'handle creation failed'

!$acc data copy(a) create(workspace_d)

!$acc host_data use_device(a)
  istat = cusolverDnZpotrf_bufferSize(h, &
       CUBLAS_FILL_MODE_LOWER, n, a, n, Lwork)
!$acc end host_data

  if (istat /= CUSOLVER_STATUS_SUCCESS) &
       write(*,*) 'cusolverDnZpotrf_buffersize failed'
  print *, "working space (bytes) =", Lwork
  allocate(workspace_d(Lwork))

!$acc enter data copyin(workspace_d) copyin(devInfo_d)

!$acc host_data use_device(a, workspace_d, devInfo_d)
  istat = cusolverDnZpotrf(h, CUBLAS_FILL_MODE_LOWER, &
       n, a, n, workspace_d, Lwork, devInfo_d)
!$acc end host_data

  if (istat /= CUSOLVER_STATUS_SUCCESS) &
       write(*,*) 'cusolverDnZpotrf failed'

!$acc exit data copyout(devInfo_d)
!$acc exit data delete(workspace_d)
!$acc end data

  istat = devInfo_d
  if (istat /= 0) write(*,*) 'Cholesky factorization failed'


  istat = cusolverDnDestroy(h)
  if (istat /= CUSOLVER_STATUS_SUCCESS) &
       write(*,*) 'handle destruction failed'

!!! a = a_d

  write(*,"(3(f0.0,SP,f0.0,'i',2x))") a(1,:)
  write(*,"(3(f0.0,SP,f0.0,'i',2x))") a(2,:)
  write(*,"(3(f0.0,SP,f0.0,'i',2x))") a(3,:)
end program main
```

It is necessary to compile and link with -acc, -Mcudalib=cusolver,cublas , and -Mcuda options as compile options. In addition to OpenACC's -acc option, the -Mcuda option is required to tell the linker the list of CUDA-related system libraries.

Compile & run

```
$ pgf90 -fast -Minfo -acc -ta=tesla,cc60,cc35,cuda8.0 -Mcudalib=cusolver,cublas opena
main:
     48, Generating copy(a(:,:))
         Generating create(workspace_d(:))
     60, Generating enter data copyin(devinfo_d,workspace_d(:))
     70, Generating exit data copyout(devinfo_d)
     71, Generating exit data delete(workspace_d(:))
     84, Loop unrolled 3 times (completely unrolled)
     85, Loop unrolled 3 times (completely unrolled)
     86, Loop unrolled 3 times (completely unrolled)
$ ./a.out
 working space (bytes) = 1
5.+0.i +15.+0.i -5.+0.i
3.+0.i +3.+0.i +0.+0.i
-1.+0.i +1.+0.i +3.+0.i
```

Top of page

Dense Linear Solver - cusolverDnZpotrf with OpenACC C++

Rewrite the Fortran program above using PGI C++ OpenACC instead of the NVIDIA nvcc compiler. You can also create C/C++ programs that use OpenACC and NVIDIA CUDA libraries. In the example below, when referencing the NVIDIA cuSOLVE library from a C++ program, the complex number type must be declared using a CUDA complex number type (for example, cuDoubleComplex, etc.). C++ std:complex template complex If you make a declaration with , type matching of the cuSOLVE routine argument could not be done, so it is safe to use the CUDA complex type. In addition, the numerical arrangement of cuSOLVER's "dense matrix" is column-major order (Fortran type) .

OpenACC + cusolverDnDsyevd (DnZpotrf.cpp)

```
! Copyright (C) HPC WORLD (Prometech Software, Inc. and GDEP Solutions, Inc.) All rights reserve
!
! Permission is hereby granted, free of charge, to any person obtaining a
! copy of this software and associated documentation files (the "Software"),
! to deal in the Software without restriction, including without limitation
! the rights to use, copy, modify, merge, publish, distribute, sublicense,
! and/or sell copies of the Software, and to permit persons to whom the
! Software is furnished to do so, subject to the following conditions:
!
! The above copyright notice and this permission notice shall be included in
! all copies or substantial portions of the Software.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
! IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
! FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
! THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
! LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
! FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
! DEALINGS IN THE SOFTWARE.

/*
 * How to compile (assume cuda is installed at /usr/local/cuda-9.0/)
 * pgc++ -I/usr/local/cuda-9.0/include DnZpotrf.cpp -acc -ta=tesla,cc60,cc35 -Mcudalib=cusolver
 *
 *
 */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <cusolverDn.h>

void printD_CMatrix(int m, int n, const cuDoubleComplex *A, int lda, const char* name)
{
    for(int row = 0 ; row < m ; row++){
        for(int col = 0 ; col < n ; col++){
            cuDoubleComplex Areg = A[row + col*lda];
            printf("%s(%d,%d) = %f %f\n", name, row+1, col+1, cuCreal(Areg), cuCimag(Areg));
        }
    }
}

int main(int argc, char*argv[])
{
    cusolverDnHandle_t cusolverH = NULL;
    cusolverStatus_t cusolver_status = CUSOLVER_STATUS_SUCCESS;
    const int m = 3;
    const int lda = m;

/* | 25 15 -5 |
 * A = | 15 18 0 |
 * | -5 0 11 |
 *
 */
    cuDoubleComplex *A; // input

    int *devInfo = NULL;
    cuDoubleComplex *work = NULL;
    int lwork = 0;

    A = (cuDoubleComplex *) malloc (sizeof(cuDoubleComplex) * lda * m);
    devInfo = (int *) malloc (sizeof(int));

// step 1: create cusolver/cublas handle
    cusolver_status = cusolverDnCreate(&cusolverH);

// step 2: set A
    A[0] = make_cuDoubleComplex(25.0,0.0);
    A[3] = make_cuDoubleComplex(15.0,0.0);
    A[6] = make_cuDoubleComplex(-5.0,0.0);
    A[1] = A[3];
    A[4] = make_cuDoubleComplex(18.0,0.0);
    A[7] = make_cuDoubleComplex(0.0,0.0);
    A[2] = A[6];
    A[5] = A[7];
    A[8] = make_cuDoubleComplex(11.0,0.0);

    printD_CMatrix(m, m, A, lda, "A");
    printf("=====\n");

// step 3: query working space of syevd
    cublasFillMode_t uplo = CUBLAS_FILL_MODE_LOWER;

#pragma acc data copy(A[0:lda*m], devInfo[0:1])
{
#pragma acc host_data use_device(A)
  {
    cusolver_status = cusolverDnZpotrf_bufferSize(
        cusolverH,
        uplo,
        m,
        A,
        lda,
        &lwork);
  }
```

```
    assert (cusolver_status == CUSOLVER_STATUS_SUCCESS);

    work = (cuDoubleComplex *) malloc(sizeof(cuDoubleComplex)*lwork);
    printf("Lwork = %d\n",lwork);

#pragma acc enter data create(work[0:lwork])

// step 4: compute spectrum
#pragma acc host_data use_device(A, work, devInfo)
  {
    cusolver_status = cusolverDnZpotrf(
        cusolverH,
        uplo,
        m,
        A,
        lda,
        work,
        lwork,
        devInfo);
  }
#pragma acc wait
#pragma acc exit data delete(work)
}
    printf("DevInfo = %d\n",devInfo[0]);

    printf("A = (base 1)\n");
    printD_CMatrix(m, m, A, lda, "A_result");
    printf("=====\n");

// free resources
    if (A) free(A);
    if (devInfo) free(devInfo);
    if (work) free(work);

    if (cusolverH) cusolverDnDestroy(cusolverH);

    return 0;
}
```

If you want to use cuSOLVER with the PGI C/C++ OpenACC compiler, compile using the C/C++ include files bundled with the NVIDIA CUDA Toolkit (the PGI compiler includes cuSOLVER/cuBLAS libraries for C/C++). include files such as are not bundled). The example here is written assuming that CUDA 9.0 is implemented under /usr/local/cuda-9.0. It is necessary to compile and link with the –I/usr/local/cuda-9.0/include -acc -ta=tesla -Mcudalib=cusolver option as a compile option.
(For CUDA 8.0)
pgc++ -I/usr/local/cuda-8.0/include DnZpotrf.cpp -acc -ta=tesla,cc60,cc35 -Mcudalib=cusolver -Minfo=accel -O2
(For CUDA 9.0)
pgc++ – I/usr/local/cuda-9.0/include DnZpotrf.cpp -acc -ta=tesla,cc60,cc35 -Mcudalib=cusolver -Minfo=accel -O2

Compile & run

```
[kato@photon32 C]$ pgc++ -I/usr/local/cuda-9.0/include DnZpotrf.cpp -acc -ta=tesla,cc60,cc35
-Mcudalib=cusolver -Minfo=accel -O2
main:
     64, Generating copy(A[:9],devInfo[:1])
     84, Generating enter data create(work[:lwork])
     97, Generating exit data delete(work[:1])
[kato@photon32 C]$ a.out
A(1,1) = 25.000000 0.000000
```

```
A(1,2) = 15.000000 0.000000
A(1,3) = -5.000000 0.000000
A(2,1) = 15.000000 0.000000
A(2,2) = 18.000000 0.000000
A(2,3) = 0.000000 0.000000
A(3,1) = -5.000000 0.000000
A(3,2) = 0.000000 0.000000
A(3,3) = 11.000000 0.000000
=====
Lwork = 1
DevInfo = 0
A = (base 1)
A(1,1) = 5.000000 0.000000
A(1,2) = 15.000000 0.000000
A(1,3) = -5.000000 0.000000
A(2,1) = 3.000000 0.000000
A(2,2) = 3.000000 0.000000
A(2,3) = 0.000000 0.000000
A(3,1) = -1.000000 0.000000
A(3,2) = 1.000000 0.000000
A(3,3) = 3.000000 0.000000
=====
```

Top of page

3. cuSolverDN: Linear solver using Cholesky decomposition of symmetric positive definite matrices (equivalent to LAPACK Dpotrs)

This is an example of the use of a double-precision solver (cusolverDnDpotrs) that solves a symmetric positive definite matrix using Cholesky decomposition. Cholesky decomposition uses the cusolverDnDpotrf routine to solve a system of linear equations using cusolverDnDpotrs. Note that RHS (right side) is solved by giving multiple vectors. Note that the dense matrix A must be input in column-major order .

OpenACC Fortran + cusolverDnDpotrf, cusolverDnDpotrs (Solver.f90)

```
! Copyright (C) HPC WORLD (Prometech Software, Inc. and GDEP Solutions, Inc.) All rights reserve
!
! Permission is hereby granted, free of charge, to any person obtaining a
! copy of this software and associated documentation files (the "Software"),
! to deal in the Software without restriction, including without limitation
! the rights to use, copy, modify, merge, publish, distribute, sublicense,
! and/or sell copies of the Software, and to permit persons to whom the
! Software is furnished to do so, subject to the following conditions:
!
! The above copyright notice and this permission notice shall be included in
! all copies or substantial portions of the Software.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
! IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
! FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
! THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
! LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
! FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
! DEALINGS IN THE SOFTWARE.
!
program main
  use cublas_v2
  use cusolverDn
  use cudafor
  implicit none
  integer, parameter :: n=5, nrhs=3
  integer, parameter :: lda=n, ldb=n
  integer :: i
  real(8) :: a(lda,n)
  real(8) :: b(ldb, nrhs)
  real(8), allocatable :: workspace_d(:)
  integer :: devInfo_d
  integer :: istat, Lwork
  type(cusolverDnHandle) :: h

  data a /3.14, 0.00, 0.00, 0.00, 0.00, &
          0.17, 0.79, 0.00, 0.00, 0.00, &
         -0.90, 0.83, 4.53, 0.00, 0.00, &
          1.65,-0.65,-3.70, 5.32, 0.00, &
         -0.72, 0.28, 1.60,-1.37, 1.98/

  data b /-7.29, 9.25, 5.99,-1.94,-8.30, &
           6.11, 2.90,-5.05,-3.80, 9.66, &
           0.59, 8.88, 7.57, 5.57,-1.67/

  print *, "Matrix A"
  do i = 1, n
  write(*,"(5(f8.2,2x))") a(i,:)
  end do
  print *, "RHS"
  do i = 1, n
  write(*,"(5(f8.2,2x))") b(i,:)
  end do

  istat = cusolverDnCreate(h)
  if (istat /= CUSOLVER_STATUS_SUCCESS) &
      write(*,*) 'handle creation failed'

!$acc data copy(a,b) create(workspace_d)
  !$acc host_data use_device(a)
    istat = cusolverDnDpotrf_bufferSize(h, &
         CUBLAS_FILL_MODE_LOWER, n, a, n, Lwork)
  !$acc end host_data

    if (istat /= CUSOLVER_STATUS_SUCCESS) &
         write(*,*) 'cusolverDnDpotrf_buffersize failed'
    print *, "working space (bytes) =", Lwork
    allocate(workspace_d(Lwork))

  !$acc enter data copyin(workspace_d) copyin(devInfo_d)

  !$acc host_data use_device(a, b, workspace_d, devInfo_d)
    istat = cusolverDnDpotrf(h, CUBLAS_FILL_MODE_UPPER, &
             n, a, lda, workspace_d, Lwork, devInfo_d)
    istat = cusolverDnDpotrs(h, CUBLAS_FILL_MODE_UPPER, &
             n, nrhs, a, lda, b, ldb, devInfo_d)
  !$acc end host_data

    if (istat /= CUSOLVER_STATUS_SUCCESS) &
         write(*,*) 'cusolverDnDpotrs failed'

  !$acc exit data copyout(devInfo_d)
  !$acc exit data delete(workspace_d)
!$acc end data

  istat = devInfo_d
  if (istat /= 0) write(*,*) 'Cholesky Solver failed'

  istat = cusolverDnDestroy(h)
  if (istat /= CUSOLVER_STATUS_SUCCESS) &
      write(*,*) 'handle destruction failed'

  print *, "Cholesky factorization"
  do i = 1, n
  write(*,"(5(f8.2,2x))") a(i,:)
  end do
  print *, "Solution"
  do i = 1, n
  write(*,"(5(f8.2,2x))") b(i,:)
  end do
end program main
```

```
$ pgf90 -c -O2 -Minfo -ta=tesla,cuda9.0,cc35,cc60 -Mcudalib=cusolver,cublas -Kieee solver.f90
main:
     47, Loop not vectorized/parallelized: contains call
     48, Loop unrolled 5 times (completely unrolled)
     51, Loop not vectorized/parallelized: contains call
     52, Loop unrolled 3 times (completely unrolled)
     59, Generating copy(a(:,:))
         Generating create(workspace_d(:))
         Generating copy(b(:,:))
     70, Generating enter data copyin(devinfo_d,workspace_d(:))
     82, Generating exit data copyout(devinfo_d)
     83, Generating exit data delete(workspace_d(:))
     94, Loop not vectorized/parallelized: contains call
     95, Loop unrolled 5 times (completely unrolled)
     98, Loop not vectorized/parallelized: contains call
     99, Loop unrolled 3 times (completely unrolled)
$ pgf90 -o a.out solver.o -Mcuda -acc -ta=tesla,cuda9.0,cc35,cc60 -Mcudalib=cusolver,cublas
$a.out
 Matrix A
    3.14 0.17 -0.90 1.65 -0.72
    0.00 0.79 0.83 -0.65 0.28
    0.00 0.00 4.53 -3.70 1.60
    0.00 0.00 0.00 5.32 -1.37
    0.00 0.00 0.00 0.00 1.98
 RHS
   -7.29 6.11 0.59
    9.25 2.90 8.88
    5.99 -5.05 7.57
   -1.94 -3.80 5.57
   -8.30 9.66 -1.67
 working space (bytes) = 2
 Cholesky factorization
    1.77 0.10 -0.51 0.93 -0.41
    0.00 0.88 0.99 -0.84 0.36
    0.00 0.00 1.81 -1.32 0.57
    0.00 0.00 0.00 1.42 0.05
    0.00 0.00 0.00 0.00 1.16
 Solution
   -6.02 3.95 -3.14
   15.62 4.32 13.05
    3.02 -8.25 4.91
    3.25 -4.83 6.11
   -8.78 9.04 -3.57
```

Top of page

4. cuSolverDN: Linear system solver using LU decomposition (equivalent to LAPACK Dgetrs)

This is an example of using a double-precision linear solver (cusolverDnDgetrs) that uses LU decomposition to solve a Dense Matrix. Cholesky decomposition uses the cusolverDngetrf routine to solve a system of linear equations using cusolverDnSgetrs. Note that RHS (right side) is solved by giving multiple vectors. Note that the dense matrix A must be input in column-major order .

OpenACC Fortran + cusolverDnDpotrf, cusolverDnDpotrs (Solver.f90)

```
! Copyright (C) HPC WORLD (Prometech Software, Inc. and GDEP Solutions, Inc.) All rights reserve
!
! Permission is hereby granted, free of charge, to any person obtaining a
! copy of this software and associated documentation files (the "Software"),
! to deal in the Software without restriction, including without limitation
! the rights to use, copy, modify, merge, publish, distribute, sublicense,
! and/or sell copies of the Software, and to permit persons to whom the
! Software is furnished to do so, subject to the following conditions:
!
! The above copyright notice and this permission notice shall be included in
! all copies or substantial portions of the Software.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
! IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
! FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
! THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
! LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
! FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
! DEALINGS IN THE SOFTWARE.
!
program main
  use cublas_v2
  use cusolverDn
  use cudafor
  implicit none
  integer, parameter :: m=5, n=3, nrhs=2
  integer, parameter :: lda=m, ldb=m
  integer :: i, max_mn
  real(8) :: a(lda,n)
  real(8) :: b(ldb, nrhs)
  integer, allocatable :: devIpiv(:)
  real(8), allocatable :: workspace_d(:)
  integer :: devInfo_d
  integer :: istat, Lwork
  type(cusolverDnHandle) :: h

  data a /1.00, 2.00, 3.00, 4.0, 5.0,&
          1.00, 3.00, 5.00, 2.0, 4.0,&
          1.00, 4.00, 2.00, 5.0, 3.0/

  data b /-10.0, 12.00, 14.0, 16.0, 18.0,&
           -3.0, 14.0, 12.0, 16.0,16.0/

  print *, "Matrix A"
  do i = 1, m
  write(*,"(5(f8.2,2x))") a(i,:)
  end do
  print *, "RHS"
  do i = 1, m
  write(*,"(5(f8.2,2x))") b(i,:)
  end do

  max_mn = max (m, n)
  allocate (devIpiv(max_mn))

  istat = cusolverDnCreate(h)
  if (istat /= CUSOLVER_STATUS_SUCCESS) &
       write(*,*) 'handle creation failed'

!$acc data copy(a,b) create(workspace_d) copyout(devIpiv)
  !$acc host_data use_device(a)
    istat = cusolverDnDgetrf_bufferSize(h, &
         m, n, a, lda, Lwork)
  !$acc end host_data

    if (istat /= CUSOLVER_STATUS_SUCCESS) &
         write(*,*) 'cusolverDnDgetrf_buffersize failed'
    print *, "working space (bytes) =", Lwork
    allocate(workspace_d(Lwork))

  !$acc enter data copyin(workspace_d) copyin(devInfo_d)

  !$acc host_data use_device(a, b, workspace_d, devInfo_d, devIpiv)
    istat = cusolverDnDgetrf(h, &
             m, n, a, lda, workspace_d, devIpiv, devInfo_d)
    istat = cusolverDnDgetrs(h, CUBLAS_OP_N, &
             n, nrhs, a, lda, devIpiv, b, ldb, devInfo_d)
  !$acc end host_data

    if (istat /= CUSOLVER_STATUS_SUCCESS) &
         write(*,*) 'cusolverDnDgetrs failed'

  !$acc exit data copyout(devInfo_d)
  !$acc exit data delete(workspace_d)
!$acc end data

  istat = devInfo_d
  if (istat /= 0) write(*,*) 'LU Solver failed'

  istat = cusolverDnDestroy(h)
  if (istat /= CUSOLVER_STATUS_SUCCESS) &
       write(*,*) 'handle destruction failed'

  print *, "LU factorization"
  do i = 1, m
  write(*,"(5(f8.2,2x))") a(i,:)
  end do
  print *, "Pivot Indices"
  print *, devIpiv(:)
  print *, "Solution"
  do i = 1, m
  write(*,"(5(f8.2,2x))") b(i,:)
  end do
end program main
```

```
$ pgf90 -c -O2 -Minfo -ta=tesla,cuda9.0,cc35,cc60 -Mcudalib=cusolver,cublas -Kieee solver.f90
main:
     45, Loop not vectorized/parallelized: contains call
     46, Loop unrolled 3 times (completely unrolled)
     49, Loop not vectorized/parallelized: contains call
     50, Loop unrolled 2 times (completely unrolled)
     60, Generating copy(a(:,:))
         Generating copyout(devipiv(:))
         Generating create(workspace_d(:))
         Generating copy(b(:,:))
     71, Generating enter data copyin(devinfo_d,workspace_d(:))
     83, Generating exit data copyout(devinfo_d)
     84, Generating exit data delete(workspace_d(:))
     95, Loop not vectorized/parallelized: contains call
     96, Loop unrolled 3 times (completely unrolled)
    101, Loop not vectorized/parallelized: contains call
    102, Loop unrolled 2 times (completely unrolled)
pgf90 -o a.out solver.o -Mcuda -acc -ta=tesla,cuda9.0,cc35,cc60 -Mcudalib=cusolver,cublas
$a.out
 Matrix A
    1.00 1.00 1.00
    2.00 3.00 4.00
    3.00 5.00 2.00
    4.00 2.00 5.00
```

```
   5.00 4.00 3.00
RHS
 -10.00 -3.00
  12.00 14.00
  14.00 12.00
  16.00 16.00
  18.00 16.00
working space (bytes) = 18
LU factorization
   5.00 4.00 3.00
   0.60 2.60 0.20
   0.40 0.54 2.69
   0.80 -0.46 1.00
   0.20 0.08 0.14
Pivot Indices
           5  3  3  0  0
Solution
   2.00 1.20
   1.14 0.74
   1.14 2.34
  16.00 16.00
 -10.00 -3.00
```

Top of page

5. cuSolverDN: Eigenvalue solver for symmetric matrices (equivalent to LAPACK Dsyevd)

Use a solver (cusolverDnDsyevd) to find eigenvalues and eigenvectors for a symmetric dense matrix. The following example is the same process as D.1. Standard Symmetric Dense Eigenvalue Solvern written in Fortran OpenACC and C++ OpenACC.

Dense Linear Solver - cusolverDnDsyevd with OpenACC Fortran

Compute the eigenvalues and eigenvectors of a symmetric (Hermitian) n×n matrix A. The standard symmetric eigenvalue problem is expressed by the following formula.

$$A * V = V * \Lambda$$

where Λ is the actual n×n diagonal matrix. V is an n×n unitary matrix. The diagonal elements of Λ are the ascending eigenvalues of A. The cusolverDnDsyevd solver needs to provide a work space (d_work) in advance, so it is necessary to use the syevd_bufferSize function to find the lwork size. If output parameter devInfo = -i (less than zero), it indicates that the i-th parameter is incorrect. If devInfo = i (greater than zero), it means that i off-diagonal elements of the tridiagonal did not converge to zero during internal calculation. Note that the dense matrix A must be input in column-major order .

OpenACC + cusolverDnDsyevd (syevd.f90)

```
! Copyright (C) HPC WORLD (Prometech Software, Inc. and GDEP Solutions, Inc.) All rights reserve
!
! Permission is hereby granted, free of charge, to any person obtaining a
! copy of this software and associated documentation files (the "Software"),
! to deal in the Software without restriction, including without limitation
! the rights to use, copy, modify, merge, publish, distribute, sublicense,
! and/or sell copies of the Software, and to permit persons to whom the
! Software is furnished to do so, subject to the following conditions:
!
```

```
! The above copyright notice and this permission notice shall be included in
! all copies or substantial portions of the Software.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
! IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
! FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
! THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
! LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
! FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
! DEALINGS IN THE SOFTWARE.

! pgf90 -acc -O2 -Minfo -ta=tesla,cc60,cc35 -Mcuda -Mcudalib=cusolver syevd.f90

program main
  use cusolverDn
  implicit none
  integer,parameter :: prc = SELECTED_REAL_KIND(15,305)

  integer, parameter :: idbg = 1 ! debug write enable
  integer, parameter :: m = 3, lda =m
  type(cusolverDnHandle) :: cusolver_H
  integer :: cusolver_status
  integer :: status1, ierr_code
  integer :: i, j

! | 3.5 0.5 0 |
! A = | 0.5 3.5 0 |
! | 0 0 2 |
!
! lambda = (/ 2.0, 3.0, 4.0 /) exact eigenvalues

  real(prc), allocatable, dimension(:) :: A
  real(prc), allocatable, dimension(:) :: lambda


  real(prc), allocatable, dimension(:) :: V ! eigenvectors
  real(prc), allocatable, dimension(:) :: W ! eigenvaluse
  real(prc), allocatable, dimension(:) :: d_work
  integer::devInfo
  integer :: Lwork

  allocate ( A(lda*m), V(lda*m), W(m) )
  allocate ( lambda(m) )

  Lwork = 0

! set a data
  A(1:lda*m) = (/3.5, 0.5, 0, 0.5, 3.5, 0, 0, 0, 2.0/)

! step 1 create cusolver/cublas handle
  cusolver_status = cusolverDnCreate(cusolver_H)
    if (idbg == 1) then
      if (cusolver_status /= CUSOLVER_STATUS_SUCCESS) print *, "step1 cusolver_status = ", cusol
    end if

!$acc data copy(A, devInfo) copyout(W)
!$acc host_data use_device(A, W)

! step 2 working space of syevd
  cusolver_status = cusolverDnDsyevd_bufferSize (cusolver_H, CUSOLVER_EIG_MODE_VECTOR, &
                    CUSOLVER_EIG_MODE_VECTOR, m, A, lda, W, Lwork)
!$acc end host_data

    if (idbg == 1) then
      if (cusolver_status /= CUSOLVER_STATUS_SUCCESS) print *, "step2 cusolver_status = ", cusol
    end if

! step 3 d_work buffer allocated
```

```
    allocate(d_work(Lwork), stat=ierr_code) ! words
      if (idbg == 1) print *, "step5: alloc ierr_code = ", ierr_code

!$acc enter data copyin(d_work)

! step 4 compute spectrum = eigen{vectors, valuse}
! $acc host_data use_device(A, W, d_work, devInfo)
  cusolver_status = cusolverDnDsyevd (cusolver_H, CUSOLVER_EIG_MODE_VECTOR, &
                    CUSOLVER_EIG_MODE_VECTOR, m, A, lda, W, d_work, Lwork, devInfo)
!$acc end host_data
!$acc exit data delete(d_work)
!$acc end data

  print *, "Output parameter devInfo =", devInfo

    if (idbg == 1) then
      if (cusolver_status /= CUSOLVER_STATUS_SUCCESS) print *, "step4 cusolver_status = ", cusol
    end if

! copy [A].output to [V] (eigenvectors)
  V = A

! atep 5 print out
  !Eigenvalues
  print *, "Eigenvalue : "
  do i = 1, m
    print *, "W(",i,") = ", w(i)
  end do

  !Eigenvectors
  print *, "Eigenvector : "
  do i = 1, m
    do j = 1, m
       print '(1x,a,i5,a,i5,1x,a4,F15.7)', "(",i,",",j,") = ", V(i+(j-1)*lda )
    end do
  end do


  deallocate(A, V, W, d_work)

  end program main
```

It is necessary to compile and link with the -acc -Mcudalib=cusolver -Mcuda option   as a compile option .

Compile & run

```
[kato@photon32 Dens_Eigen]$ pgf90 -acc -O2 -Minfo -ta=tesla,cc60,cc35 -Mcuda -Mcudalib=cusolver
main:
     57, Generated vector simd code for the loop
         Residual loop unrolled 1 times (completely unrolled)
     65, Generating copy(a(:))
         Generating copyout(w(:))
         Generating copy(devinfo)
     81, Generating enter data copyin(d_work(:))
     88, Generating exit data delete(d_work(:))
     98, Generated vector simd code for the loop
         Residual loop unrolled 1 times (completely unrolled)
    103, Loop not vectorized/parallelized: contains call
    110, Loop not vectorized/parallelized: contains call
[kato@photon32 Dens_Eigen]$ a.out
 step5: alloc ierr_code = 0
 Output parameter devInfo = 0
 Eigenvalue:
 W( 1 ) = 2.000000000000000
 W( 2 ) = 3.000000000000000
 W( 3 ) = 4.000000000000000
 Eigenvector:
 ( 1, 1 ) = 0.0000000
 ( 1, 2 ) = -0.7071068
 ( 1, 3 ) = 0.7071068
 (2, 1) = 0.0000000
 (2, 2) = 0.7071068
 (2, 3) = 0.7071068
 (3, 1) = 1.0000000
 (3, 2) = 0.0000000
 (3, 3) = 0.0000000
```

Top of page

Dense Linear Solver - cusolverDnDsyevd with OpenACC C++

Rewrite the above Fortran program using PGI C++ OpenACC.

OpenACC + cusolverDnDsyevd (syevd.cpp)

```
! Copyright (C) HPC WORLD (Prometech Software, Inc. and GDEP Solutions, Inc.) All rights reserve
!
! Permission is hereby granted, free of charge, to any person obtaining a
! copy of this software and associated documentation files (the "Software"),
! to deal in the Software without restriction, including without limitation
! the rights to use, copy, modify, merge, publish, distribute, sublicense,
! and/or sell copies of the Software, and to permit persons to whom the
! Software is furnished to do so, subject to the following conditions:
!
! The above copyright notice and this permission notice shall be included in
! all copies or substantial portions of the Software.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
! IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
! FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
! THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
! LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
! FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
! DEALINGS IN THE SOFTWARE.

/*
 * How to compile (assume cuda is installed at /usr/local/cuda-9.0/)
 * pgc++ -I/usr/local/cuda-9.0/include syevd.cpp -acc -ta=tesla,cc60,cc35 -Mcudalib=cusolver -Mi
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <cusolverDn.h>

void printMatrix(int m, int n, const double*A, int lda, const char* name)
{
    for(int row = 0 ; row < m ; row++){
        for(int col = 0 ; col < n ; col++){
            double Areg = A[row + col*lda];
            printf("%s(%d,%d) = %f\n", name, row+1, col+1, Areg);
        }
    }
}

int main(int argc, char*argv[])
```

```
{
    cusolverDnHandle_t cusolverH = NULL;
    cusolverStatus_t cusolver_status = CUSOLVER_STATUS_SUCCESS;
    const int m = 3;
    const int lda = m;

/* | 3.5 0.5 0 |
 * A = | 0.5 3.5 0 |
 * | 0 0 2 |
 *
 */
// double lambda[m] = { 2.0, 3.0, 4.0}; /* Exact eigenvalues */
    double *A; // ibput
    double *V; // eigenvectors
    double *W; // eigenvalues

    int *devInfo = NULL;
    double *work = NULL;
    int lwork = 0;

    A = (double *) malloc (sizeof(double) * lda * m);
    V = (double *) malloc (sizeof(double) * lda * m);
    W = (double *) malloc (sizeof(double) * m);
    devInfo = (int *) malloc (sizeof(int));

// step 1: create cusolver/cublas handle
    cusolver_status = cusolverDnCreate(&cusolverH);

// step 2: set A
    A[0] = 3.5; A[1] = 0.5; A[2] = 0.0;
    A[3] = 0.5; A[4] = 3.5; A[5] = 0.0;
    A[6] = 0.0; A[7] = 0.0; A[8] = 2.0;

    printMatrix(m, m, A, lda, "A");
    printf("=====\n");

// step 3: query working space of syevd
    cusolverEigMode_t jobz = CUSOLVER_EIG_MODE_VECTOR; // compute eigenvalues and eigenvectors.
    cublasFillMode_t uplo = CUBLAS_FILL_MODE_LOWER;

#pragma acc data copy(A[0:lda*m], devInfo[0:1]) copyout(W[0:m])
{
#pragma acc host_data use_device(A, W)
  {
    cusolver_status = cusolverDnDsyevd_bufferSize(
        cusolverH,
        jobz,
        uplo,
        m,
        A,
        lda,
        W,
        &lwork);
  }
    assert (cusolver_status == CUSOLVER_STATUS_SUCCESS);

    work = (double *) malloc(sizeof(double)*lwork);
    printf("Lwork = %d\n",lwork);

#pragma acc enter data create(work[0:lwork])

// step 4: compute spectrum
#pragma acc host_data use_device(A, W, work, devInfo)
  {
    cusolver_status = cusolverDnDsyevd(
        cusolverH,
        jobz,
        uplo,
          m,
          A,
          lda,
          W,
          work,
          lwork,
          devInfo);
   }
#pragma acc wait
#pragma acc exit data delete(work)
}
    printf("DevInfo = %d\n",devInfo[0]);
    printf("eigenvalue = (base 1), ascending order\n");
    for(int i = 0 ; i < m ; i++){
        printf("W[%d] = %E\n", i+1, W[i]);
    }

// print eigenvecvtors
    for(int i = 0 ; i < m*m ; i++){
      V[i] = A[i];
    }
    printf("V = (base 1)\n");
    printMatrix(m, m, V, lda, "V");
    printf("=====\n");

// free resources
    if (A) free(A);
    if (W) free(W);
    if (devInfo) free(devInfo);
    if (work) free(work);

    if (cusolverH) cusolverDnDestroy(cusolverH);

// cudaDeviceReset();

    return 0;
}
```

If you want to use cuSOLVER with the PGI C/C++ OpenACC compiler, compile using the C/C++ include files bundled with the NVIDIA CUDA Toolkit (the PGI compiler includes cuSOLVER/cuBLAS libraries for C/C++). include files such as are not bundled). The example here is written assuming that CUDA 9.0 is implemented under /usr/local/cuda-9.0. It is necessary to compile and link with the -I/usr/local/cuda-9.0/include -acc -ta=tesla -Mcudalib=cusolver option as a compile option.
(For CUDA 8.0)
pgc++ -I/usr/local/cuda-8.0/include syevd.cpp -acc -ta=tesla,cc60,cc35 -Mcudalib=cusolver -Minfo=accel -O2
(For CUDA 9.0)
pgc++ - I/usr/local/cuda-9.0/include syevd.cpp -acc -ta=tesla,cc60,cc35 -Mcudalib=cusolver -Minfo=accel -O2

Compile & run

```
[kato@photon32 C]$ pgc++ -I/usr/local/cuda-9.0/include syevd.cpp -acc -ta=tesla,cc60,cc35
-Mcudalib=cusolver -Minfo=accel -O2
main:
     65, Generating copyout(W[:3])
         Generating copy(devInfo[:1],A[:9])
     87, Generating enter data create(work[:lwork])
    103, Generating update self(devInfo[:1])
         Generating exit data delete(work[:1])
[kato@photon32 C]$ a.out
```

```
A(1,1) = 3.500000
A(1,2) = 0.500000
A(1,3) = 0.000000
A(2,1) = 0.500000
A(2,2) = 3.500000
A(2,3) = 0.000000
A(3,1) = 0.000000
A(3,2) = 0.000000
A(3,3) = 2.000000
=====
Lwork = 83
DevInfo = 0
eigenvalue = (base 1), ascending order
W[1] = 2.000000E+00
W[2] = 3.000000E+00
W[3] = 4.000000E+00
V = (base 1)
V(1,1) = 0.000000
V(1,2) = -0.707107
V(1,3) = 0.707107
V(2,1) = 0.000000
V(2,2) = 0.707107
V(2,3) = 0.707107
V(3,1) = 1.000000
V(3,2) = 0.000000
V(3,3) = 0.000000
=====
```

Top of page

6. cuSolverDN: Dense matrix linear solver using QR decomposition (equivalent to LAPACK Dgeqrf)

An example of using QR decomposition to solve a dense matrix linear system is shown. The following example is the same process as C.1. QR Factorization Dense Linear Solver written in Fortran OpenACC. Note that the dense matrix A must be input in column-major order .

QR Factorization Dense Linear Solver with OpenACC Fortran

Solve the following linear system using QR decomposition.

$$Ax = b$$

The program below shows an example of a 3 x 3 dense matrix (regular matrix).

$$A = \begin{pmatrix} 1.0 & 2.0 & 3.0 \\ 4.0 & 5.0 & 6.0 \\ 2.0 & 1.0 & 1.0 \end{pmatrix}$$

Use cuSOLVE routines (geqrf,ormqr) and cuBLAS routines (trsm).

Step 1: A = Q*R by geqrf.
Step 2: B := Q^T*B by ormqr.
Step 3: solve R*X = B by trsm.

OpenACC + cuSOLVE-geqrf,ormqr,trsm (ormqr.f90)

```
! Copyright (C) HPC WORLD (Prometech Software, Inc. and GDEP Solutions, Inc.) All rights reserve
!
```

```
! Permission is hereby granted, free of charge, to any person obtaining a
! copy of this software and associated documentation files (the "Software"),
! to deal in the Software without restriction, including without limitation
! the rights to use, copy, modify, merge, publish, distribute, sublicense,
! and/or sell copies of the Software, and to permit persons to whom the
! Software is furnished to do so, subject to the following conditions:
!
! The above copyright notice and this permission notice shall be included in
! all copies or substantial portions of the Software.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
! IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
! FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
! THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
! LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
! FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
! DEALINGS IN THE SOFTWARE.

! pgf90 -acc -O2 -Minfo=accel -ta=tesla,cc60,cc35 -Mcudalib=cusolver,cublas
!-Mcuda ormqr.f90

program main
  use cublas_v2
  use cusolverDn
  implicit none
  integer,parameter :: prc = SELECTED_REAL_KIND(15,305)

  integer, parameter :: idbg = 1 ! debug write enable
  integer, parameter :: m = 3, lda =m, ldb = m
  integer, parameter :: nrhs = 1 ! number of right hand side vectors
  real(prc) :: one = 1.0d0
  type(cusolverDnHandle) :: cusolver_H
  type(cublasHandle) :: cublas_H
  integer :: cusolver_status
  integer :: cublas_status
  integer :: status1, ierr_code
  integer :: i, j

! | 1 2 3 |
! A = | 4 5 6 |
! | 2 1 1 |
!
! x = (1 1 1) exact solution
! b = (6 15 4)

  real(prc), allocatable, dimension(:) :: A
  real(prc), allocatable, dimension(:) :: B
  real(prc), allocatable, dimension(:) :: Xc ! solution matrix from GPU

  real(prc), allocatable, dimension(:) :: tau
  real(prc), allocatable, dimension(:) :: d_work
  integer::devInfo
  integer :: Lwork

  allocate ( A(lda*m), B(ldb*nrhs), Xc(ldb*nrhs) )
  allocate ( tau(m) )

  Lwork = 0

! set a data
  A(1:lda*m) = (/1.0, 4.0, 2.0, 2.0, 5.0, 1.0, 3.0, 6.0, 1.0/)
  B(1:ldb*nrhs) = (/6.0, 15.0, 4.0/)

! print A
  print *, "A ======"
  do i = 1, m
    do j = 1, m
      print '(1x,a,i5,a,i5,1x,a4,F15.7)', "(",i,",",j,") = ", A(i+(j-1)*lda )
    end do
  end do
! print B
  print *, "B ======"
  do i = 1, ldb
    do j = 1, 1
       print '(1x,a,i5,a,i5,1x,a4,F15.7)', "(",i,",",j,") = ", B(i)
    end do
  end do

! step 1 create cusolver/cublas handle
  cusolver_status = cusolverDnCreate(cusolver_H)
  cublas_status = cublasCreate(cublas_H)
    if (idbg == 1) then
      if (cusolver_status /= CUSOLVER_STATUS_SUCCESS) print *, "step1 cusolver_status = ", cusol
      if (cublas_status /= CUBLAS_STATUS_SUCCESS ) print *, "step1 cublas_status = ", cublas_sta
    end if

!$acc data copy(A, B, devInfo) create(tau)

! step 2 query working space of geqrf and ormqr
!$acc host_data use_device(A)
  cusolver_status = cusolverDnDgeqrf_bufferSize(cusolver_H, &
                    m, m, A, lda, Lwork)
!$acc end host_data

    if (idbg == 1) then
      if (cusolver_status /= CUSOLVER_STATUS_SUCCESS) print *, "step2 cusolver_status = ", cusol
    end if

! step 3 d_work buffer allocated
    allocate(d_work(Lwork), stat=ierr_code) ! words
      if (idbg == 1) then
        if (ierr_code /= 0) print *, "step3: alloc ierr_code = ", ierr_code
      end if
    print *, "Lwork = ", Lwork

!$acc enter data copyin(d_work)

! step 4 compute QR factorization
!$acc host_data use_device(A, tau, d_work, devInfo)
  cusolver_status = cusolverDnDgeqrf(cusolver_H, &
                    m, m, A, lda, tau, d_work, Lwork, devInfo)
!$acc end host_data
!$acc update self(devInfo)
  print *, "after geqrf: devInfo =", devInfo

! step 5: compute Q^T*B
!$acc host_data use_device(A, B, tau, d_work, devInfo)
  cusolver_status = cusolverDnDormqr(cusolver_H, CUBLAS_SIDE_LEFT, CUBLAS_OP_T, &
                    m, nrhs, m, A, lda, tau, B, ldb, d_work, Lwork, devInfo)
!$acc end host_data

!$acc update self(devInfo)

    print *, "after ormqr: devInfo =", devInfo
    if (idbg == 1) then
      if (cusolver_status /= CUSOLVER_STATUS_SUCCESS) print *, "step5 cusolver_status = ", cusol
    end if

! step 6: compute x = R \ Q^T*B
!$acc host_data use_device(A, B)
  cublas_status = cublasDtrsm_v2(cublas_H, CUBLAS_SIDE_LEFT, CUBLAS_FILL_MODE_UPPER, CUBLAS_OP_N
                   CUBLAS_DIAG_NON_UNIT, m, nrhs, one, A, lda, B, ldb)
!$acc end host_data

!$acc exit data delete(tau, d_work)
!$acc end data
```

```
    if (idbg == 1) then
      if (cublas_status /= CUBLAS_STATUS_SUCCESS) print *, "step6 cublas_status = ", cublas_stat
    end if

! copy [B].output to result [Xc]
  Xc = B

! atep 5 print out

  print *, "Result ===================== "
  do i = 1, m
    do j = 1, nrhs
       print '(1x,a,i5,a,i5,1x,a4,F15.7)', "(",i,",",j,") = ", Xc(i+(j-1)*ldb )
    end do
  end do

  deallocate(A, B, Xc, tau, d_work)

  end program main
```

It is necessary to compile and link with the -acc -Mcudalib=cusolver,cublas -Mcuda -ta=tesla option as a compile option.

Compile & run

```
[kato@photon32 QRf]$ pgf90 -acc -O2 -Minfo=accel -ta=tesla,cc60,cc35,cuda8.0
-Mcudalib=cusolver,cublas -Mcuda ormqr.f90
main:
     88, Generating copy(a(:),b(:))
         Generating create(tau(:))
         Generating copy(devinfo)
    107, Generating enter data copyin(d_work(:))
    114, Generating update self(devinfo)
    123, Generating update self(devinfo)
    136, Generating exit data delete(tau(:),d_work(:))
[kato@photon32 QRf]$ a.out
 A ======
 ( 1, 1 ) = 1.0000000
 ( 1, 2 ) = 2.0000000
 ( 1, 3 ) = 3.0000000
 (2, 1) = 4.0000000
 (2, 2) = 5.0000000
 (2, 3) = 6.0000000
 (3, 1) = 2.0000000
 (3, 2) = 1.0000000
 (3, 3) = 1.0000000
 B ======
 ( 1, 1 ) = 6.0000000
 (2, 1) = 15.0000000
 ( 3, 1 ) = 4.0000000
 Lwork = 15
 after geqrf: devInfo = 0
 after ormqr: devInfo = 0
 Result =====================
 ( 1, 1 ) = 1.0000000
 (2, 1) = 1.0000000
 (3, 1) = 1.0000000
```

Top of page

7. cuSolverDN: Orthogonalization by QR decomposition (equivalent to LAPACK

Dgeqrf,Dorgqr)

An example of orthogonalization calculation using QR decomposition is shown. The following example is the same process as C.2. orthogonalization written in Fortran OpenACC. Note that the dense matrix A must be input in column-major order .

Orthgonalization by QR Factorization with OpenACC Fortran

Use CUDA Library to calculate "orthogonalization" by QR decomposition.

$$A = Q * R$$

The program below shows an example of a 3 x 2 dense matrix.

$$A = \begin{pmatrix} 1.0 & 2.0 \\ 4.0 & 5.0 \\ 2.0 & 1.0 \end{pmatrix}$$

Use cuSOLVE routines (geqrf, orgqr).

Step 1: A = Q*R by geqrf.
Step 2: form Q by orgqr.
Step 3: check if Q is unitary or not.

OpenACC + cuSOLVE-geqrf,orgqr (ortho.f90)

```
! Copyright (C) HPC WORLD (Prometech Software, Inc. and GDEP Solutions, Inc.) All rights reserve
!
! Permission is hereby granted, free of charge, to any person obtaining a
! copy of this software and associated documentation files (the "Software"),
! to deal in the Software without restriction, including without limitation
! the rights to use, copy, modify, merge, publish, distribute, sublicense,
! and/or sell copies of the Software, and to permit persons to whom the
! Software is furnished to do so, subject to the following conditions:
!
! The above copyright notice and this permission notice shall be included in
! all copies or substantial portions of the Software.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
! IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
! FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
! THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
! LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
! FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
! DEALINGS IN THE SOFTWARE.

! pgf90 -acc -O2 -Minfo=accel -ta=tesla,cc60,cuda8.0 -Mcudalib=cusolver,cublas
! -Mcuda ormqr.f90 -Mcuda

program main
  use cublas_v2
  use cusolverDn
  implicit none
  integer,parameter :: prc = SELECTED_REAL_KIND(15,305)

  integer, parameter :: idbg = 1 ! debug write enable
  integer, parameter :: m = 3, n = 2, lda =m
  real(prc) :: beta = 1.0d0, alpha = -1.0d0
  type(cusolverDnHandle) :: cusolver_H
  type(cublasHandle) :: cublas_H
  integer :: cusolver_status, cusolver_status2
  integer :: cublas_status
  integer :: status1, ierr_code
  integer :: i, j

! | 1 2 |
! A = | 4 5 |
! | 2 1 |
!
! x = (1 1 1) exact solution
! b = (6 15 4)

  real(prc), allocatable, dimension(:) :: A
  real(prc), allocatable, dimension(:) :: Q ! orthonormal columns
  real(prc), allocatable, dimension(:) :: R ! R = I - Q**T*Q
  real(prc) :: R_nrm2

  real(prc), allocatable, dimension(:) :: tau
  real(prc), allocatable, dimension(:) :: d_work
  integer::devInfo
  integer :: Lwork

  integer :: Lwork_geqrf
  integer :: Lwork_orgqr

  allocate ( A(lda*n), Q(lda*n), R(n*n) )
  allocate ( tau(m) )

  Lwork = 0
  Lwork_geqrf = 0
  Lwork_orgqr = 0

! set a data
  A(1:lda*n) = (/1.0, 4.0, 2.0, 2.0, 5.0, 1.0/)

! print A
  print *, "A ======"
  do i = 1, m
    do j = 1, n
       print '(1x,a,i5,a,i5,1x,a4,F15.7)', "(",i,",",j,") = ", A(i+(j-1)*lda )
    end do
  end do

! step 1 create cusolver/cublas handle
  cusolver_status = cusolverDnCreate(cusolver_H)
  cublas_status = cublasCreate(cublas_H)
    if (idbg == 1) then
      if (cusolver_status /= CUSOLVER_STATUS_SUCCESS) print *, "step1 cusolver_status = ", cusol
      if (cublas_status /= CUBLAS_STATUS_SUCCESS ) print *, "step1 cublas_status = ", cublas_sta
    end if

!$acc data copy(A, devInfo) create(R, tau)

! step 2 query working space of geqrf and ormqr
!$acc host_data use_device(A)
  cusolver_status = cusolverDnDgeqrf_bufferSize(cusolver_H, &
                    m, n, A, lda, Lwork_geqrf)
  cusolver_status2= cusolverDnDorgqr_bufferSize(cusolver_H, &
                    m, n, n, A, lda, tau, Lwork_orgqr)
!$acc end host_data

    Lwork = max(Lwork_geqrf, Lwork_orgqr)
    print *, "Lwork_geqrf, Lwork_orgqr, Lwork = ", Lwork_geqrf, Lwork_orgqr, Lwork

    if (idbg == 1) then
      if (cusolver_status /= CUSOLVER_STATUS_SUCCESS) print *, "step2 cusolver_status = ", cusol
      if (cusolver_status2/= CUSOLVER_STATUS_SUCCESS) print *, "step2 cusolver_status2= ", cusol
    end if

! step 3 d_work buffer allocated
```

```
    allocate(d_work(Lwork), stat=ierr_code) ! words
    if (idbg == 1) then
      if (ierr_code /= 0) print *, "step3: alloc ierr_code = ", ierr_code
    end if

!$acc enter data copyin(d_work)

! step 4 compute QR factorization
!$acc host_data use_device(A, tau, d_work, devInfo)
    cusolver_status = cusolverDnDgeqrf(cusolver_H, &
                        m, n, A, lda, tau, d_work, Lwork, devInfo)
!$acc end host_data
!$acc update self(devInfo)
    print *, "after geqrf: devInfo =", devInfo

! step 5: compute Q
!$acc host_data use_device(A, tau, d_work, devInfo)
    cusolver_status = cusolverDnDorgqr(cusolver_H, &
                        m, n, n, A, lda, tau, d_work, Lwork, devInfo)
!$acc end host_data

!$acc update self(A, devInfo)

    print *, "after orgqr: devInfo =", devInfo
    if (idbg == 1) then
      if (cusolver_status /= CUSOLVER_STATUS_SUCCESS) print *, "step5 cusolver_status = ", cusol
    end if

! step 6: copy A to Q ... orthonormal columns
    Q = A

! print Q
    print *, "Q =========="
    do i = 1, m
      do j = 1, n
         print '(1x,a,i5,a,i5,1x,a4,F15.7)', "(",i,",",j,") = ", Q(i+(j-1)*lda )
      end do
    end do

! step 7: measure R = I - Q**T*Q
    R = 0.0
    do i = 1, n
      R( i + n*(i-1) ) = 1.0 ! R(i,i) = 1
    end do
!$acc update device(R)

! R = -Q**T*Q + I (Q**T = CUBLAS_OP_T, Q = CUBLAS_OP_T)
!$acc host_data use_device(A, R)
    cublas_status = cublasDgemm_v2(cublas_H, CUBLAS_OP_T, CUBLAS_OP_N, &
                    n, n, m, alpha, A, lda, A, lda, beta, R, n)
!$acc end host_data
!$acc update self(R)

    if (idbg == 1) then
      if (cublas_status /= CUBLAS_STATUS_SUCCESS) print *, "step7 cublas_status = ", cublas_stat
    end if

    R_nrm2 = 0.0d0
!$acc host_data use_device(R)
    cublas_status = cublasDnrm2_v2(cublas_H, n*n, R, 1, R_nrm2)
!$acc end host_data

!$acc exit data delete(tau, d_work)
!$acc end data

! atep 5 print |I - Q**T*Q|

    print *, "Result ==== |I - Q**T*Q| === "
    print '(1x, a, 2x, D16.6)', '|I - Q**T*Q| =', R_nrm2

    deallocate(A, R, tau, d_work)

  end program main
```

It is necessary to compile and link with the -acc -Mcudalib=cusolver,cublas -Mcuda -ta=tesla option as a compile option.

Compile & run

```
[kato@photon32 Ortho]$ pgf90 -acc -Minfo=accel -ta=tesla,cc60,cc35,cuda8.0
-Mcudalib=cusolver,cublas ortho.f90 -Mcuda
main:
     85, Generating copy(a(:),devinfo)
         Generating create(r(:),tau(:))
    109, Generating enter data copyin(d_work(:))
    116, Generating update self(devinfo)
    125, Generating update self(a(:),devinfo)
    148, Generating update device(r(:))
    155, Generating update self(r(:))
    166, Generating exit data delete(tau(:),d_work(:))
[kato@photon32 Ortho]$ a.out
 A ======
 ( 1, 1 ) = 1.0000000
 ( 1, 2 ) = 2.0000000
 (2, 1) = 4.0000000
 (2, 2) = 5.0000000
 (3, 1) = 2.0000000
 (3, 2) = 1.0000000
 Lwork_geqrf, Lwork_orgqr, Lwork = 14 2 14
 after geqrf: devInfo = 0
 after orgqr: devInfo = 0
 Q ==========
 ( 1, 1 ) = -0.2182179
 (1, 2) = 0.5345225
 ( 2, 1 ) = -0.8728716
 (2, 2) = 0.2672612
 ( 3, 1 ) = -0.4364358
 ( 3, 2 ) = -0.8017837
 Result ==== |I - Q**T*Q| ===
 |I - Q**T*Q| = 0.450975D-15
```

Top of page

8. cuSolverDN: Singular value decomposition including singular vector calculation (by QR) (equivalent to LAPACK Dgesvd)

This is an example of performing singular value decomposition on a matrix whose components are complex numbers or real numbers. Here, we will show an example that deals with real numbers. Corresponds to **LAPACK's dgesvd routine .** Singular value decomposition is used in the fields of signal processing and statistics. The example below is the same process as F.1. SVD with singular vectors written in Fortran OpenACC. Note that the dense matrix A must be input in column-major order .

The program below shows an example of a 3 x 2 dense matrix. SVD is expressed by the following formula. Here Σ is an m×n matrix that is zero except for its min(m,n) diagonal elements, U is an m×m unitary matrix, and V is an n×n unitary matrix. The diagonal elements of Σ are singular values of A. They are real numbers, not negative numbers, and are returned in descending order. The first min (m,

n) columns of U and V are the left and right singular vectors of A. The cusolverDnDgesvd routine uses the QR algorithm. There is also cusolverDngesvdj, which has the same functionality. However, the latter is solved using the Jacobian method . If you are targeting small or medium-sized matrices, you may be able to enjoy high GPU performance due to the parallelism of the Jacobian method. Furthermore, using the Jacobi method is convenient for obtaining an approximate solution up to a certain level of accuracy. Here, we will show an example of singular value decomposition (SVD), which is solved using the QR algorithm.

$$A = U * \Sigma * V^H$$

$$A = \begin{pmatrix} 1.0 & 2.0 \\ 4.0 & 5.0 \\ 2.0 & 1.0 \end{pmatrix}$$

Implement the following three calculation steps.

Step 1: Perform A = U*S*VT singular value decomposition.
Step 2: Verify the singular values.
Step 3: Calculate the residual error AU*S*VT using cuBLAS.

When calculating the residual error in Step 3, use the BLAS NVIDIA extention function. Since this is not included in PGI Fortran's Interface Module, I created a module called module cuBLASext to interface Fortran with the cuBLAS function.

OpenACC + cuSOLVER-cusolverDnDgesvd (svd_simgular.f90)

```
! Copyright (C) HPC WORLD (Prometech Software, Inc. and GDEP Solutions, Inc.) All rights reserve
! Copyright (c) 2017, NVIDIA CORPORATION. All rights reserved.
!
!
! Permission is hereby granted, free of charge, to any person obtaining a
! copy of this software and associated documentation files (the "Software"),
! to deal in the Software without restriction, including without limitation
! the rights to use, copy, modify, merge, publish, distribute, sublicense,
! and/or sell copies of the Software, and to permit persons to whom the
! Software is furnished to do so, subject to the following conditions:
!
! The above copyright notice and this permission notice shall be included in
! all copies or substantial portions of the Software.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
! IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
! FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
! THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
! LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
! FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
! DEALINGS IN THE SOFTWARE.
!
module cuBLASext
! NVIDIA extension : the BLAS-extension functions
! because PGI Fortran don't include NVIDIA extention BLAS module.
use cublas_v2
interface cublasDdgmm
  integer(c_int) function cublasDdgmm( handle, side_mode, m, n, &
                                       A, lda, x , incx , C, ldc) &
                                       bind(C,name="cublasDdgmm")
  use iso_c_binding
  import cublasHandle
  type(cublasHandle), value :: handle
  integer(c_int), value :: side_mode, m, n, lda, ldc, incx
  real(c_double), device :: A(*), C(*)
  real(c_double), device :: x(*)
  end function cublasDdgmm
end interface cublasDdgmm
end module cuBLASext

module aux_routine
contains

subroutine print_D_matrix(m, n, a, lda)
integer :: m, n, i, j
real(8) :: a(*)

do i=1, m
  do j=1, n
    print "(5(,i5,1x,i5,2x,f10.6,2x))", i, j, a(i+(j-1)*lda)
  end do
end do
end subroutine print_D_matrix
end module aux_routine

program main
  use cublas_v2
  use cuBLASext
  use cusolverDn
  use aux_routine
  implicit none
  integer, parameter :: m=3, n=2
  integer, parameter::lda = m
  integer, parameter :: idbg = 1
  real(8), parameter :: one = 1.d0, minus_one = -1.d0
  real(8) :: a (lda*n) ! column-major format with dimensions mxn
  real(8) :: u (lda*m) ! m by m unitary matrix
  real(8) :: vt(lda*n) ! n by n unitary matrix
  real(8) :: w (lda*n) ! w = s*vt
  real(8), allocatable, dimension(:) :: s ! singular values
  real(8), allocatable, dimension(:) :: rwork ! min(m,n)-1
  real(8), allocatable :: workspace_d(:)
  real(8) :: S_exact(n)
  real(8) :: err, diff_s, diff_fro
  integer :: devInfo_d
  integer :: istat, Lwork
  integer :: i
  character*1 :: jobu, jobvt
  type(cusolverDnHandle) :: cusolver_H
  type(cublasHandle) :: cublas_H
  integer :: cusolver_status
  integer :: cublas_status

! | 1 2 |
! a = | 4 5 |
! | 2 1 |
!

  allocate(s(1:min(m,n)))
  allocate(rwork(1:min(m,n)-1))
  Lwork = 0

! set a data
  a(1:lda*n) = (/1.0, 4.0, 2.0, 2.0, 5.0, 1.0/)

! Exact SV numbers
  S_exact(1:n) =(/7.065283497082729d0, 1.040081297712078d0/)

  cusolver_status = cusolverDnCreate(cusolver_H)
```

```
    if (cusolver_status /= CUSOLVER_STATUS_SUCCESS) &
         print *, 'handle creation failed'
    cublas_status = cublasCreate(cublas_H)
    if (cublas_status /= CUBLAS_STATUS_SUCCESS ) &
         print *, "cublas_status = ", cublas_status

  !$acc data copy(a) create(s, u, vt, devInfo_d, workspace_d, rwork, w)

        ! Step 1: compute A = U*S*VT

    !$acc host_data use_device(a)
          istat = cusolverDnDgesvd_bufferSize(cusolver_H, m, n, Lwork)
    !$acc end host_data

          if (istat /= CUSOLVER_STATUS_SUCCESS) &
                   write(*,*) 'cusolverDnDgesvd_bufferSize failed'
          print *, "working space (words) =", Lwork
          allocate(workspace_d(Lwork))

    !$acc enter data copyin(workspace_d) copyin(devInfo_d)

          jobu = 'A' ! all m columns of U
          jobvt= 'A' ! all m columns of VT

          !$acc host_data use_device(a, s, u, vt, devInfo_d, workspace_d, rwork)
            istat = cusolverDnDgesvd (cusolver_H, jobu, jobvt, &
                     m, n, a, lda, s, u, lda, vt, lda, workspace_d, Lwork, rwork, devInfo_d)
          !$acc end host_data
          !$acc wait

            if (istat /= CUSOLVER_STATUS_SUCCESS) &
                     write(*,*) 'cusolverDnDgesvd failed', istat

          ! copyout u, vt, s, devInfo_d to host
          !$acc update self(u, vt, s, devInfo_d)

          !print
          print *, "After gesvd: devInfo_d =", devInfo_d
          if (devInfo_d /= 0) write(*,*) 'SVD operation failed'
          print *, "S="
          call print_D_matrix(n, 1, s, 0)
          print *, "U="
          call print_D_matrix(m, m, u, lda)
          print *, "VT="
          call print_D_matrix(n, n, vt, lda)

          istat = cusolverDnDestroy(cusolver_H)
          if (istat /= CUSOLVER_STATUS_SUCCESS) &
                   write(*,*) 'handle destruction failed'

          ! Step 2: check accuracy of singular value

          diff_s = 0.d0
          diff_fro = 0.d0
          do i = 1, n
            err = abs( s(i) - s_exact(i) )
            diff_s = max( err, diff_s )
           end do
           print '(1x, a, d17.5)',"|S - S_exact| = ", diff_s

          ! Step 3: measure residual AU*S*VT

          ! w = s*vT
          !$acc host_data use_device(s, vt, w)
            cublas_status = cublasDdgmm(cublas_H, CUBLAS_SIDE_LEFT, n, n, &
                                        vt, lda, s, 1, w, lda)
          !$acc end host_data
          if (idbg == 1) then
            !$acc update self(w)
            print *, "W="
            call print_D_matrix(n, n, w, lda)
          end if

      ! Original A matrix is copied to device
          !$acc update device(a)

          ! a := -u*w + a
          !$acc host_data use_device(a, u, w)
            cublas_status = cublasDgemm_v2(cublas_H, CUBLAS_OP_N, CUBLAS_OP_N, &
                        m, n, n, minus_one, u, lda, w, lda, one, a, lda)
            cublas_status = cublasDnrm2_v2(cublas_H, lda*n, a, 1, diff_fro)
          !$acc end host_data

          if (idbg == 1) then
            !$acc update self(a)
            print *, "A = "
            call print_D_matrix(m, n, a, lda)
          end if

          print '(1x, a, d17.5)',"|A - U*S*VTt| = ", diff_fro

    !$acc exit data delete(workspace_d)
  !$acc end data


  end program main
```

It is necessary to compile and link with the -acc -Mcudalib=cusolver,cublas -Mcuda -ta=tesla option as a compile option.

Compile & run

```
$ make run
pgf90 -c -O2 -Minfo -ta=tesla,cuda9.0,cc35,cc60 -Mcudalib=cusolver,cublas -Kieee -acc -Mcuda svc
print_d_matrix:
     47, Loop not vectorized/parallelized: contains call
main:
     92, Loop unrolled 6 times (completely unrolled)
         Loop not vectorized: loop count too small
     95, Loop unrolled 2 times (completely unrolled)
    104, Generating create(rwork(:),s(:),u(:))
         Generating copy(a(:))
         Generating create(w(:),workspace_d(:),devinfo_d,vt(:))
    117, Generating enter data copyin(workspace_d(:),devinfo_d)
    132, Generating update self(devinfo_d,vt(:),u(:),s(:))
    152, Loop unrolled 2 times (completely unrolled)
    167, Generating update self(w(:))
    173, Generating update device(a(:))
    183, Generating update self(a(:))
    190, Generating exit data delete(workspace_d(:))
===== Program run =====
a.out
 working space (words) = 131
 After gesvd: devInfo_d = 0
 S =
    1 1 7.065283
    2 1 1.040081
 U =
    1 1 -0.308219
    1 2 0.488195
    1 3 0.816497
    2 1 -0.906133
```

```
   2 2 0.110706
   2 3 -0.408248
   3 1 -0.289695
   3 2 -0.865685
   3 3 0.408248
VT =
   1 1 -0.638636
   1 2 -0.769509
   2 1 -0.769509
   2 2 0.638636
|S - S_exact| = 0.22204D-15
W =
   1 1 -4.512143
   1 2 -5.436800
   2 1 -0.800352
   2 2 0.664233
A=
   1 1 0.000000
   1 2 0.000000
   2 1 0.000000
   2 2 0.000000
   3 1 0.000000
   3 2 0.000000
|A - U*S*VTt| = 0.23604D-14
```

Top of page

9. cuSolverDN: Singular value decomposition including singular vector calculation (by Jacobi) (equivalent to LAPACK Dgesvd)

In the previous section, we showed an example of solving singular value decomposition using the QR algorithm . Here, we will show an example of performing singular value decomposition using the Jacobian method. If you are targeting small and medium-sized matrices, which corresponds to **LAPACK's dgesvd routine , you may be able to enjoy high GPU performance due to the parallelism of the Jacobian method.** Furthermore, using the Jacobi method is convenient for obtaining an approximate solution up to a certain level of accuracy. The example below is the same process as F.2. SVD with singular vectors (via Jacobi method) written in Fortran OpenACC. Note that the dense matrix A must be input in column-major order .

The program below shows an example of a 3 x 2 dense matrix. SVD is expressed by the following formula. Here Σ is an m×n matrix that is zero except for its min(m,n) diagonal elements, U is an m×m unitary matrix, and V is an n×n unitary matrix. The diagonal elements of Σ are singular values of A. They are real numbers, not negative numbers, and are returned in descending order. The first min (m, n) columns of U and V are the left and right singular vectors of A.

$$A = U * \Sigma * V^H$$

$$A = \begin{pmatrix} 1.0 & 2.0 \\ 4.0 & 5.0 \\ 2.0 & 1.0 \end{pmatrix}$$

The Fortran interface module (cusolverDn.mod) for cuSOLVER provided by PGI 17.7 does not include an Interface to the CUDA C/C++ cuSOLVER library to use the singular value decomposition routine cusolverDnDgesvdj using the Jacobian method. I understand. That's why I made it myself. We defined an Interface for using the C library in the module cusolverDn_gesvdj module shown below. This allows cuSOLVER's cusolverDnDgesvdj routine to be used from Fortran.

In the program example below, a stream tag is created using cusolverDnSetStream(), so that the cusolverDN library task can be run redundantly in the CUDA stream. Since this program is a single stream, this act itself is meaningless, but by using CUDA streams, the program can be configured to perform multiple library tasks. This is just a demonstration of how to use CUDA streams. When using CUDA streams, it is important to note that it is necessary to use the cudaDeviceSynchronize() function to synchronize at points where subsequent processing must be synchronized. Note that this function requires direct use of the CUDA (Fortran) API and cannot be synchronized using !$acc wait at the OpenACC level.

OpenACC + cuSOLVER-cusolverDnDgesvdj (jacobi_svd.f90)

```
! Copyright (C) HPC WORLD (Prometech Software, Inc. and GDEP Solutions, Inc.) All rights reserve
! Copyright (c) 2017, NVIDIA CORPORATION. All rights reserved.
!
!
! Permission is hereby granted, free of charge, to any person obtaining a
! copy of this software and associated documentation files (the "Software"),
! to deal in the Software without restriction, including without limitation
! the rights to use, copy, modify, merge, publish, distribute, sublicense,
! and/or sell copies of the Software, and to permit persons to whom the
! Software is furnished to do so, subject to the following conditions:
!
! The above copyright notice and this permission notice shall be included in
! all copies or substantial portions of the Software.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
! IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
! FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
! THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
! LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
! FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
! DEALINGS IN THE SOFTWARE.
!
module cusolverDn_gesvdj
  use cudafor
  use cusolverDN

  type gesvdjInfo
    type(c_ptr) :: svInfo
  end type gesvdjInfo

! cusolverDnCreateGesvdjInfo(info)
  interface
     integer(c_int) function cusolverDnCreateGesvdjInfo(info) bind(C,name='cusolverDnCreateGesvc
       import gesvdjInfo
       type(gesvdjInfo) :: info
     end function cusolverDnCreateGesvdjInfo
  end interface

! cusolverDnXgesvdjSetTolerance(info, tolerance)
  interface
     integer(c_int) function cusolverDnXgesvdjSetTolerance(info, tolerance) &
     bind(C,name='cusolverDnXgesvdjSetTolerance')
       use iso_c_binding
       import gesvdjInfo
       type(gesvdjInfo) :: info
       real(c_double), value :: tolerance
     end function cusolverDnXgesvdjSetTolerance
  end interface
```

```
! cusolverDnXgesvdjSetMaxSweeps(info, max_sweeps)
  interface
     integer(c_int) function cusolverDnXgesvdjSetMaxSweeps(info, max_sweeps) &
     bind(C,name='cusolverDnXgesvdjSetMaxSweeps')
       use iso_c_binding
       import gesvdjInfo
       type(gesvdjInfo) :: info
       integer(c_int),value :: max_sweeps
     end function cusolverDnXgesvdjSetMaxSweeps
  end interface

! cusolverDnDgesvdj_bufferSize
  interface
     integer(c_int) function cusolverDnDgesvdj_bufferSize(cusolver_Hndl, jobz, econ, &
     m, n, a, lda, s, u, ldu, v, ldv, lwork, info) bind(C,name='cusolverDnDgesvdj_bufferSize')
       use iso_c_binding
       import cusolverDnHandle
       import gesvdjInfo
       type(cusolverDnHandle), value :: cusolver_Hndl
       integer(c_int), value :: jobz
       integer(c_int),value :: econ, m, n, lda, ldu, ldv
       real(c_double), device :: a(*), s(*), u(*), v(*)
       type(gesvdjInfo) :: info
     end function cusolverDnDgesvdj_bufferSize
  end interface

! cusolverDnDgesvdj
  interface
     integer(c_int) function cusolverDnDgesvdj(cusolver_Hndl, jobz, econ, &
     m, n, a, lda, s, u, ldu, v, ldv, work, lwork, devinfo, info ) bind(C,name='cusolverDnDgesvd
       use iso_c_binding
       import cusolverDnHandle
       import gesvdjInfo
       type(cusolverDnHandle), value :: cusolver_Hndl
       integer(c_int), value :: jobz
       integer(c_int),value :: econ, m, n, lda, ldu, ldv
       real(c_double), device :: a(*), s(*), u(*), v(*), work(*)
       integer(c_int) :: devinfo
       type(gesvdjInfo) :: info
     end function cusolverDnDgesvdj
  end interface

! cusolverDnXgesvdjGetSweeps
  interface
     integer(c_int) function cusolverDnXgesvdjGetSweeps(cusolver_Hndl, info, executed_sweeps) &
     bind(C,name='cusolverDnXgesvdjGetSweeps')
       use iso_c_binding
       import cusolverDnHandle
       import gesvdjInfo
       type(cusolverDnHandle), value :: cusolver_Hndl
       type(gesvdjInfo) :: info
       integer(c_int) :: executed_sweeps
     end function cusolverDnXgesvdjGetSweeps
  end interface

! cusolverDnXgesvdjGetResidual
  interface
     integer(c_int) function cusolverDnXgesvdjGetResidual(cusolver_Hndl, info, residual) &
     bind(C,name='cusolverDnXgesvdjGetResidual')
       use iso_c_binding
       import cusolverDnHandle
       import gesvdjInfo
       type(cusolverDnHandle), value :: cusolver_Hndl
       type(gesvdjInfo) :: info
       real(c_double) :: residual
     end function cusolverDnXgesvdjGetResidual
  end interface
end module cusolverDn_gesvdj

module aux_routine
contains

subroutine print_D_matrix(m, n, a, lda)
integer :: m, n, i, j
real(8) :: a(*)

do i=1, m
  do j=1, n
    print "(5(,i5,1x,i5,2x,D20.14,2x))", i, j, a(i+(j-1)*lda)
  end do
end do
end subroutine print_D_matrix
end module aux_routine

program main
  use cudafor ! cudaStreamCreateWithFlags in the cudafor.mod
  use cusolverDn
  use cusolverDn_gesvdj
  use aux_routine
  implicit none

  integer, parameter :: m=3, n=2
  integer, parameter::lda = m
  integer, parameter :: idbg = 1
  real(8), parameter :: one = 1.d0, minus_one = -1.d0

  real(8) :: a (lda*n) ! column-major format with dimensions mxn
  real(8) :: u (lda*m) ! m by m unitary matrix
  real(8) :: vt(lda*n) ! n by n unitary matrix
  real(8), allocatable, dimension(:) :: s ! singular value
  real(8), allocatable :: workspace_d(:)
  real(8) :: S_exact(n)
  real(8) :: err, diff_s
  integer :: devInfo_d
  integer :: istat, Lwork
  integer :: i

! configuration of GESVDJ
  real(8) :: tol
  integer :: max_sweeps
  integer, parameter :: econ=0 ! econ = 1 for economy size

! numerical results of GESVDJ
  real(8) :: residual
  integer :: executed_sweeps

!cuda library variables
  type(cusolverDnHandle) :: cusolver_H
  type(gesvdjInfo) :: gesvdj_params
  integer :: jobz
  integer(kind=cuda_stream_kind) :: stream
  integer :: cusolver_status
  integer :: cublas_status

! | 1 2 |
! a = | 4 5 |
! | 2 1 |
!

! cusolverDnDgesvdj() argument list
! jobz CUSOLVER_EIG_MODE_NOVECTOR: compute singular values only
! CUSOLVER_EIG_MODE_VECTOR: compute singular value and singular vectors
!econ econ = 1 for economy size
! m nubmer of rows of A, 0 <= m
! n number of columns of A, 0 <= n
! a m-by-n
!lda leading dimension of A
! s min(m,n)
! the singular values in descending order
! u m-by-m if econ = 0
! m-by-min(m,n) if econ = 1
! ldu leading dimension of U, ldu >= max(1,m)
!vt n-by-n if econ = 0
!n-by-min(m,n) if econ = 1
!ldv leading dimension of V, ldv >= max(1,n)

!Initialize
  allocate(s(1:min(m,n)))
  Lwork = 0
  tol = 1.d-7
  max_sweeps = 15
  residual = 0.d0
  executed_sweeps = 0
  jobz = CUSOLVER_EIG_MODE_VECTOR
  print *,"Eigmode=", jobz

  print *, "tol =", tol
  print *, "max sweeps =", max_sweeps
  print *, "econ = ", econ

! set a data
  a(1:lda*n) = (/1.0, 4.0, 2.0, 2.0, 5.0, 1.0/)
  print *, "A="
  call print_D_matrix(m, n, a, lda)

! Exact SV numbers
   S_exact(:) =(/7.065283497082729d0, 1.040081297712078d0/)

!$acc data copy(a) create(s, u, vt, devInfo_d)

        ! Step 1: create handle and stream

    cusolver_status = cusolverDnCreate(cusolver_H)
    if (cusolver_status /= CUSOLVER_STATUS_SUCCESS) &
        print *, 'handle creation failed'

    ! set a CUDA Lib stream -- when performing several small independent computations (run in as
    ! you explicitly need to set cudaDeviceSynchronize() at an appropriate place by CUDA-API

    istat = cudaStreamCreateWithFlags(stream, cudaStreamNonBlocking)
    istat = cusolverDnSetStream(cusolver_H, stream)

    ! step 2 : configure gesvdj, Tolerance, Max sweeps

    istat = cusolverDnCreateGesvdjInfo(gesvdj_params)
    istat = cusolverDnXgesvdjSetTolerance(gesvdj_params, tol)
    istat = cusolverDnXgesvdjSetMaxSweeps(gesvdj_params, max_sweeps)

        ! Step 3: query workspace of SVD, allocate workspace_d(Lwork)

  !$acc host_data use_device(a, s, u, vt)
        istat = cusolverDnDgesvdj_bufferSize(cusolver_H, jobz, econ, &
            m, n, a, lda, s, u, lda, vt, lda, Lwork, gesvdj_params)
  !$acc end host_data

        if (istat /= CUSOLVER_STATUS_SUCCESS) &
                  write(*,*) 'cusolverDnDgesvdj_bufferSize failed'
        print *, "working space (words) =", Lwork
        allocate(workspace_d(Lwork))

  !$acc enter data copyin(workspace_d) copyin(devInfo_d)
```

```
     ! step 4 : compute SVD

         !$acc host_data use_device(a, s, u, vt, devInfo_d, workspace_d)
           istat = cusolverDnDgesvdj (cusolver_H, jobz, econ, &
                    m, n, a, lda, s, u, lda, vt, lda, workspace_d, Lwork, devInfo_d, gesvdj_parar
         !$acc end host_data

     ! Synchronize for cusolverDnDgesvdj completion (required)
      istat =cudaDeviceSynchronize()                   ! CUDA-API, it can't use a OpenACC wait di

       if (istat /= CUSOLVER_STATUS_SUCCESS) &
          write(*,*) 'cusolverDnDgesvd failed', istat
     ! copyout u, vt, s, devInfo_d to host
       !$acc update self(s, u, vt, devInfo_d)

         ! Step 5: measure residual |A - U*S*V**H|_F

     istat = cusolverDnXgesvdjGetSweeps(cusolver_H, gesvdj_params, executed_sweeps)
     istat = cusolverDnXgesvdjGetResidual(cusolver_H, gesvdj_params, residual)

   !$acc exit data delete(workspace_d)
 !$acc end data

         ! print out

         print *, "After gesvd: devInfo_d =", devInfo_d
         if (devInfo_d == 0) then
           write(*,*) 'SVD operation was converged'
         else if (devInfo_d < 0) then
           write(*,*) -devInfo_d, "d-th parameter is wrong"
           stop
         else
           write(*,*) "WARNING: gesvdj does not converge ", devInfo_d
         end if

         print *, "Singular values ="
         call print_D_matrix(n, 1, s, 0)
         print *, "U = left singular vectors"
         call print_D_matrix(m, m, u, lda)
         print *, "VT = right singular vectors"
         call print_D_matrix(n, n, vt, lda)

         ! check accuracy of singular value

         diff_s = 0.d0
         do i = 1, n
           err = abs( s(i) - s_exact(i) )
           diff_s = max( err, diff_s )
          end do
          print '(1x, a, d17.5)',"|S - S_exact| = ", diff_s

         ! measure residual AU*S*VT**H

         print '(1x, a, d17.5)',"|A - U*S*V**H| = ", residual
         print '(1x, a, I5 )',"number of executed sweeps = ", executed_sweeps


         istat = cusolverDnDestroy(cusolver_H)
         if (istat /= CUSOLVER_STATUS_SUCCESS) &
                  write(*,*) 'handle destruction failed'

 end program main
```

It is necessary to compile and link with the -acc -Mcudalib=cusolver -Mcuda -ta=tesla options as compile options .

Compile & run

```
$ make run
pgf90 -c -O2 -Minfo -ta=tesla,cuda9.0,cc35,cc60 -Mcudalib=cusolver,cublas -Kieee -acc -Mcuda jac
print_d_matrix:
    126, Loop not vectorized/parallelized: contains call
main:
    210, Loop unrolled 6 times (completely unrolled)
         Loop not vectorized: loop count too small
    215, Loop unrolled 2 times (completely unrolled)
    217, Generating create(devinfo_d,s(:),vt(:),u(:))
         Generating copy(a(:))
    249, Generating enter data copyin(workspace_d(:),devinfo_d)
    264, Generating update self(vt(:),u(:),s(:),devinfo_d)
    271, Generating exit data delete(workspace_d(:))
    296, Loop unrolled 2 times (completely unrolled)
pgf90 -o a.out jacobi_svd.o -ta=tesla,cuda9.0,cc35,cc60 -Mcudalib=cusolver -acc -Mcuda
===== Program run =====
a.out
 Eigmode=1
 tol = 9.9999999999999995E-008
 max sweeps = 15
 econ = 0
 A=
    1 1 0.10000000000000D+01
    1 2 0.20000000000000D+01
    2 1 0.40000000000000D+01
    2 2 0.50000000000000D+01
    3 1 0.20000000000000D+01
    3 2 0.10000000000000D+01
 working space (words) = 3168
 After gesvd: devInfo_d = 0
 SVD operation was converged
 Singular values =
    1 1 0.70652834970827D+01
    2 1 0.10400812977121D+01
 U = left singular vectors
    1 1 0.30821892063279D+00
    1 2 -.48819507401990D+00
    1 3 0.81649658092773D+00
    2 1 0.90613333377729D+00
    2 2 -.11070553170904D+00
    2 3 -.40824829046386D+00
    3 1 0.28969549251172D+00
    3 2 0.86568461633075D+00
    3 3 0.40824829046386D+00
 VT = right singular vectors
    1 1 0.63863583713640D+00
    1 2 0.76950910814953D+00
    2 1 0.76950910814953D+00
    2 2 -.63863583713640D+00
 |S - S_exact| = 0.88818D-15
 |A - U*S*V**H| = 0.36550D-14
 number of executed sweeps = 1
```

Top of page

10. cuSolverSP: Parallel direct sparse solver (CSR QR decomposition)

PGI 17.7 provided a Fortran MODULE interface for the cuSolverDN type routines, but not yet for the cuSolverSP and cuSolverRF type libraries. I believe that it will be provided in the future, but here I will create a Fortran MODULE interface for cuSolverSP and present an example program that performs sparse QR decomposition. The example used here is a version of Batched Sparse QR example 1 shown in the NVIDIA cuSOLVER reference manual, which was rewritten by the author in his Fortran language. Note that the library here uses a "batch processing routine" that solves multiple linear systems. A technical explanation of the batch solver used here can also be found on the NVIDIA Parallel FORALL: Parallel Direct Solvers with cuSOLVER: Batched QR blog .

$$A_i x_i = b_i$$

```
       | 1  | Prepare multiple Marices A (small perturbations).
  A =  | 2  | for batch processing
       | 3  |
       | 0.1 0.1 0.1 4 |
```

I will post how to use this program from CUDA Fortran and OpenACC Fortran.

The programming response differs depending on which programming model is used to manage device memory for arrays and variables, but here we will explain how to declare a device array on the CUDA Fortran side, and how to declare an array on the OpenACC side. An example of this is shown below.

Batched Sparse QR example with CUDA Fortran

As of PGI 17.7, a Fortran Module interface to the cuSOLVER SP (sparse) library developed in the CUDA C language is not provided, so I created it myself (cusolver_mod.cuf) as shown below. This is an interface to mediate the binding between C and Fortran. In the future, if the PGI compilation system provides a Fortran MODULE for cuSOLVER SP (possibly available with use cusplverSP), the following cusolver_mod.cuf routine will no longer be needed. First, we will present an example of coding using CUDA Fortran. After this, I will show an example of modifying this program and coding it using OpenACC . Please refer to the specifications of the batch library routine (cusolverSpXcsrqrBatched()) used here. In the explanation of the argument specifications for this function, there is a distinction between host and device, which indicates which side the variable is placed on. By looking at this, you can tell whether or not you need to handle it as a device variable.

Fortran Module Interface for cuSOLVER SP QR (cusolver_mod.cuf)

```
 ! Copyright (C) HPC WORLD (Prometech Software, Inc. and GDEP Solutions, Inc.) All rights reserve
 !
 ! Permission is hereby granted, free of charge, to any person obtaining a
 ! copy of this software and associated documentation files (the "Software"),
 ! to deal in the Software without restriction, including without limitation
 ! the rights to use, copy, modify, merge, publish, distribute, sublicense,
 ! and/or sell copies of the Software, and to permit persons to whom the
 ! Software is furnished to do so, subject to the following conditions:
 !
 ! The above copyright notice and this permission notice shall be included in
 ! all copies or substantial portions of the Software.
 !
 ! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 ! IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 ! FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
 ! THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 ! LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 ! FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
 ! DEALINGS IN THE SOFTWARE.
 !
module cuSOLVER_SP_QR
```

```
  use cudafor
  use cusparse

  enum, bind(C) ! cusolverStatus_t
    enumerator :: CUSOLVER_STATUS_SUCCESS=0
    enumerator :: CUSOLVER_STATUS_NOT_INITIALIZED=1
    enumerator :: CUSOLVER_STATUS_ALLOC_FAILED=2
    enumerator :: CUSOLVER_STATUS_INVALID_VALUE=3
    enumerator :: CUSOLVER_STATUS_ARCH_MISMATCH=4
    enumerator :: CUSOLVER_STATUS_MAPPING_ERROR=5
    enumerator :: CUSOLVER_STATUS_EXECUTION_FAILED=6
    enumerator :: CUSOLVER_STATUS_INTERNAL_ERROR=7
    enumerator :: CUSOLVER_STATUS_MATRIX_TYPE_NOT_SUPPORTED=8
    enumerator::CUSOLVER_STATUS_NOT_SUPPORTED = 9
    enumerator :: CUSOLVER_STATUS_ZERO_PIVOT=10
    enumerator :: CUSOLVER_STATUS_INVALID_LICENSE=11
  end enum

  enum, bind(C) ! cusolverEigType_t
    enumerator :: CUSOLVER_EIG_TYPE_1=1
    enumerator :: CUSOLVER_EIG_TYPE_2=2
    enumerator :: CUSOLVER_EIG_TYPE_3=3
  end enum

  type cusolverSpContext
        type(c_ptr) :: cusolverSpHandle
  end type cusolverSpContext
  type csrqrInfo
    type(c_ptr) :: Info
  end type csrqrInfo
  type cusolverSpHandle
    type(c_ptr) :: SpHandle
  end type cusolverSpHandle

! cusolverSpCreate
  interface
     integer(c_int) function cusolverSpCreate(handle) bind(C,name='cusolverSpCreate')
       import cusolverSpHandle
       type(cusolverSpHandle) :: handle
     end function cusolverSpCreate
  end interface
! cusolverSpDestroy
  interface
     integer(c_int) function cusolverSpDestroy(handle) bind(C,name='cusolverSpDestroy')
       import cusolverSpHandle
       type(cusolverSpHandle), value :: handle
     end function cusolverSpDestroy
  end interface

! cusolverSpCreateCsrqrInfo
  interface
     integer(c_int) function cusolverSpCreateCsrqrInfo(info) bind(C,name='cusolverSpCreateCsrqrI
       import csrqrInfo
       type(csrqrInfo) :: info
     end function cusolverSpCreateCsrqrInfo
  end interface
! cusolverSpDestroyCsrqrInfo
  interface
     integer(c_int) function cusolverSpDestroyCsrqrInfo(info) bind(C,name='cusolverSpDestroyCsrq
       import csrqrInfo
       type(csrqrInfo) :: info
     end function cusolverSpDestroyCsrqrInfo
  end interface

!cusolverSpDcsrqrBufferInfoBatched
interface cusolverSpDcsrqrBufferInfoBatched
  integer(c_int) function cusolverSpDcsrqrBufferInfoBatched( cusolver_Hndl, m, n, nnzA, &
     descrA, csrValA, csrRowPtrA, csrColIndA, BatchSize, info, internalDataInBytes, workspaceInB
```

```
      bind(C,name="cusolverSpDcsrqrBufferInfoBatched")
   use iso_c_binding
   import cusolverSpHandle, cusparseMatDescr, csrqrInfo
   type(cusolverSpHandle), value :: cusolver_Hndl
   type(cusparseMatDescr), value :: descrA
   type(csrqrInfo), value :: info
   integer(c_int), value :: m, n, nnzA, batchSize
   real(c_double), device :: csrValA(*)
 !! !pgi$ ignore_tkr (d) csrRowPtrA, (d) csrColIndA
   integer(c_int), device :: csrRowPtrA(*), csrColIndA(*)
 !! !pgi$ ignore_tkr (k) internalDataInBytes, (k) workspaceInBytes
   integer(8) :: internalDataInBytes, workspaceInBytes
   end function cusolverSpDcsrqrBufferInfoBatched
 end interface cusolverSpDcsrqrBufferInfoBatched


 ! cusolverSpXcsrqrAnalysis
 !interface cusolverSpXcsrqrAnalysis
 ! integer(c_int) function cusolverSpXcsrqrAnalysis( cusolver_Hndl, m, n, nnzA, &
 ! descrA, csrRowPtrA, csrColIndA, info ) &
 ! bind(C,name="cusolverSpXcsrqrAnalysis")
 ! use iso_c_binding
 ! import cusolverSpHandle, cusparseMatDescr, csrqrInfo
 ! type(cusolverSpHandle), value :: cusolver_Hndl
 ! type(cusparseMatDescr), value :: descrA
 ! type(csrqrInfo), value :: info
 ! integer(c_int), value :: m, n, nnzA
 !!pgi$ ignore_tkr (d) csrRowPtrA, (d) csrColIndA
 ! integer(c_int), device :: csrRowPtrA(*), csrColIndA(*)
 ! end function cusolverSpXcsrqrAnalysis
 !end interface cusolverSpXcsrqrAnalysis

 ! cusolverSpXcsrqrAnalysisBatched
 interface cusolverSpXcsrqrAnalysisBatched
   integer(c_int) function cusolverSpXcsrqrAnalysisBatched( cusolver_Hndl, m, n, nnzA, &
                  descrA, csrRowPtrA, csrColIndA, info ) &
                  bind(C,name="cusolverSpXcsrqrAnalysisBatched")
   use iso_c_binding
   import cusolverSpHandle, cusparseMatDescr, csrqrInfo
   type(cusolverSpHandle), value :: cusolver_Hndl
   type(cusparseMatDescr), value :: descrA
   type(csrqrInfo), value :: info
   integer(c_int), value :: m, n, nnzA
 !! !pgi$ ignore_tkr (d) csrRowPtrA, (d) csrColIndA
   integer(c_int), device :: csrRowPtrA(*), csrColIndA(*)
   end function cusolverSpXcsrqrAnalysisBatched
 end interface cusolverSpXcsrqrAnalysisBatched

 ! cusolverSpDcsrqrsvBatched
 interface cusolverSpDcsrqrsvBatched
   integer(c_int) function cusolverSpDcsrqrsvBatched( cusolver_Hndl, m, n, nnzA, &
                  descrA, csrValA, csrRowPtrA, csrColIndA, b, x, BatchSize, info, pBuffer ) &
                  bind(C,name="cusolverSpDcsrqrsvBatched")
   use iso_c_binding
   import cusolverSpHandle, cusparseMatDescr, csrqrInfo
   type(cusolverSpHandle), value :: cusolver_Hndl
   type(cusparseMatDescr), value :: descrA
   type(csrqrInfo), value :: info
   integer(c_int), value :: m, n, nnzA, Batchsize
   real(c_double), device :: csrValA(*)
 !! !pgi$ ignore_tkr (d) csrRowPtrA, (d) csrColIndA
   integer(c_int), device :: csrRowPtrA(*), csrColIndA(*)
   real(c_double), device :: b(*), x(*)
 !! !pgi$ ignore_tkr pBuffer
   character(c_char), device :: pBuffer(*)

   end function cusolverSpDcsrqrsvBatched
 end interface cusolverSpDcsrqrsvBatched
```

```
end module cuSOLVER_SP_QR
```

Next, the driver routine (main.cuf) for Batched Sparse QR decomposition is illustrated. The matrix CSR format is input in row-major order.

Batched Sparse QR main program (main.cuf)

```
! Copyright (C) HPC WORLD (Prometech Software, Inc. and GDEP Solutions, Inc.) All rights reserve
!
! Permission is hereby granted, free of charge, to any person obtaining a
! copy of this software and associated documentation files (the "Software"),
! to deal in the Software without restriction, including without limitation
! the rights to use, copy, modify, merge, publish, distribute, sublicense,
! and/or sell copies of the Software, and to permit persons to whom the
! Software is furnished to do so, subject to the following conditions:
!
! The above copyright notice and this permission notice shall be included in
! all copies or substantial portions of the Software.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
! IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
! FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
! THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
! LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
! FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
! DEALINGS IN THE SOFTWARE.
!
program main
  use cudafor
  use cusparse
  use cusolver_SP_QR
  implicit none

  integer, parameter :: idbg = 2 ! debug write enable
  integer, parameter :: m = 4, nnzA = 7
  integer, parameter :: batchsize = 4

  integer :: cusolver_status
  integer :: status1, status2, status3, status4, status5
  type(cusolverSpHandle) :: cusolver_Hndl
  type(cusparseMatDescr) :: descrA
  type(csrqrInfo) :: info

  integer :: csrRowPtrA(m+1), csrColIndA(nnzA)
  integer, device :: d_csrRowPtrA(m+1), d_csrColIndA(nnzA)

  double precision :: csrValA(nnzA), b(m)
  double precision, device:: d_csrValA(nnzA*batchsize), d_b(m*batchsize), d_x(m*batchsize)

  double precision :: csrValABatch(nnzA*batchsize), bBatch(m*batchsize), xbatch(m*batchsize)

  integer(8) :: size_internal, size_qr
  character(c_char), device, allocatable :: pBuffer(:)

  !locals
  integer :: i, j, colidx, batchId
  integer::ierr_code
  double precision :: eps, Areg, breg, xreg

  !Result
  integer :: row, baseA, start, end, col
  double precision :: csrValAj
  double precision :: sup_res, r, Ax
```

```
    !Random Number
    double precision :: rnd

    ! | 1 |
    ! A = | 2 |
    ! | 3 |
    ! | 0.1 0.1 0.1 4 |
    !
    ! CSR of A is based 1 indexing <==== caution (CUSPARSE_INDEX_BASE_ONE)
    !
    ! b = [1 1 1 1]

  ! step 1: data setting
      csrRowPtrA(1:m+1) = (/1, 2, 3, 4, 8/)
      csrColIndA(1:nnzA) = (/1, 2, 3, 1, 2, 3, 4/)
      csrValA(1:nnzA) = (/1.0d0, 2.0d0, 3.0d0, 0.1d0, 0.1d0, 0.1d0, 4.0d0/)
      b(1:m) = (/1.0d0, 1.0d0, 1.0d0, 1.0d0/)

      print *, "sizeof(csrRowPtrA, d_csrRowPtrA, csrColIndA, d_csrColIndA)", &
                sizeof(csrRowPtrA(1)), sizeof(d_csrRowPtrA(1)), sizeof(csrColIndA(1)), sizeof(d_cs

  ! prepare Aj and bj on host
     do colidx = 1, nnzA
          Areg = csrValA(colidx)
          do batchId = 1, batchSize
             call random_number(rnd)
             eps = dble( mod(rnd,100.d0) + 1.0d0 ) * 1.0D-4
             csrValABatch((batchId-1)*nnzA + colidx) = Areg + eps
          enddo
     enddo

     do j = 1, m
          breg = b(j)
          do batchId = 1, batchSize
             call random_number(rnd)
             eps = dble( mod(rnd,100.d0) + 1.0d0 ) * 1.0D-4
             bBatch((batchId-1)*m + j) = breg + eps;
          enddo
     enddo

      if (idbg == 2) print *,"csrValABatch", csrValABatch
      if (idbg == 2) print *,"bBatch", bBatch

  ! step 2: create cusolver handle, qr info and matrix descriptor

      status1 = cusolverSpCreate(cusolver_Hndl)
            if (idbg == 1) print *, "status1 = ", status1
      status2 = cusparseCreateMatDescr(descrA)
            if (idbg == 1) print *, "status2 = ", status2
      status3 = cusparseSetMatType(descrA, CUSPARSE_MATRIX_TYPE_GENERAL)
            if (idbg == 1) print *, "status3 = ", status3
      status4 = cusparseSetMatIndexBase(descrA, CUSPARSE_INDEX_BASE_ONE) ! base=1
            if (idbg == 1) print *, "status4 = ", status4
      status5 = cusolverSpCreateCsrqrInfo(info)
            if (idbg == 1) print *, "status5 = ", status5

  ! step 3: copy Aj and bj to device
      d_csrValA = csrValABatch
      d_csrColIndA= csrColIndA
      d_csrRowPtrA= csrRowPtrA
      d_b = bBatch

  ! step 4: symbolic analysis

      cusolver_status = cusolverSpXcsrqrAnalysisBatched( &
                        cusolver_Hndl, m, m, nnzA, descrA, d_csrRowPtrA, d_csrColIndA, info )
      if (idbg == 1) then
        if (cusolver_status /= CUSOLVER_STATUS_SUCCESS) print *, "step4 cusolver_status = ", cusol
        print *, 'error code = ' ,cusolver_status, cudaGetErrorString( cudaGetLastError() )
      end if

  ! step 5: prepare working space

      cusolver_status = cusolverSpDcsrqrBufferInfoBatched( &
                        cusolver_Hndl, m, m, nnzA, descrA, d_csrValA, d_csrRowPtrA, d_csrColIndA,
                        batchsize, info, size_internal, size_qr)

      if (idbg == 1) then
        if (cusolver_status /= CUSOLVER_STATUS_SUCCESS) print *, "step5 cusolver_status = ", cuso
      end if

      print *, "numerical factorization needs internal data (bytes) =", size_internal
      print *, "numerical factorization needs working space (bytes) =", size_qr

      allocate(pBuffer(size_qr), stat=ierr_code) ! Bytes
        if (idbg == 1) print *, "step5: alloc ierr_code = ", ierr_code

  ! step 6: numerical factorization

      cusolver_status = cusolverSpDcsrqrsvBatched( &
                        cusolver_Hndl, m, m, nnzA, &
                        descrA, d_csrValA, d_csrRowPtrA, d_csrColIndA, &
                        d_b, d_x, &
                        batchSize, &
                        info, &
                        pBuffer)

      if (idbg == 1) then
        if (cusolver_status /= CUSOLVER_STATUS_SUCCESS) print *, "step6 cusolver_status = ", cuso
        print *, 'error code = ' ,cusolver_status, cudaGetErrorString( cudaGetLastError() )
      end if

  ! step 7: check residual
  !xBatch = [x0, x1, x2, ...]

  !copy into CPU mem
      xbatch = d_x

      if (idbg == 2) print *, "xbatch=",xbatch

      print *, "******* Fortran Index Base is adjusted : BaseA = 0 ********************* "
      baseA = 0

      print *, " ============ Residual CHECK ==============="
      do batchId = 1 , batchsize
       ! measure |bj - Aj*Xj|
        sup_res = 0.d0
        do row = 1, m
          start = csrRowPtrA(row ) - baseA
          end = csrRowPtrA(row+1) - baseA
          Ax = 0.d0

          do colidx = start, end-1
            col = csrColIndA(colidx) - baseA
            Areg = csrValAbatch((batchId-1)*nnzA + colidx) ! Areg = csrValAj(colidx)
            Xreg = xBatch((batchId-1)*m + col) ! Xreg = xj(col)
            Ax = Ax + Areg * Xreg
          end do

          r = bBatch((batchID-1)*m + row) - Ax ! r = bj(row) -Ax
          ! sup_res = (sup_res > fabs(r))? sup_res : fabs(r);
          sup_res = max( abs(r), sup_res )

  ! if (sup_res > abs(r)) then
  ! sup_res = sup_res
  !else
```

```
! sup_res = abs(r)
! end if
      end do
      print *, "batchId =", batchId," sup|bj - Aj*xj| =", sup_res
    end do


    print *, " ============ Result X ==============="
    do batchId = 1 , batchsize
       do row = 1, m
          print *, "x(", batchId, ")= [", row,"]", xBatch((batchId-1)*m + row)
       end do
       print *, ""
    end do
end
```

Simply copying and pasting the Makefile below will result in a make execution error. The [tab] delimiter in the makefile description has been converted to a space, so it is necessary to explicitly change the space to a [tab]. (Example below)

```
main : main.f90
 [ tab ]   pgf90 main.f90
```

Makefile (compilation method)

```
CC = nvcc
CFLAGS =
FC = pgf90
FFLAGS = -O3 -Minfo -Mcuda=cuda8.0,cc35,cc60 -Mcudalib=cusparse
LDFLAGS = -Mcuda=cuda8.0,cc35,cc60

TARGET = SPsolve
INCLUDE =
LIBS = -lcusolver # -lgomp
# On Ubuntu, the following library path and -lgomp may be required
# NVIDIA cuSOLVER library requires part of gcc's openmp library
#LDIR = -L/usr/lib/gcc/x86_64-linux-gnu/5


modfile = cusolver_mod.cuf
SRC1 = main.cuf

# Define extensions to which suffix rules apply
.SUFFIXES: .cuf .cu .o

MODOBJ = $(modfile:.cuf=.o)
OBJ1 = $(SRC1:.cuf=.o)
MOD1 = cusolver_sp_qr.mod

RUN : $(TARGET)
                  @echo "===== Program run ====="
                  $(TARGET)
$(TARGET): $(MODOBJ) $(OBJ1)
                  $(FC) -o $@ $^ $(LDFLAGS) $(LDIR) $(LIBS)


$(MOD1) : $(modfile)
                  $(FC) $(FFLAGS) -c $?
$(OBJ1): $(MOD1) $(SRC1)

.cuf.o:
                  $(FC) -c $(FFLAGS) $<
.cu.o:
                  $(CC) -c $(CFLAGS) $<
```

```
.PHONY: clean
clean:
                @echo 'Cleaning up...'
            rm -f *.o *.mod $(TARGET) *~
```

Compilation & execution results

```
[kato@photon32 QRtest1]$ make
pgf90 -c -O3 -Minfo -Mcuda=cuda8.0,cc35,cc60 -Mcudalib=cusparse cusolver_mod.cuf
pgf90 -c -O3 -Minfo -Mcuda=cuda8.0,cc35,cc60 -Mcudalib=cusparse main.cuf
main:
     71, Loop unrolled 5 times (completely unrolled)
     72, Array assignment / Forall at line 73 fused
         Loop unrolled 7 times (completely unrolled)
         Loop not vectorized: loop count too small
     74, Loop unrolled 4 times (completely unrolled)
     82, Loop not vectorized/parallelized: contains call
         FMA (fused multiply-add) instruction(s) generated
     91, Loop not vectorized/parallelized: contains call
         FMA (fused multiply-add) instruction(s) generated
    172, Loop not vectorized/parallelized: contains call
    175, Outer loop unrolled 4 times (completely unrolled)
    180, Unrolled inner loop 4 times
         Generated 2 prefetch instructions for the loop
         Unrolled inner loop 4 times
         Generated 2 prefetch instructions for the loop
         Unrolled inner loop 4 times
         Generated 2 prefetch instructions for the loop
         Unrolled inner loop 4 times
         Generated 2 prefetch instructions for the loop
         FMA (fused multiply-add) instruction(s) generated
    203, Loop not vectorized/parallelized: contains call
pgf90 -o SPsolve cusolver_mod.o main.o -Mcuda=cuda8.0,cc35,cc60 -lcusolver
===== Program run =====
SPsolve
 sizeof(csrRowPtrA, d_csrRowPtrA, csrColIndA, d_csrColIndA) 4
            4 4 4
 csrValABatch 1.000190792295671 2.000179731460909
    3.000132064769846 0.1001762263696543 0.1001628016430586
   0.1001500500225553 4.000169009995976 1.000119069206783
    2.000163682999292 3.000144817609972 0.1001437479734487
   0.1001670186653253 0.1001425331039673 4.000182114792401
    1.000106716529557 2.000158878275024 3.000165795016470
   0.1001463869055930 0.1001628171802322 0.1001307016646178
    4.000187350713939 1.000180008450829 2.000115906560827
    3.000160754590159 0.1001700415717258 0.1001531034351497
   0.1001216954552562 4.000196496681088
 bBatch 1.000182450045416 1.000166538252814
    1.000186841054682 1.000110375401338 1.000145236365631
    1.000145251688474 1.000116588275369 1.000155851050659
    1.000125862765850 1.000112255030956 1.000187784792441
    1.000198703067770 1.000133737619532 1.000188679946082
    1.000142956695574 1.000175135506380
 numerical factorization needs internal data (bytes) = 864
 numerical factorization needs working space (bytes) = 246528
 xbatch= 0.9999916593410789 0.5000383328163729
   0.3333809377259567 0.2041067826368187 1.000026164043516
   0.5000317024998334 0.3333561041470322 0.2041256247608860
    1.000019144193291 0.5000164066433919 0.3333775041545496
   0.2041369446005358 0.9999537374963444 0.5000653595450339
   0.3333631223478223 0.2041274097506157
 ******* Fortran Index Base is adjusted : BaseA = 0 *********************
  ============ Residual CHECK ===============
 batchId = 1 sup|bj - Aj*xj| = 8.8817841970012523E-016
 batchId = 2 sup|bj - Aj*xj| = 6.6613381477509392E-016
 batchId = 3 sup|bj - Aj*xj| = 1.1102230246251565E-015
 batchId = 4 sup|bj - Aj*xj| = 4.4408920985006262E-016
  ============ Result X ===============
 x( 1 )= [ 1 ] 0.9999916593410789
 x( 1 )= [ 2 ] 0.5000383328163729
 x( 1 )= [ 3 ] 0.3333809377259567
 x( 1 )= [ 4 ] 0.2041067826368187

 x( 2 )= [ 1 ] 1.000026164043516
 x( 2 )= [ 2 ] 0.5000317024998334
 x( 2 )= [ 3 ] 0.3333561041470322
 x( 2 )= [ 4 ] 0.2041256247608860

 x( 3 )= [ 1 ] 1.000019144193291
 x( 3 )= [ 2 ] 0.5000164066433919
 x( 3 )= [ 3 ] 0.3333775041545496
 x( 3 )= [ 4 ] 0.2041369446005358

 x( 4 )= [ 1 ] 0.9999537374963444
 x( 4 )= [ 2 ] 0.5000653595450339
 x( 4 )= [ 3 ] 0.3333631223478223
 x( 4 )= [ 4 ] 0.2041274097506157
```

Top of page

Batched Sparse QR example with OpenACC Fortran

For the Fortran Module for the cuSOLVER SP (sparse) library, use the cusolver_mod.cuf used above as is, but please change the suffix of the file to cusolver_mod.f90. The driver routine (main.f90) for Batched Sparse QR decomposition using OpenACC is shown below.

One thing to keep in mind with the OpenACC directive is that if the actual argument passed when calling a (library) routine written in CUDA Fortran or CUDA C is a "device pointer", **host_data use_device** is used to inform the compiler of this. **()** use. If you use this directive correctly, porting an existing program-based program using OpenACC should not be a big burden.

Batched Sparse QR main program (main.f90)

```
! Copyright (C) HPC WORLD (Prometech Software, Inc. and GDEP Solutions, Inc.) All rights reserve
!
! Permission is hereby granted, free of charge, to any person obtaining a
! copy of this software and associated documentation files (the "Software"),
! to deal in the Software without restriction, including without limitation
! the rights to use, copy, modify, merge, publish, distribute, sublicense,
! and/or sell copies of the Software, and to permit persons to whom the
! Software is furnished to do so, subject to the following conditions:
!
! The above copyright notice and this permission notice shall be included in
! all copies or substantial portions of the Software.
!
! THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
! IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
! FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
! THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
! LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
! FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
! DEALINGS IN THE SOFTWARE.
!
program main
  use cusparse
  use cusolver_SP_QR
  implicit none
  integer, parameter :: idbg = 2 ! debug write enable
  integer, parameter :: m = 4, nnzA = 7
  integer, parameter :: batchsize = 4

  integer :: cusolver_status
  integer :: status1, status2, status3, status4, status5
  type(cusolverSpHandle) :: cusolver_Hndl
  type(cusparseMatDescr) :: descrA
  type(csrqrInfo) :: info

  integer :: csrRowPtrA(m+1), csrColIndA(nnzA)
  double precision :: csrValA(nnzA), b(m)

  double precision :: csrValABatch(nnzA*batchsize), bBatch(m*batchsize), xbatch(m*batchsize)

  integer(8) :: size_internal, size_qr
  character(c_char), allocatable :: pBuffer(:)

  !locals
  integer :: i, j, colidx, batchId
  integer::ierr_code
  double precision :: eps, Areg, breg, xreg

  !Result
  integer :: row, baseA, start, end, col
  double precision :: csrValAj
  double precision :: sup_res, r, Ax

  !Random Number
  double precision :: rnd

  ! | 1 |
  ! A = | 2 |
  ! | 3 |
  ! | 0.1 0.1 0.1 4 |
  !
  ! CSR of A is based 1 indexing <==== caution (CUSPARSE_INDEX_BASE_ONE)
  !
  ! b = [1 1 1 1]

! step 1: data setting
   csrRowPtrA(1:m+1) = (/1, 2, 3, 4, 8/)
   csrColIndA(1:nnzA) = (/1, 2, 3, 1, 2, 3, 4/)
   csrValA(1:nnzA) = (/1.0d0, 2.0d0, 3.0d0, 0.1d0, 0.1d0, 0.1d0, 4.0d0/)
   b(1:m) = (/1.0d0, 1.0d0, 1.0d0, 1.0d0/)

   print *, "sizeof(csrRowPtrA, csrRowPtrA, csrColIndA, csrColIndA)", &
             sizeof(csrRowPtrA(1)), sizeof(csrRowPtrA(1)), sizeof(csrColIndA(1)), sizeof(csrCol

!!call RANDOM_SEED()
! prepare Aj and bj on host
   do colidx = 1, nnzA
        Areg = csrValA(colidx)
        do batchId = 1, batchSize
           call random_number(rnd)
           eps = dble( mod(rnd,100.d0) + 1.0d0 ) * 1.0D-4
           csrValABatch((batchId-1)*nnzA + colidx) = Areg + eps
        enddo
   enddo

   do j = 1, m
        breg = b(j)
        do batchId = 1, batchSize
           call random_number(rnd)
           eps = dble( mod(rnd,100.d0) + 1.0d0 ) * 1.0D-4
           bBatch((batchId-1)*m + j) = breg + eps;
        enddo
   enddo

   xbatch = 0.d0 ! initilize

    if (idbg == 2) print *,"csrValABatch", csrValABatch
    if (idbg == 2) print *,"bBatch", bBatch

! step 2: create cusolver handle, qr info and matrix descriptor

    status1 = cusolverSpCreate(cusolver_Hndl)
          if (idbg == 1) print *, "status1 = ", status1
    status2 = cusparseCreateMatDescr(descrA)
          if (idbg == 1) print *, "status2 = ", status2
    status3 = cusparseSetMatType(descrA, CUSPARSE_MATRIX_TYPE_GENERAL)
          if (idbg == 1) print *, "status3 = ", status3
    status4 = cusparseSetMatIndexBase(descrA, CUSPARSE_INDEX_BASE_ONE) ! base=1
          if (idbg == 1) print *, "status4 = ", status4
    status5 = cusolverSpCreateCsrqrInfo(info)
          if (idbg == 1) print *, "status5 = ", status5

! step 3: copy Aj and bj to device
!d_csrValA = csrValABatch
!d_csrColIndA= csrColIndA
! d_csrRowPtrA= csrRowPtrA
!d_b = bBatch


! step 4: symbolic analysis

!$acc data copyin(csrValABatch, csrColIndA, csrRowPtrA) copyin(bBatch) copyout(xbatch)
!$acc host_data use_device(csrRowPtrA, csrColIndA)
    cusolver_status = cusolverSpXcsrqrAnalysisBatched( &
                      cusolver_Hndl, m, m, nnzA, descrA, csrRowPtrA, csrColIndA, info )
!$acc end host_data

    if (idbg == 1) then
      if (cusolver_status /= CUSOLVER_STATUS_SUCCESS) print *, "step4 cusolver_status = ", cusol
    end if

! step 5: prepare working space

!$acc host_data use_device(csrValABatch, csrRowPtrA, csrColIndA)
    cusolver_status = cusolverSpDcsrqrBufferInfoBatched( &
                      cusolver_Hndl, m, m, nnzA, descrA, csrValABatch, csrRowPtrA, csrColIndA, &
                      batchsize, info, size_internal, size_qr)
!$acc end host_data

    if (idbg == 1) then
      if (cusolver_status /= CUSOLVER_STATUS_SUCCESS) print *, "step5 cusolver_status = ", cusol
    end if

    print *, "numerical factorization needs internal data (bytes) =", size_internal
    print *, "numerical factorization needs working space (bytes) =", size_qr

    allocate(pBuffer(size_qr), stat=ierr_code) ! Bytes
      if (idbg == 1) print *, "step5: alloc ierr_code = ", ierr_code

!$acc enter data copyin(pBuffer)

! step 6: numerical factorization

!$acc host_data use_device(csrValABatch, csrRowPtrA, csrColIndA, bBatch, xbatch, pBuffer)
    cusolver_status = cusolverSpDcsrqrsvBatched( &
                      cusolver_Hndl, m, m, nnzA, &
                      descrA, csrValABatch, csrRowPtrA, csrColIndA, &
                      bBatch, xbatch, &
                      batchSize, &
                      info, &
```

```
                        pBuffer)
!$acc end host_data
!$acc exit data delete(pBuffer)
!$acc end data

    if (idbg == 1) then
      if (cusolver_status /= CUSOLVER_STATUS_SUCCESS) print *, "step6 cusolver_status = ", cusol


! step 7: check residual
!xBatch = [x0, x1, x2, ...]

!copy into CPU mem

    if (idbg == 2) print *, "xbatch=",xbatch

    print *, "******* Fortran Index Base is adjusted : BaseA = 0 ********************* "
    baseA = 0

    print *, " ============ Residual CHECK ==============="
    do batchId = 1 , batchsize
     ! measure |bj - Aj*Xj|
      sup_res = 0.d0
      do row = 1, m
        start = csrRowPtrA(row ) - baseA
        end = csrRowPtrA(row+1) - baseA
        Ax = 0.d0

        do colidx = start, end-1
          col = csrColIndA(colidx) - baseA
          Areg = csrValAbatch((batchId-1)*nnzA + colidx) ! Areg = csrValAj(colidx)
          Xreg = xBatch((batchId-1)*m + col) ! Xreg = xj(col)
          Ax = Ax + Areg * Xreg
        end do

        r = bBatch((batchID-1)*m + row) - Ax ! r = bj(row) -Ax
        ! sup_res = (sup_res > fabs(r))? sup_res : fabs(r);
        sup_res = max( abs(r), sup_res )

      end do
      print *, "batchId =", batchId," sup|bj - Aj*xj| =", sup_res
    end do


    print *, " ============ Result X ==============="
    do batchId = 1 , batchsize
       do row = 1, m
          print *, "x(", batchId, ")= [", row,"]", xBatch((batchId-1)*m + row)
       end do
       print *, ""
    end do
end
```

Simply copying and pasting the Makefile below will result in a make execution error. The [tab] delimiter in the makefile description has been converted to a space, so it is necessary to explicitly change the space to a [tab]. (Here is an example) Be sure to specify the –Mcuda option in the makefile below to instruct the linker to specify the CUDA system library list.

```
main : main.f90
 [ tab ]   pgf90 main.f90
```

Makefile (compilation method)

```
CC = nvcc
```

```
CFLAGS =
FC = pgf90
FFLAGS = -O2 -Minfo -acc -ta=tesla,cuda8.0,cc35,cc60 -Mcudalib=cusparse -Mcuda
LDFLAGS = -acc -ta=tesla,cuda8.0,cc35,cc60 -Mcuda

TARGET = SPsolve
INCLUDE =
LIBS = -lcusolver # -lgomp
# On Ubuntu, the following library path and -lgomp may be required
#LDIR = -L/usr/lib/gcc/x86_64-linux-gnu/5

modfile = cusolver_mod.f90
SRC1 = main.f90

# Define extensions to which suffix rules apply
.SUFFIXES: .f90 .cu .o

MODOBJ = $(modfile:.f90=.o)
OBJ1 = $(SRC1:.f90=.o)
MOD1 = cusolver_sp_qr.mod

RUN : $(TARGET)
                  @echo "===== Program run ====="
                  $(TARGET)
$(TARGET): $(MODOBJ) $(OBJ1)
                  $(FC) -o  $@ $^ $(LDFLAGS) $(LDIR) $(LIBS)


$(MOD1) : $(modfile)
                  $(FC) $(FFLAGS) -c $?
$(OBJ1): $(MOD1) $(SRC1)

.f90.o :
                  $(FC) -c $(FFLAGS) $<
.cu.o :
                  $(CC) -c $(CFLAGS) $<

.PHONY: clean
clean:
                @echo 'Cleaning up...'
            rm -f *.o  *.mod $(TARGET) *~
```

コンパイル&実行結果

```
[kato@photon32 OpenACC]$ make
pgf90 -c -O2 -Minfo -acc -ta=tesla,cuda8.0,cc35,cc60 -Mcudalib=cusparse -Mcuda cusolver_mod.f90
pgf90 -c -O2 -Minfo -acc -ta=tesla,cuda8.0,cc35,cc60 -Mcudalib=cusparse -Mcuda main.f90
main:
     67, Loop unrolled 5 times (completely unrolled)
     68, Array assignment / Forall at line 69 fused
         Loop unrolled 7 times (completely unrolled)
         Loop not vectorized: loop count too small
     70, Loop unrolled 4 times (completely unrolled)
     79, Loop not vectorized/parallelized: contains call
         FMA (fused multiply-add) instruction(s) generated
     88, Loop not vectorized/parallelized: contains call
         FMA (fused multiply-add) instruction(s) generated
     95, Loop unrolled 16 times (completely unrolled)
    122, Generating copyin(csrrowptra(:),bbatch(:),csrcolinda(:))
         Generating copyout(xbatch(:))
         Generating copyin(csrvalabatch(:))
    150, Generating enter data copyin(pbuffer(:))
    163, Generating exit data delete(pbuffer(:))
    182, Loop not vectorized/parallelized: contains call
    185, Outer loop unrolled 4 times (completely unrolled)
    190, Unrolled inner loop 4 times
```

```
         Generated 2 prefetch instructions for the loop
         Unrolled inner loop 4 times
         Generated 2 prefetch instructions for the loop
         Unrolled inner loop 4 times
         Generated 2 prefetch instructions for the loop
         Unrolled inner loop 4 times
         Generated 2 prefetch instructions for the loop
         FMA (fused multiply-add) instruction(s) generated
    208, Loop not vectorized/parallelized: contains call
pgf90 -o SPsolve cusolver_mod.o main.o -acc -ta=tesla,cuda8.0,cc35,cc60 -Mcuda -lcusolver
===== Program run =====
SPsolve
 sizeof(csrRowPtrA, csrRowPtrA,  csrColIndA, csrColIndA)            4
            4            4            4
 csrValABatch    1.000190792295671         2.000179731460909
    3.000132064769846        0.1001762263696543        0.1001628016430586
   0.1001500500225553         4.000169009995976         1.000119069206783
    2.000163682999292         3.000144817609972        0.1001437479734487
   0.1001670186653253        0.1001425331039673         4.000182114792401
    1.000106716529557         2.000158878275024         3.000165795016470
   0.1001463869055930        0.1001628171802322        0.1001307016646178
    4.000187350713939         1.000180008450829         2.000115906560827
    3.000160754590159        0.1001700415717258        0.1001531034351497
   0.1001216954552562         4.000196496681088
 bBatch    1.000182450045416         1.000166538252814
    1.000186841054682         1.000110375401338         1.000145236365631
    1.000145251688474         1.000116588275369         1.000155851050659
    1.000125862765850         1.000112255030956         1.000187784792441
    1.000198703067770         1.000133737619532         1.000188679946082
    1.000142956695574         1.000175135506380
 numerical factorization needs internal data (bytes) =                      864
 numerical factorization needs working space (bytes) =                   246528
 xbatch=   0.9999916593410789        0.5000383328163729
   0.3333809377259567        0.2041067826368187         1.000026164043516
   0.5000317024998334        0.3333561041470322        0.2041256247608860
    1.000019144193291        0.5000164066433919        0.3333775041545496
   0.2041369446005358        0.9999537374963444        0.5000653595450339
   0.3333631223478223        0.2041274097506157
 ******* Fortran Index Base is adjusted : BaseA = 0 *********************
  ============  Residual CHECK ===============
 batchId =            1  sup|bj - Aj*xj| =   8.8817841970012523E-016
 batchId =            2  sup|bj - Aj*xj| =   6.6613381477509392E-016
 batchId =            3  sup|bj - Aj*xj| =   1.1102230246251565E-015
 batchId =            4  sup|bj - Aj*xj| =   4.4408920985006262E-016
  ============  Result X ===============
 x(            1 )= [            1 ]   0.9999916593410789
 x(            1 )= [            2 ]   0.5000383328163729
 x(            1 )= [            3 ]   0.3333809377259567
 x(            1 )= [            4 ]   0.2041067826368187

 x(            2 )= [            1 ]    1.000026164043516
 x(            2 )= [            2 ]   0.5000317024998334
 x(            2 )= [            3 ]   0.3333561041470322
 x(            2 )= [            4 ]   0.2041256247608860

 x(            3 )= [            1 ]    1.000019144193291
 x(            3 )= [            2 ]   0.5000164066433919
 x(            3 )= [            3 ]   0.3333775041545496
 x(            3 )= [            4 ]   0.2041369446005358

 x(            4 )= [            1 ]   0.9999537374963444
 x(            4 )= [            2 ]   0.5000653595450339
 x(            4 )= [            3 ]   0.3333631223478223
 x(            4 )= [            4 ]   0.2041274097506157
```

ページの先頭へ

11. cuFFT example with OpenACC Fortran

OpenACC による cuFFT ライブラリを使用した 2 次元 FFT のプログラムの例を以下に示す。acc data データ領域の指定と acc host_data use_device でデバイスポインタを使用する変数を指定するだけで、FFT の処理が可能となる。

cuFFT example（cuFFT.f90）

```
program cufft2dTest
  use cufft
  use openacc
  integer, parameter :: m=768, n=512
  complex, allocatable  :: a(:,:),b(:,:),c(:,:)
  real, allocatable     :: r(:,:),q(:,:)
  integer :: iplan1, iplan2, iplan3, ierr

  allocate(a(m,n),b(m,n),c(m,n))
  allocate(r(m,n),q(m,n))

  a = 1; r = 1
  xmx = -99.0

  ierr = cufftPlan2D(iplan1,m,n,CUFFT_C2C)
!$acc data copyin(a)  create(b) copyout(c)
  !$acc host_data use_device(a,b,c)
  ierr = ierr + cufftExecC2C(iplan1,a,b,CUFFT_FORWARD)
  ierr = ierr + cufftExecC2C(iplan1,b,c,CUFFT_INVERSE)
  !$acc end host_data

  ! scale c
  !$acc kernels
  c = c / (m*n)
  !$acc end kernels
!$acc end data

  ! Check forward answer
  write(*,*) 'Max error C2C FWD: ', cmplx(maxval(real(b)) - sum(real(b)), &
                                        maxval(imag(b)))
  ! Check inverse answer
  write(*,*) 'Max error C2C INV: ', maxval(abs(a-c))

  ! Real transform
  ierr = ierr + cufftPlan2D(iplan2,m,n,CUFFT_R2C)
  ierr = ierr + cufftPlan2D(iplan3,m,n,CUFFT_C2R)

!$acc data copyin(r) copyout(q) create(b)
  !$acc host_data use_device(r,b,q)
  ierr = ierr + cufftExecR2C(iplan2,r,b)
  ierr = ierr + cufftExecC2R(iplan3,b,q)
  !$acc end host_data

  !$acc kernels
  xmx = maxval(abs(r-q/(m*n)))
  !$acc end kernels
!$acc end data

  ! Check R2C + C2R answer
  write(*,*) 'Max error R2C/C2R: ', xmx

  ierr = ierr + cufftDestroy(iplan1)
  ierr = ierr + cufftDestroy(iplan2)
  ierr = ierr + cufftDestroy(iplan3)

  if (ierr.eq.0) then
```

```
    print *,"test PASSED"
  else
    print *,"test FAILED"
  endif

end program cufft2dTest
```

コンパイルとリンクオプションには、必ず、-acc -Mcudalib=cufft を指定することが必要である。

コンパイル&実行結果

```
[kato@photon32 Example]$ pgf90 -acc -Minfo=accel -O2 -ta=tesla,cc60,cc35,cuda8.0 -Mcudalib=cufft
cufft2dtest:
      0, Accelerator kernel generated
         Generating Tesla code
     16, Generating copyin(a(:,:))
         Generating create(b(:,:))
         Generating copyout(c(:,:))
     24, Loop is parallelizable
         Accelerator kernel generated
         Generating Tesla code
         24, !$acc loop gang, vector(4) ! blockidx%y threadidx%y
             !$acc loop gang, vector(32) ! blockidx%x threadidx%x
     38, Generating copyin(r(:,:))
         Generating copyout(q(:,:))
         Generating create(b(:,:))
     45, Loop is parallelizable
         Accelerator scalar kernel generated
         Accelerator kernel generated
         Generating Tesla code
         45, !$acc loop gang, vector(4) ! blockidx%y threadidx%y
             !$acc loop gang, vector(32) ! blockidx%x threadidx%x
             Generating implicit reduction(max:r$r)

[kato@photon32 Example]$ a.out
 Max error C2C FWD:    (0.000000,0.000000)
 Max error C2C INV:      0.000000
 Max error R2C/C2R:      0.000000
 test PASSED
```

ページの先頭へ

12. cuRAND example (乱数発生)

cuRAND example from OpenACC Host Code

OpenACC による cuRAND ライブラリを使用したプログラムの例（XORWOW乱数) を以下に示す。ホストプログラムから CUDA ライブラリを利用するため、デバイス側で使用するデータのみを OpenACC の acc data で指定し、acc host_data use_device でデバイス側で使用する変数を指定するだけで利用可能となる。なお、curand 関数を使用するルーチンでは必ず use curand （インタフェース）を指定することが必要である。

cuRAND test from OpenACC Host Code（cuRand.f90）

```
program testcurand
! compile with the flags -ta=tesla -Mcuda -Mcudalib=curand
call cur1(1000, .true.); call cur1(1000, .false.)
call cur2(1000, .true.); call cur2(1000, .false.)
call cur3(1000, .true.); call cur3(1000, .false.)
```

```
end
!
subroutine cur1(n, onhost)
use curand    ! 必ず指定すること
integer :: a(n)      ! 整数
type(curandGenerator) :: g
integer(8) nbits
logical onhost, passing
a = 0
passing = .true.
if (onhost) then     ! host 側のライブラリも提供している
  istat = curandCreateGeneratorHost(g,CURAND_RNG_PSEUDO_XORWOW)
  istat = curandGenerate(g, a, n)
  istat = curandDestroyGenerator(g)
else                 ! GPU 側処理
  !$acc data copy(a)
  istat = curandCreateGenerator(g,CURAND_RNG_PSEUDO_XORWOW)
  !$acc host_data use_device(a)
  istat = curandGenerate(g, a, n)
  !$acc end host_data
  istat = curandDestroyGenerator(g)
  !$acc end data
endif
nbits = 0
do i = 1, n
  if (i.lt.10) print *,i,a(i)
  nbits = nbits + popcnt(a(i))
end do
print *,"Should be roughly half the bits set"
nbits = nbits / n
if ((nbits .lt. 12) .or. (nbits .gt. 20)) then
  passing = .false.
else
  print *,"nbits is ",nbits," which passes"
endif
if (passing) then
  print *,"Test PASSED"
else
  print *,"Test FAILED"
endif
end
!
subroutine cur2(n, onhost)
use curand
real :: a(n)     ! 単精度
type(curandGenerator) :: g
logical onhost, passing
a = 0.0
passing = .true.
if (onhost) then
  istat = curandCreateGeneratorHost(g,CURAND_RNG_PSEUDO_XORWOW)
  istat = curandGenerate(g, a, n)
  istat = curandDestroyGenerator(g)
else
  !$acc data copy(a)
  istat = curandCreateGenerator(g,CURAND_RNG_PSEUDO_XORWOW)
  !$acc host_data use_device(a)
  istat = curandGenerate(g, a, n)
  !$acc end host_data
  istat = curandDestroyGenerator(g)
  !$acc end data
endif
print *,"Should be uniform around 0.5"
do i = 1, n
  if (i.lt.10) print *,i,a(i)
  if ((a(i).lt.0.0) .or. (a(i).gt.1.0)) passing = .false.
end do
rmean = sum(a)/n
```

```
if ((rmean .lt. 0.4) .or. (rmean .gt. 0.6)) then
  passing = .false.
else
  print *,"Mean is ",rmean," which passes"
endif
if (passing) then
  print *,"Test PASSED"
else
  print *,"Test FAILED"
endif
end
!
subroutine cur3(n, onhost)
use curand
real(8) :: a(n)    ! 倍精度
type(curandGenerator) :: g
logical onhost, passing
a = 0.0d0
passing = .true.
if (onhost) then
  istat = curandCreateGeneratorHost(g,CURAND_RNG_PSEUDO_XORWOW)
  istat = curandGenerate(g, a, n)
  istat = curandDestroyGenerator(g)
else
  !$acc data copy(a)
  istat = curandCreateGenerator(g,CURAND_RNG_PSEUDO_XORWOW)
  !$acc host_data use_device(a)
  istat = curandGenerate(g, a, n)
  !$acc end host_data
  istat = curandDestroyGenerator(g)
  !$acc end data
endif
do i = 1, n
  if (i.lt.10) print *,i,a(i)
  if ((a(i).lt.0.0d0) .or. (a(i).gt.1.0d0)) passing = .false.
end do
rmean = sum(a)/n
if ((rmean .lt. 0.4d0) .or. (rmean .gt. 0.6d0)) then
  passing = .false.
else
  print *,"Mean is ",rmean," which passes"
endif
if (passing) then
  print *,"Test PASSED"
else
  print *,"Test FAILED"
endif
end
```

コンパイルとリンクオプションには、必ず、–acc –Mcudalib=curand を指定することが必要である。また、システム CUDA ライブラリもリンクする必要があるため、–Mcuda オプションも必須となる。

コンパイル＆実行結果

```
[kato@photon32]$ pgf90 -acc -Minfo=accel -O2 -ta=tesla,cc60,cc35,cuda8.0 -Mcudalib=curand -Mcuda
cur1:
     21, Generating copy(a(:))
cur2:
     60, Generating copy(a(:))
cur3:
     98, Generating copy(a(:))
[kato@photon32 cuRAND]$ a.out
            1  -1115749450
            2   -339329097
            3    167591721
```

```
           4   -133303299
           5   -321518802
           6   1917059131
           7  -1428853312
           8    472148682
           9   2019573489
 Should be roughly half the bits set
 nbits is                        16  which passes
 Test PASSED
           1  -1115749450
           2   -339329097
           3    167591721
           4   -133303299
           5   -321518802
           6   1917059131
           7  -1428853312
           8    472148682
           9   2019573489
 Should be roughly half the bits set
 nbits is                        16  which passes
 Test PASSED
 Should be uniform around 0.5
           1   0.7402194
           2   0.9209938
           3   3.9020490E-02
           4   0.9689629
           5   0.9251406
           6   0.4463501
           7   0.6673192
           8   0.1099307
           9   0.4702186
 Mean is    0.5014752      which passes
 Test PASSED
 Should be uniform around 0.5
           1   0.7402194
           2   0.9209938
           3   3.9020490E-02
           4   0.9689629
           5   0.9251406
           6   0.4463501
           7   0.6673192
           8   0.1099307
           9   0.4702186
 Mean is    0.5014752      which passes
 Test PASSED
           1   0.4384508447184235
           2   0.4603647022031429
           3   0.2502147080176417
           4   0.4947437677616333
           5   5.3011123716805220E-002
           6   0.3376992580306317
           7   0.3967625188337749
           8   0.8744186605648221
           9   0.4821668323147305
 Mean is    0.5150273      which passes
 Test PASSED
           1   0.4384508447184235
           2   0.4603647022031429
           3   0.2502147080176417
           4   0.4947437677616333
           5   5.3011123716805220E-002
           6   0.3376992580306317
           7   0.3967625188337749
           8   0.8744186605648221
           9   0.4821668323147305
 Mean is    0.5150273      which passes
 Test PASSED
```

ページの先頭へ

cuRAND example from OpenACC device code

OpenACC の計算領域で cuRAND のデバイスルーチン・ライブラリ関数（curand_uniform等）を使用する場合の例である。これは、ホストプログラム上で使用するものではなく、device 側専用のルーチンである。以下のプログラムでは、module mtests の中で、OpenACC によるデバイスコードを定義している。これは、グローバルサブルーチンとして他のルーチンから利用できる。ここでの技術的な留意点は、PGI Fortran 専用の cuRAND OpenACC 専用 MODULE インタフェース(openacc_curand) を use 文で指定する必要があるという点である。PGI は、cuRand 関数に対して openacc routine ディレクティブを指定した Fortran MODULE を提供している。

cuRAND test from OpenACC device Code（cuRand2.f90）

```
module mtests
  integer, parameter :: n = 1000
  contains
    subroutine testrand( a, b )
    use openacc_curand    ! OpenACC用のcurand I/F 必ず指定する
    real :: a(n), b(n)
    type(curandStateXORWOW) :: h
    integer(8) :: seed, seq, offset

    !$acc parallel num_gangs(1) vector_length(1) copy(a,b) private(h)
    seed = 12345            ! この時点でデバイスが側に処理が移る。 a, b 配列ともにデバイスへコピー
    seq = 0
    offset = 0
    call curand_init(seed, seq, offset, h)
    !$acc loop seq        ! 以下のループは sequential 実行
    do i = 1, n
      a(i) = curand_uniform(h)
      b(i) = curand_normal(h)
    end do
    !$acc end parallel    ! この時点で a(n), b(n) の値はホスト側に戻される
    return
    end subroutine
end module mtests

program t
use mtests
real :: a(n), b(n), c(n)
logical passing
a = 1.0
b = 2.0
passing = .true.
call testrand(a,b)    ! ホスト側から call する。引数a, b はホスト上の配列を渡す
c = a
print *,"Should be uniform around 0.5"
do i = 1, n
  if (i.lt.10) print *,i,c(i)
  if ((c(i).lt.0.0) .or. (c(i).gt.1.0)) passing = .false.
end do
rmean = sum(c)/n
if ((rmean .lt. 0.4) .or. (rmean .gt. 0.6)) then
  passing = .false.
else
  print *,"Mean is ",rmean," which passes"
endif
c = b
print *,"Should be normal around 0.0"
nc1 = 0;
nc2 = 0;
```

```
do i = 1, n
  if (i.lt.10) print *,i,c(i)
  if ((c(i) .gt. -4.0) .and. (c(i) .lt. 0.0)) nc1 = nc1 + 1
  if ((c(i) .gt.  0.0) .and. (c(i) .lt. 4.0)) nc2 = nc2 + 1
end do
print *,"Found on each side of zero ",nc1,nc2
if (abs(nc1-nc2) .gt. (n/10)) npassing = .false.
rmean = sum(c,mask=abs(c).lt.4.0)/n
if ((rmean .lt. -0.1) .or. (rmean .gt. 0.1)) then
  passing = .false.
else
  print *,"Mean is ",rmean," which passes"
endif

if (passing) then
  print *,"Test PASSED"
else
  print *,"Test FAILED"
endif
end program
```

コンパイルとリンクオプションには、–acc –Mcudalib=cusparse –Mcuda のオプションの指定が必要である。OpenACC 領域内で cuRAND ライブラリを利用する場合は、PGI Fortran OpenACC 専用の cuRAND のデバイスルーチン・ライブラリをリンクする必要があるため、リンクオプションに注意が必要である。リンク時に NVIDIA LLVM 配下のリンク環境を使うのではなく、従来のリンケージ処理を行うために、–ta=tesla,**nollvm** サブオプションの指定が必要となる。

コンパイル＆実行結果

```
[kato@photon32 cuRAND]$ pgf90 -acc -Minfo=accel -O2 -ta=tesla,cc60,cc35,cuda8.0,nollvm  cuRand2.
testrand:
     10, Generating copy(b(:),a(:))
         Accelerator kernel generated
         Generating Tesla code
         16, !$acc loop seq
     10, CUDA shared memory used for h

[kato@photon32 cuRAND]$ a.out
 Should be uniform around 0.5
            1   0.2988985
            2   0.4019704
            3   0.7425825
            4   0.7073491
            5   0.5512256
            6   0.4850157
            7   9.7107515E-02
            8   0.3596036
            9   0.5777864
 Mean is    0.4975990      which passes
 Should be normal around 0.0
            1    1.348672
            2  -0.3331721
            3    1.273486
            4   -1.008033
            5  -0.6210795
            6  -0.5052772
            7    1.598479
            8    1.133891
            9   -1.831225
 Found on each side of zero           469          531
 Mean is    4.2783763E-02  which passes
 Test PASSEDD
```

ページの先頭へ

13. cuSPARSE from OpenACC Host Code

ホスト用プログラムからデバイス側で処理する cuSPARSE ライブラリの利用例を示す。

cuSPARSE ルーチンの処理には直接関係ないが、以下の例では、cusparseSetStream を使って個々の cuSPARSE ライブラリルーチンによって使われる stream をセットしている。これを行うことで、対象となるハンドル h の cuSPARSE 関数は分離された stream で処理され、GPU 上で自動的なオーバラップ処理が可能となる。このアプローチはとりわけ、シングルタスクの実行が相対的に小さな場合で GPU 内の処理容量を満たさないような時に有効である。あるいは、計算とデータ転送が並行に行われるような場面で使用される。

cuSPARSE from OpenACC Host Code（cuSPARSE.f90）

```
program sparseMatVec
    integer n
    n = 25 ! # rows/cols in dense matrix
    call sparseMatVecSub1(n)
    n = 45 ! # rows/cols in dense matrix
    call sparseMatVecSub1(n)
end program

subroutine sparseMatVecSub1(n)
  use openacc
  use cusparse

  implicit none

  integer n

  ! dense data
  real(4), allocatable :: Ade(:,:), x(:), y(:)

  ! sparse CSR arrays
  real(4), allocatable :: csrValA(:)
  integer, allocatable :: nnzPerRowA(:), csrRowPtrA(:), csrColIndA(:)

  allocate(Ade(n,n), x(n), y(n))
  allocate(csrValA(n))
  allocate(nnzPerRowA(n), csrRowPtrA(n+1), csrColIndA(n))

  call sparseMatVecSub2(Ade, x, y, csrValA, nnzPerRowA, csrRowPtrA, &
                                                         csrColIndA, n)
  deallocate(Ade)
  deallocate(x)
  deallocate(y)
  deallocate(csrValA)
  deallocate(nnzPerRowA)
  deallocate(csrRowPtrA)
  deallocate(csrColIndA)
end subroutine

subroutine sparseMatVecSub2(Ade, x, y, csrValA, nnzPerRowA, csrRowPtrA, &
                                                             csrColIndA, n)
  use openacc
  use cusparse

  implicit none

  ! dense data
  real(4) :: Ade(n,n), x(n), y(n)

  ! sparse CSR arrays
```

```
    real(4) :: csrValA(n)
    integer :: nnzPerRowA(n), csrRowPtrA(n+1), csrColIndA(n)

    integer :: n, nnz, status, i
    type(cusparseHandle) :: h
    type(cusparseMatDescr) :: descrA

    ! parameters
    real(4) :: alpha, beta

    ! result
    real(4) :: xerr

    ! initalize CUSPARSE and matrix descriptor
    status = cusparseCreate(h)
    if (status /= CUSPARSE_STATUS_SUCCESS) &
         write(*,*) 'cusparseCreate error: ', status
    status = cusparseCreateMatDescr(descrA)
    status = cusparseSetMatType(descrA, &
         CUSPARSE_MATRIX_TYPE_GENERAL)
    status = cusparseSetMatIndexBase(descrA, &
         CUSPARSE_INDEX_BASE_ONE)
    status = cusparseSetStream(h, acc_get_cuda_stream(acc_async_sync))  ! cusparseが利用する一つのcu

    !$acc data create(Ade, x, y, csrValA, nnzPerRowA, csrRowPtrA, csrColIndA)

    ! Initialize matrix (upper circular shift matrix) ! Circulant matrix
    !$acc kernels
    Ade = 0.0
    do i = 1, n-1
       Ade(i,i+1) = 1.0
    end do
    Ade(n,1) = 1.0

    ! Initialize vectors and constants
    do i = 1, n
       x(i) = i
    enddo
    y = 0.0
    !$acc end kernels

    !$acc update host(x)
    write(*,*) 'Original vector:'
    write(*,'(5(1x,f7.2))') x

    ! convert matrix from dense to csr format
    !$acc host_data use_device(Ade, nnzPerRowA, csrValA, csrRowPtrA, csrColIndA)
    status = cusparseSnnz_v2(h, CUSPARSE_DIRECTION_ROW, &
         n, n, descrA, Ade, n, nnzPerRowA, nnz)
    status = cusparseSdense2csr(h, n, n, descrA, Ade, n, &
         nnzPerRowA, csrValA, csrRowPtrA, csrColIndA)
    !$acc end host_data

    ! A is upper circular shift matrix
    ! y = alpha*A*x + beta*y
    alpha = 1.0
    beta = 0.0
    !$acc host_data use_device(csrValA, csrRowPtrA, csrColIndA, x, y)
    status = cusparseScsrmv(h, CUSPARSE_OPERATION_NON_TRANSPOSE, &
         n, n, n, alpha, descrA, csrValA, csrRowPtrA, &
         csrColIndA, x, beta, y)
    !$acc end host_data

    !$acc wait
    write(*,*) 'Shifted vector:'
    write(*,'(5(1x,f7.2))') y

    ! shift-down y and add original x
```

```
  ! A' is lower circular shift matrix
  ! x = alpha*A'*y + beta*x
  beta = -1.0
  !$acc host_data use_device(csrValA, csrRowPtrA, csrColIndA, x, y)
  status = cusparseScsrmv(h, CUSPARSE_OPERATION_TRANSPOSE, &
       n, n, n, alpha, descrA, csrValA, csrRowPtrA, &
       csrColIndA, y, beta, x)
  !$acc end host_data

  !$acc kernels
  xerr = maxval(abs(x))
  !$acc end kernels
  !$acc end data

  write(*,*) 'Max error = ', xerr
  if (xerr.le.1.e-5) then
    write(*,*) 'Test PASSED'
  else
    write(*,*) 'Test FAILED'
  endif

end subroutine
```

コンパイルとリンクオプションには、-acc -Mcudalib=cusparse -Mcuda が必要となる。

コンパイル＆実行結果

```
[kato@photon32 cuSPARSE]$ pgf90 -Minfo=accel -acc -O2 -ta=tesla,cc60,cuda8.0 -Mcudalib=cusparse
sparsematvecsub2:
      0, Accelerator kernel generated
         Generating Tesla code
     74, Generating create(nnzperrowa(:),x(:),y(:),csrrowptra(:),csrvala(:),csrcolinda(:),ade(:,
     78, Loop is parallelizable
         Accelerator kernel generated
         Generating Tesla code
         78, !$acc loop gang, vector(4) ! blockidx%y threadidx%y
             !$acc loop gang, vector(32) ! blockidx%x threadidx%x
     79, Loop is parallelizable
         Accelerator kernel generated
         Generating Tesla code
         79, !$acc loop gang, vector(128) ! blockidx%x threadidx%x
     81, Accelerator scalar kernel generated
     85, Loop is parallelizable
         Accelerator kernel generated
         Generating Tesla code
         85, !$acc loop gang, vector(128) ! blockidx%x threadidx%x
     88, Loop is parallelizable
         Accelerator kernel generated
         Generating Tesla code
         88, !$acc loop gang, vector(128) ! blockidx%x threadidx%x
     91, Generating update self(x(:))
    128, Loop is parallelizable
         Accelerator scalar kernel generated
         Accelerator kernel generated
         Generating Tesla code
        128, !$acc loop gang, vector(128) ! blockidx%x threadidx%x
             Generating implicit reduction(max:x$r)
[kato@photon32 cuSPARSE]$ a.out
 Original vector:
    1.00     2.00     3.00     4.00     5.00
    6.00     7.00     8.00     9.00    10.00
   11.00    12.00    13.00    14.00    15.00
   16.00    17.00    18.00    19.00    20.00
   21.00    22.00    23.00    24.00    25.00
 Shifted vector:
    0.00     0.00     0.00     0.00     0.00
```

```
      0.00    0.00    0.00    0.00    0.00
      0.00    0.00    0.00    0.00    0.00
      0.00    0.00    0.00    0.00    0.00
      0.00    0.00    0.00    0.00    0.00
   Max error =     0.000000
   Test PASSED
   Original vector:
      1.00    2.00    3.00    4.00    5.00
      6.00    7.00    8.00    9.00   10.00
     11.00   12.00   13.00   14.00   15.00
     16.00   17.00   18.00   19.00   20.00
     21.00   22.00   23.00   24.00   25.00
     26.00   27.00   28.00   29.00   30.00
     31.00   32.00   33.00   34.00   35.00
     36.00   37.00   38.00   39.00   40.00
     41.00 42.00 43.00 44.00 45.00
   Shifted vector:
      0.00 0.00 0.00 0.00 0.00
      0.00 0.00 0.00 0.00 0.00
      0.00 0.00 0.00 0.00 0.00
      0.00 0.00 0.00 0.00 0.00
      0.00 0.00 0.00 0.00 0.00
      0.00 0.00 0.00 0.00 0.00
      0.00 0.00 0.00 0.00 0.00
      0.00 0.00 0.00 0.00 0.00
      0.00 0.00 0.00 0.00 0.00
   Max error = 0.000000
   Test PASSED
```

Top of page

Go to previous chapter

Next chapter

OpenACC programming index

Top of page