CUDA SDK — Libraries, Numerical Accuracy

Siegfried Höfinger

ASC Research Center, TU Wien

October 20, 2025

→ https://tinyurl.com/cudafordummies/ii/l4/notes-l4.pdf

Outline

To the Fortran-Minded
CUDA SDK

- Fortran is the language of choice in many scientific code development projects

→ https://developer.nvidia.com/hpc-sdk

To the Fortran-Minded
CUDA SDK

- Fortran is the language of choice in many scientific code development projects
- Professional coverage from the NVIDIA hpc SDK — formerly portland group compilers — now an entire suite of compilers, libraries and developmental tools

→ https://developer.nvidia.com/hpc-sdk

To the Fortran-Minded
CUDA SDK

- Fortran is the language of choice in many scientific code development projects
- Professional coverage from the NVIDIA hpc SDK — formerly portland group compilers — now an entire suite of compilers, libraries and developmental tools
- NVFORTRAN, NVC++, NVC and NVCC compilers

→ https://developer.nvidia.com/hpc-sdk

To the Fortran-Minded

CUDA SDK

- Fortran is the language of choice in many scientific code development projects
- Professional coverage from the NVIDIA hpc SDK — formerly portland group compilers — now an entire suite of compilers, libraries and developmental tools
- NVFORTRAN, NVC++, NVC and NVCC compilers
- Simply download from `https://developer.nvidia.com/nvidia-hpc-sdk-251-downloads`

→ `https://developer.nvidia.com/hpc-sdk`

To the Fortran-Minded
CUDA SDK

- Fortran is the language of choice in many scientific code development projects
- Professional coverage from the NVIDIA hpc SDK — formerly portland group compilers — now an entire suite of compilers, libraries and developmental tools
- NVFORTRAN, NVC++, NVC and NVCC compilers
- Simply download from https://developer.nvidia.com/nvidia-hpc-sdk-251-downloads
- Follow guidelines in https://docs.nvidia.com/hpc-sdk/archive/25.1/hpc-sdk-install-guide/index.html

→ https://developer.nvidia.com/hpc-sdk

To the Fortran-Minded
CUDA SDK

- Fortran is the language of choice in many scientific code development projects
- Professional coverage from the NVIDIA hpc SDK — formerly portland group compilers — now an entire suite of compilers, libraries and developmental tools
- NVFORTRAN, NVC++, NVC and NVCC compilers
- Simply download from `https://developer.nvidia.com/nvidia-hpc-sdk-251-downloads`
- Follow guidelines in `https://docs.nvidia.com/hpc-sdk/archive/25.1/hpc-sdk-install-guide/index.html`
- Install and load the module to get ready

→ https://developer.nvidia.com/hpc-sdk

To the Fortran-Minded

CUDA SDK

- Fortran is the language of choice in many scientific code development projects
- Professional coverage from the NVIDIA hpc SDK — formerly portland group compilers — now an entire suite of compilers, libraries and developmental tools
- NVFORTRAN, NVC++, NVC and NVCC compilers
- Simply download from `https://developer.nvidia.com/nvidia-hpc-sdk-251-downloads`
- Follow guidelines in `https://docs.nvidia.com/hpc-sdk/archive/25.1/hpc-sdk-install-guide/index.html`
- Install and load the module to get ready
- ```
  cuda-zen sh@n3073-004:~$   module purge
  cuda-zen sh@n3073-004:~$   module load nvhpc/25.1-gcc-9.5.0-efr6qch
  ```

→ `https://developer.nvidia.com/hpc-sdk`
  ```

To the Fortran-Minded cont.

CUDA SDK cont.

vecadd.cuf

```
! host
program t1
use cudafor
use myvecadd
    integer, parameter :: n = 100
    integer :: i
    real, allocatable, device :: da(:), db(:), dc(:)
    real :: ha(n), hb(n), hc(n)
    istat = cudaSetDevice(0)
    allocate(da(n))
    ...
    da = ha
    ...
    call vecadd<<< 1, n >>>(da, db, dc)
    ha = da
    ...
    deallocate(da)
    ...
end program t1
```

```
! kernel
module myvecadd
contains
attributes(global) &
subroutine vecadd( a, b, c )
    integer :: i
    real :: a(*), b(*), c(*)

    i = (blockidx%x-1) * blockdim%x + threadidx%x
    c(i) = a(i) + b(i)
end subroutine vecadd
end module myvecadd
```

→ https://tinyurl.com/cudafordummies/ii/l4/vecadd.cuf

To the Fortran-Minded cont.

CUDA SDK cont.

vecadd.cuf

```
! host
program t1
use cudafor
use myvecadd
    integer, parameter :: n = 100
    integer :: i
    real, allocatable, device :: da(:), db(:), dc(:)
    real :: ha(n), hb(n), hc(n)
    istat = cudaSetDevice(0)
    allocate(da(n))
    ...
    da = ha
    ...
    call vecadd<<< 1, n >>>(da, db, dc)
    ha = da
    ...
    deallocate(da)
    ...
end program t1
```

Select GPU 0

```
! kernel
module myvecadd
contains
attributes(global) &
subroutine vecadd( a, b, c )
    integer :: i
    real :: a(*), b(*), c(*)

    i = (blockidx%x-1) * blockdim%x + threadidx%x
    c(i) = a(i) + b(i)
end subroutine vecadd
end module myvecadd
```

→ https://tinyurl.com/cudafordummies/ii/l4/vecadd.cuf

vecadd.cuf

```
! host
program t1
use cudafor
use myvecadd
    integer, parameter :: n = 100
    integer :: i
    real, allocatable, device :: da(:), db(:), dc(:)
    real :: ha(n), hb(n), hc(n)
    istat = cudaSetDevice(0)
    allocate(da(n))
    ...
    da = ha
    ...
    call vecadd<<< 1, n >>>(da, db, dc)
    ha = da
    ...
    deallocate(da)
    ...
end program t1
```

Select GPU 0

Simple htod/dtoh copies

```
! kernel
module myvecadd
contains
attributes(global) &
subroutine vecadd( a, b, c )
    integer :: i
    real :: a(*), b(*), c(*)

    i = (blockidx%x-1) * blockdim%x + threadidx%x
    c(i) = a(i) + b(i)
end subroutine vecadd
end module myvecadd
```

→ https://tinyurl.com/cudafordummies/ii/l4/vecadd.cuf

vecadd.cuf

```
! host
program t1
use cudafor
use myvecadd
    integer, parameter :: n = 100
    integer :: i
    real, allocatable, device :: da(:), db(:), dc(:)
    real :: ha(n), hb(n), hc(n)
    istat = cudaSetDevice(0)
    allocate(da(n))
    ...
    da = ha
    ...
    call vecadd<<< 1, n >>>(da, db, dc)
    ha = da
    ...
    deallocate(da)
    ...
end program t1
```

Select GPU 0

Simple htod/dtoh copies

```
! kernel
module myvecadd
contains
attributes(global) &
subroutine vecadd( a, b, c )
    integer :: i
    real :: a(*), b(*), c(*)

    i = (blockidx%x-1) * blockdim%x + threadidx%x
    c(i) = a(i) + b(i)
end subroutine vecadd
end module myvecadd
```

Slightly different syntax

→ https://tinyurl.com/cudafordummies/ii/l4/vecadd.cuf

ASC Austrian Scientific Computing

vecadd.cuf

```
! host
program t1
use cudafor
use myvecadd
    integer, parameter :: n = 100
    integer :: i
    real, allocatable, device :: da(:), db(:), dc(:)
    real :: ha(n), hb(n), hc(n)
    istat = cudaSetDevice(0)
    allocate(da(n))
    ...
    da = ha
    ...
    call vecadd<<< 1, n >>>(da, db, dc)
    ha = da
    ...
    deallocate(da)
    ...
end program t1
```

Select GPU 0

Simple htod/dtoh copies

```
! kernel
module myvecadd
contains
attributes(global) &
subroutine vecadd( a, b, c )
    integer :: i
    real :: a(*), b(*), c(*)

    i = (blockidx%x-1) * blockdim%x + threadidx%x
    c(i) = a(i) + b(i)
end subroutine vecadd
end module myvecadd
```

Slightly different syntax

Fortran counting 1,2,3...

→ https://tinyurl.com/cudafordummies/ii/l4/vecadd.cuf

To the Fortran-Minded cont.

CUDA SDK cont.

```
cuda−zen sh@n3073−004:~$  nvfortran vecadd.cuf
cuda−zen sh@n3073−004:~$  ./a.out

           1    1.000000        99.00000        100.0000
           2    2.000000        98.00000        100.0000
           3    3.000000        97.00000        100.0000
           4    4.000000        96.00000        100.0000
           5    5.000000        95.00000        100.0000
          ........
          98    98.00000        2.000000        100.0000
          99    99.00000        1.000000        100.0000
         100    100.0000        0.000000        100.0000
```

→ https://tinyurl.com/cudafordummies/ii/l4/vecadd.cuf

To the Fortran-Minded cont.

CUDA SDK cont.

- Frequently — only a single Fortran subroutine/function needs to be ported

→ https://www.pgroup.com/resources/docs/18.4/pdf/pgi18cudaforug.pdf

- Frequently — only a single Fortran subroutine/function needs to be ported
- Basic feasibility studies can already be done with the help of short C routines called from Fortran

→ https://www.pgroup.com/resources/docs/18.4/pdf/pgi18cudaforug.pdf

To the Fortran-Minded cont.

CUDA SDK cont.

- Frequently — only a single Fortran subroutine/function needs to be ported
- Basic feasibility studies can already be done with the help of short C routines called from Fortran
- The usual Fortran↔C interfacing rules apply

→ https://www.pgroup.com/resources/docs/18.4/pdf/pgi18cudaforug.pdf

To the Fortran-Minded cont.
CUDA SDK cont.

- Frequently — only a single Fortran subroutine/function needs to be ported
- Basic feasibility studies can already be done with the help of short C routines called from Fortran
- The usual Fortran↔C interfacing rules apply
- Memory management is more complicated, would prefer cudaMallocManaged()

→ https://www.pgroup.com/resources/docs/18.4/pdf/pgi18cudaforug.pdf

To the Fortran-Minded cont.

CUDA SDK cont.

fvcadd.f

```
PROGRAM FVCADD
IMPLICIT NONE
INTEGER N
PARAMETER (N = 100)
INTEGER I
REAL A(N), B(N), C(N)

DO I=1, N
   A(I) = REAL(I)
   B(I) = REAL(N) - REAL(I)
   C(I) = REAL(0)
ENDDO

CALL NTMDTR(A, B, C, N)

DO I=1, N
   WRITE(6, '(I6F12.6)') I, C(I)
ENDDO

END
```

ntmdtr.cu

```
__global__ void VecAdd(float *A, float *B, float *C)
{
   int i = threadIdx.x;
   C[i] = A[i] + B[i];
}
extern "C" void ntmdtr_(float *A, float *B, float *C, int *N)
{
   dim3 numBlocks, threadsPerBlock;
   float *AD, *BD, *CD;
   threadsPerBlock.x = *N;
   numBlocks.x = 1;
   cudaMalloc((void **) &AD, (*N) * sizeof(float));
   cudaMalloc((void **) &BD, (*N) * sizeof(float));
   cudaMalloc((void **) &CD, (*N) * sizeof(float));
   cudaMemcpy(AD, A, (*N) * sizeof(float), cudaMemcpyHostToDevice);
   cudaMemcpy(BD, B, (*N) * sizeof(float), cudaMemcpyHostToDevice);
   cudaMemcpy(CD, C, (*N) * sizeof(float), cudaMemcpyHostToDevice);
   VecAdd <<< numBlocks, threadsPerBlock >>> (AD, BD, CD);
   cudaDeviceSynchronize();
   cudaMemcpy(C, CD, (*N) * sizeof(float), cudaMemcpyDeviceToHost);
   cudaFree(AD);
   cudaFree(BD);
   cudaFree(CD);
   return;
}
```

To the Fortran-Minded cont.

CUDA SDK cont.

fvcadd.f

```
PROGRAM FVCADD
IMPLICIT NONE
INTEGER N
PARAMETER (N = 100)
INTEGER I
REAL A(N), B(N), C(N)

DO I=1, N
   A(I) = REAL(I)
   B(I) = REAL(N) - REAL(I)
   C(I) = REAL(0)
ENDDO

CALL NTMDTR(A, B, C, N)

DO I=1, N
   WRITE(6, '(I6F12.6)') I, C(I)
ENDDO

END
```

Intermediator in C

ntmdtr.cu

```
__global__ void VecAdd(float *A, float *B, float *C)
{
   int i = threadIdx.x;
   C[i] = A[i] + B[i];
}
extern "C" void ntmdtr_(float *A, float *B, float *C, int *N)
{
   dim3 numBlocks, threadsPerBlock;
   float *AD, *BD, *CD;
   threadsPerBlock.x = *N;
   numBlocks.x = 1;
   cudaMalloc((void **) &AD, (*N) * sizeof(float));
   cudaMalloc((void **) &BD, (*N) * sizeof(float));
   cudaMalloc((void **) &CD, (*N) * sizeof(float));
   cudaMemcpy(AD, A, (*N) * sizeof(float), cudaMemcpyHostToDevice);
   cudaMemcpy(BD, B, (*N) * sizeof(float), cudaMemcpyHostToDevice);
   cudaMemcpy(CD, C, (*N) * sizeof(float), cudaMemcpyHostToDevice);
   VecAdd <<< numBlocks, threadsPerBlock >>> (AD, BD, CD);
   cudaDeviceSynchronize();
   cudaMemcpy(C, CD, (*N) * sizeof(float), cudaMemcpyDeviceToHost);
   cudaFree(AD);
   cudaFree(BD);
   cudaFree(CD);
   return;
}
```

To the Fortran-Minded cont.

CUDA SDK cont.

fvcadd.f

```
PROGRAM FVCADD
IMPLICIT NONE
INTEGER N
PARAMETER (N = 100)
INTEGER I
REAL A(N), B(N), C(N)

DO I=1, N
   A(I) = REAL(I)
   B(I) = REAL(N) - REAL(I)
   C(I) = REAL(0)
ENDDO

CALL NTMDTR(A, B, C, N)

DO I=1, N
   WRITE(6, '(I6F12.6)') I, C(I)
ENDDO

END
```

Intermediator in C

ntmdtr.cu

```
__global__ void VecAdd(float *A, float *B, float *C)
{
   int i = threadIdx.x;
   C[i] = A[i] + B[i];
}
extern "C" void ntmdtr_(float *A, float *B, float *C, int *N)
{
   dim3 numBlocks, threadsPerBlock;
   float *AD, *BD, *CD;
   threadsPerBlock.x = *N;
   numBlocks.x = 1;
   cudaMalloc((void **) &AD, (*N) * sizeof(float));
   cudaMalloc((void **) &BD, (*N) * sizeof(float));
   cudaMalloc((void **) &CD, (*N) * sizeof(float));
   cudaMemcpy(AD, A, (*N) * sizeof(float), cudaMemcpyHostToDevice);
   cudaMemcpy(BD, B, (*N) * sizeof(float), cudaMemcpyHostToDevice);
   cudaMemcpy(CD, C, (*N) * sizeof(float), cudaMemcpyHostToDevice);
   VecAdd <<< numBlocks, threadsPerBlock >>> (AD, BD, CD);
   cudaDeviceSynchronize();
   cudaMemcpy(C, CD, (*N) * sizeof(float), cudaMemcpyDeviceToHost);
   cudaFree(AD);
   cudaFree(BD);
   cudaFree(CD);
   return;
}
```

Trailing underscore !

To the Fortran-Minded cont.

CUDA SDK cont.

fvcadd.f

```
PROGRAM FVCADD
IMPLICIT NONE
INTEGER N
PARAMETER (N = 100)
INTEGER I
REAL A(N), B(N), C(N)

DO I=1, N
   A(I) = REAL(I)
   B(I) = REAL(N) - REAL(I)
   C(I) = REAL(0)
ENDDO

CALL NTMDTR(A, B, C, N)

DO I=1, N
   WRITE(6, '(I6F12.6)') I, C(I)
ENDDO

END
```

Intermediator in C

ntmdtr.cu

```
__global__ void VecAdd(float *A, float *B, float *C)
{
   int i = threadIdx.x;
   C[i] = A[i] + B[i];
}
extern "C" void ntmdtr_(float *A, float *B, float *C, int *N)
{
   dim3 numBlocks, threadsPerBlock;
   float *AD, *BD, *CD;
   threadsPerBlock.x = *N;
   numBlocks.x = 1;
   cudaMalloc((void **) &AD, (*N) * sizeof(float));
   cudaMalloc((void **) &BD, (*N) * sizeof(float));
   cudaMalloc((void **) &CD, (*N) * sizeof(float));
   cudaMemcpy(AD, A, (*N) * sizeof(float), cudaMemcpyHostToDevice);
   cudaMemcpy(BD, B, (*N) * sizeof(float), cudaMemcpyHostToDevice);
   cudaMemcpy(CD, C, (*N) * sizeof(float), cudaMemcpyHostToDevice);
   VecAdd <<< numBlocks, threadsPerBlock >>> (AD, BD, CD);
   cudaDeviceSynchronize();
   cudaMemcpy(C, CD, (*N) * sizeof(float), cudaMemcpyDeviceToHost);
   cudaFree(AD);
   cudaFree(BD);
   cudaFree(CD);
   return;
}
```

Trailing underscore !

Call by Reference

To the Fortran-Minded cont.

CUDA SDK cont.

fvcadd.f

```fortran
PROGRAM FVCADD
IMPLICIT NONE
INTEGER N
PARAMETER (N = 100)
INTEGER I
REAL A(N), B(N), C(N)

DO I=1, N
   A(I) = REAL(I)
   B(I) = REAL(N) - REAL(I)
   C(I) = REAL(0)
ENDDO

CALL NTMDTR(A, B, C, N)

DO I=1, N
   WRITE(6, '(I6F12.6)') I, C(I)
ENDDO

END
```

Intermediator in C

ntmdtr.cu

```c
__global__ void VecAdd(float *A, float *B, float *C)
{
   int i = threadIdx.x;
   C[i] = A[i] + B[i];
}
extern "C" void ntmdtr_(float *A, float *B, float *C, int *N)
{
   dim3 numBlocks, threadsPerBlock;
   float *AD, *BD, *CD;
   threadsPerBlock.x = *N;
   numBlocks.x = 1;
   cudaMalloc((void **) &AD, (*N) * sizeof(float));
   cudaMalloc((void **) &BD, (*N) * sizeof(float));
   cudaMalloc((void **) &CD, (*N) * sizeof(float));
   cudaMemcpy(AD, A, (*N) * sizeof(float), cudaMemcpyHostToDevice);
   cudaMemcpy(BD, B, (*N) * sizeof(float), cudaMemcpyHostToDevice);
   cudaMemcpy(CD, C, (*N) * sizeof(float), cudaMemcpyHostToDevice);
   VecAdd <<< numBlocks, threadsPerBlock >>> (AD, BD, CD);
   cudaDeviceSynchronize();
   cudaMemcpy(C, CD, (*N) * sizeof(float), cudaMemcpyDeviceToHost);
   cudaFree(AD);
   cudaFree(BD);
   cudaFree(CD);
   return;
}
```

Trailing underscore !

Call by Reference

Extra data transfer ↔

To the Fortran-Minded cont.

CUDA SDK cont.

```
cuda—zen sh@n3073—004:~$  ls

fvcadd.f ntmdtr.cu

cuda—zen sh@n3073—004:~$  gfortran -c fvcadd.f
cuda—zen sh@n3073—004:~$  nvcc -c ntmdtr.cu
cuda—zen sh@n3073—004:~$  nvcc fvcadd.o ntmdtr.o -lcudart -lgfortran
cuda—zen sh@n3073—004:~$  ./a.out

     1  100.000000
     2  100.000000
     3  100.000000
     4  100.000000
     5  100.000000
     6  100.000000
     7  100.000000
     ....
    98  100.000000
    99  100.000000
   100  100.000000
```

→ https://www.olcf.ornl.gov/tutorials/compiling-mixed-gpu-and-cpu-code
→ https://tinyurl.com/cudafordummies/ii/l4/fvcadd.f
→ https://tinyurl.com/cudafordummies/ii/l4/ntmdtr.cu

To the Fortran-Minded cont.

CUDA SDK cont.

```
cuda-zen sh@n3073-004:~$  ls

fvcadd.f ntmdtr.cu

cuda-zen sh@n3073-004:~$  gfortran -c fvcadd.f
cuda-zen sh@n3073-004:~$  nvcc -c ntmdtr.cu
cuda-zen sh@n3073-004:~$  nvcc fvcadd.o ntmdtr.o -lcudart -lgfortran
cuda-zen sh@n3073-004:~$  ./a.out

      1  100.000000
      2  100.000000
      3  100.000000
      4  100.000000
      5  100.000000
      6  100.000000
      7  100.000000
      ....
     98  100.000000
     99  100.000000
    100  100.000000
```

Name mangling 2b considered !

→ https://www.olcf.ornl.gov/tutorials/compiling-mixed-gpu-and-cpu-code
→ https://tinyurl.com/cudafordummies/ii/l4/fvcadd.f
→ https://tinyurl.com/cudafordummies/ii/l4/ntmdtr.cu

To the Fortran-Minded cont.

CUDA SDK cont.

```
cuda-zen sh@n3073-004:~$ ls

fvcadd.f ntmdtr.cu

cuda-zen sh@n3073-004:~$ gfortran -c fvcadd.f
cuda-zen sh@n3073-004:~$ nvcc -c ntmdtr.cu
cuda-zen sh@n3073-004:~$ nvcc fvcadd.o ntmdtr.o -lcudart -lgfortran
cuda-zen sh@n3073-004:~$ ./a.out

    1  100.000000
    2  100.000000
    3  100.000000
    4  100.000000
    5  100.000000
    6  100.000000
    7  100.000000
    ....
   98  100.000000
   99  100.000000
  100  100.000000
```

Name mangling 2b considered !

C++ like compiler !

→ https://www.olcf.ornl.gov/tutorials/compiling-mixed-gpu-and-cpu-code
→ https://tinyurl.com/cudafordummies/ii/l4/fvcadd.f
→ https://tinyurl.com/cudafordummies/ii/l4/ntmdtr.cu

- Type analogues,
  int $\leftrightarrow$ INTEGER
  float $\leftrightarrow$ REAL
  double $\leftrightarrow$ DOUBLE PRECISION
- Name mangling: compiling Fortran code will alter names of called subroutines/function by adding a trailing underscore, _ Therefore C-functions will need to be aware of this and add this underscore ahead of time
- Fortran uses column-major order for storing multidimensional arrays while C uses row-major order
- Linking may need the addition of -lcudart -lgfortran etc

→ http://www.computationalmathematics.org/topics/files/calling_cuda_from_fortran.html

CUBLAS

CUDA SDK cont.

- BLAS/LAPACK are among the most widely used libraries in scientific computing

→ http://www.netlib.org/blas/

CUBLAS

CUDA SDK cont.

- BLAS/LAPACK are among the most widely used libraries in scientific computing
- General tasks in linear algebra with systematic naming scheme and a focus on high performance computing

→ http://www.netlib.org/blas/

CUBLAS

CUDA SDK cont.

- BLAS/LAPACK are among the most widely used libraries in scientific computing
- General tasks in linear algebra with systematic naming scheme and a focus on high performance computing
- Standard API for many prominent implementations (MKL, OpenBLAS, ATLAS...)

→ http://www.netlib.org/blas/

CUBLAS

CUDA SDK cont.

- BLAS/LAPACK are among the most widely used libraries in scientific computing
- General tasks in linear algebra with systematic naming scheme and a focus on high performance computing
- Standard API for many prominent implementations (MKL, OpenBLAS, ATLAS...)
- Originally written in Fortran

→ http://www.netlib.org/blas/

CUBLAS cont.

CUDA SDK cont.

S real, single precision

D real, double precision

C complex, single precision

Z complex, double precision

DGEMM(...)

→ http://www.netlib.org/blas/

CUBLAS cont.

CUDA SDK cont.

S real, single precision

D real, double precision

C complex, single precision

Z complex, double precision

DGEMM(...)

→ http://www.netlib.org/blas/

CUBLAS cont.

CUDA SDK cont.

SY symmetric	**HE** Hermitian	**HP** Hermitian packed	**TP** triangular packed
SP symmetric packed	**GB** general band	**HB** Hermitian band	**TB** triangular band
SB symmetric band	**GE** general	**TR** triangular	

S real, single precision

D real, double precision

C complex, single precision

Z complex, double precision

DGEMM(...)

→ http://www.netlib.org/blas/

CUBLAS cont.

CUDA SDK cont.

SY symmetric
SP symmetric packed
SB symmetric band
HE Hermitian
GB general band
GE general
HP Hermitian packed
HB Hermitian band
TR triangular
TP triangular packed
TB triangular band

S real, single precision
D real, double precision
C complex, single precision
Z complex, double precision

DGEMM(...)

MV matrix · vector
SV syst eq. 1 vec unk
R rank-1 matrix update
R2 rank-2 matrix update
MM matrix · matrix
SM syst eq. matrix unk
C conjugate vector
G Givens rotation
MG mod. Givens rot constr

RK rank-k matrix update
R2K rank-2k matrix update
U unconjugate vector
M modified Givens rot

→ http://www.netlib.org/blas/

CUBLAS CONT.

CUDA SDK CONT.

```
...
DGEMM(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
...

...
Atype = Btype = 'T';
Adim = Bdim = Cdim = n;
alpha = (double) 1;
beta = (double) 0;
info = dgemm( &Atype, &Btype, &Adim, &Bdim, &Cdim, &alpha,
              &A[0], &Adim, &B[0], &Bdim, &beta, &C[0], &Cdim );
if ( info != 0 ) {
   printf("blas error in dgemm returning %d\n", info);
   exit(99);
}
...

...
stat = cublasDgemm ( handle, dev_Atype, dev_Btype, Adim, Bdim, Cdim, &alpha,
                     dev_ptr_A, Adim, dev_ptr_B, Bdim, &beta, dev_ptr_C, Cdim );
if ( stat != CUBLAS_STATUS_SUCCESS ) {
   printf("cublas error in dgemm \n");
   exit(99);
}
```

→ http://www.netlib.org/blas/

CUBLAS cont.

CUDA SDK cont.

```
...
DGEMM(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
...
```

gfortran ... -lblas

```
...
Atype = Btype = 'T';
Adim = Bdim = Cdim = n;
alpha = (double) 1;
beta = (double) 0;
info = dgemm( &Atype, &Btype, &Adim, &Bdim, &Cdim, &alpha,
              &A[0], &Adim, &B[0], &Bdim, &beta, &C[0], &Cdim );
if ( info != 0 ) {
   printf("blas error in dgemm returning %d\n", info);
   exit(99);
}
...
```

```
...
stat = cublasDgemm ( handle, dev_Atype, dev_Btype, Adim, Bdim, Cdim, &alpha,
                     dev_ptr_A, Adim, dev_ptr_B, Bdim, &beta, dev_ptr_C, Cdim );
if ( stat != CUBLAS_STATUS_SUCCESS ) {
   printf("cublas error in dgemm \n");
   exit(99);
}
```

→ http://www.netlib.org/blas/

```
...
DGEMM(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
...
```

gfortran ... -lblas

```
...
Atype = Btype = 'T';
Adim = Bdim = Cdim = n;
alpha = (double) 1;
beta = (double) 0;
info = dgemm( &Atype, &Btype, &Adim, &Bdim, &Cdim, &alpha,
              &A[0], &Adim, &B[0], &Bdim, &beta, &C[0], &Cdim );
if ( info != 0 ) {
   printf("blas error in dgemm returning %d\n", info);
   exit(99);
}
...
```

gcc -Ddgemm=dgemm_ ... -lblas

```
...
stat = cublasDgemm ( handle, dev_Atype, dev_Btype, Adim, Bdim, Cdim, &alpha,
                     dev_ptr_A, Adim, dev_ptr_B, Bdim, &beta, dev_ptr_C, Cdim );
if ( stat != CUBLAS_STATUS_SUCCESS ) {
   printf("cublas error in dgemm \n");
   exit(99);
}
```

→ http://www.netlib.org/blas/

CUBLAS cont.

CUDA SDK cont.

```
...
DGEMM(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
...
```

gfortran ... -lblas

```
...
Atype = Btype = 'T';
Adim = Bdim = Cdim = n;
alpha = (double) 1;
beta = (double) 0;
info = dgemm( &Atype, &Btype, &Adim, &Bdim, &Cdim, &alpha,
              &A[0], &Adim, &B[0], &Bdim, &beta, &C[0], &Cdim );
if ( info != 0 ) {
   printf("blas error in dgemm returning %d\n", info);
   exit(99);
}
...
```

gcc -Ddgemm=dgemm_ ... -lblas

```
...
stat = cublasDgemm ( handle, dev_Atype, dev_Btype, Adim, Bdim, Cdim, &alpha,
                     dev_ptr_A, Adim, dev_ptr_B, Bdim, &beta, dev_ptr_C, Cdim );
if ( stat != CUBLAS_STATUS_SUCCESS ) {
   printf("cublas error in dgemm \n");
   exit(99);
}
```

nvcc ... -lcublas

→ http://www.netlib.org/blas/

CUBLAS CONT.

CUDA SDK CONT.

```
...
DGEMM(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
...
```

gfortran ... -lblas

col-wise linearized matrices; ∀pointers

```
...
Atype = Btype = 'T';
Adim = Bdim = Cdim = n;
alpha = (double) 1;
beta = (double) 0;
info = dgemm( &Atype, &Btype, &Adim, &Bdim, &Cdim, &alpha,
              &A[0], &Adim, &B[0], &Bdim, &beta, &C[0], &Cdim );
if ( info != 0 ) {
    printf("blas error in dgemm returning %d\n", info);
    exit(99);
}
...
```

gcc -Ddgemm=dgemm_ ... -lblas

```
...
stat = cublasDgemm ( handle, dev_Atype, dev_Btype, Adim, Bdim, Cdim, &alpha,
                     dev_ptr_A, Adim, dev_ptr_B, Bdim, &beta, dev_ptr_C, Cdim );
if ( stat != CUBLAS_STATUS_SUCCESS ) {
    printf("cublas error in dgemm \n");
    exit(99);
}
```

nvcc ... -lcublas

→ http://www.netlib.org/blas/

CUBLAS cont.

CUDA SDK cont.

```
...
DGEMM(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
...
```

gfortran ... -lblas

col-wise linearized matrices; ∀pointers

```
...
Atype = Btype = 'T';
Adim = Bdim = Cdim = n;
alpha = (double) 1;
beta = (double) 0;
info = dgemm( &Atype, &Btype, &Adim, &Bdim, &Cdim, &alpha,
              &A[0], &Adim, &B[0], &Bdim, &beta, &C[0], &Cdim );
if ( info != 0 ) {
    printf("blas error in dgemm returning %d\n", info);
    exit(99);
}
...
```

gcc -Ddgemm=dgemm_ ... -lblas

separate cublas arg/types

```
...
stat = cublasDgemm ( handle, dev_Atype, dev_Btype, Adim, Bdim, Cdim, &alpha,
                     dev_ptr_A, Adim, dev_ptr_B, Bdim, &beta, dev_ptr_C, Cdim );
if ( stat != CUBLAS_STATUS_SUCCESS ) {
    printf("cublas error in dgemm \n");
    exit(99);
}
```

nvcc ... -lcublas

→ http://www.netlib.org/blas/

Porting code to CUBLAS requires a more-or-less standard procedure of the following steps:

1. Initiate the CUBLAS context (cublasCreate())
2. Allocate device memory using cudaMalloc() !
3. Transfer content of host arrays to the device (cublasSetMatrix())
4. Call a specfic CUBLAS routine, e.g. cublasDgemm()
5. Transfer back the result from device memory to host (cublasGetMatrix())
6. Free device memory
7. Destroy CUBLAS context

→ https://docs.nvidia.com/cuda/cublas/index.html
→ https://devtalk.nvidia.com/default/topic/1047981/b/t/post/5318441

Example cublasDgemm()

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>
#include "cublas_v2.h"

int main(int argc, char **argv)
{
   int N, i, j;
   double alpha, beta, *A, *Adev, *B, *Bdev, *C, *Cdev;
   cublasStatus_t stat;
   cublasHandle_t handle;
   cublasOperation_t Atype, Btype;

   // memory allocation and parameter set up
   N = 5;
   alpha = (double) 1;
   beta = (double) 0;
   Atype = CUBLAS_OP_N;
   Btype = CUBLAS_OP_N;
   A = (double *) malloc(N * N * sizeof(double));
   B = (double *) malloc(N * N * sizeof(double));
   C = (double *) malloc(N * N * sizeof(double));

   // fill matrices A[] and B[] with data
```

→ https://docs.nvidia.com/cuda/cublas/index.html

CUBLAS cont.

CUDA SDK cont.

Example cublasDgemm()

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>
#include "cublas_v2.h"

int main(int argc, char **argv)
{
    int N, i, j;
    double alpha, beta, *A, *Adev, *B, *Bdev, *C, *Cdev;
    cublasStatus_t stat;
    cublasHandle_t handle;
    cublasOperation_t Atype, Btype;

    // memory allocation and parameter set up
    N = 5;
    alpha = (double) 1;
    beta = (double) 0;
    Atype = CUBLAS_OP_N;
    Btype = CUBLAS_OP_N;
    A = (double *) malloc(N * N * sizeof(double));
    B = (double *) malloc(N * N * sizeof(double));
    C = (double *) malloc(N * N * sizeof(double));

    // fill matrices A[] and B[] with data
```

go for the new CUBLAS library API

→ https://docs.nvidia.com/cuda/cublas/index.html

Example cublasDgemm()

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>
#include "cublas_v2.h"

int main(int argc, char **argv)
{
    int N, i, j;
    double alpha, beta, *A, *Adev, *B, *Bdev, *C, *Cdev;
    cublasStatus_t stat;
    cublasHandle_t handle;
    cublasOperation_t Atype, Btype;

    // memory allocation and parameter set up
    N = 5;
    alpha = (double) 1;
    beta = (double) 0;
    Atype = CUBLAS_OP_N;
    Btype = CUBLAS_OP_N;
    A = (double *) malloc(N * N * sizeof(double));
    B = (double *) malloc(N * N * sizeof(double));
    C = (double *) malloc(N * N * sizeof(double));

    // fill matrices A[] and B[] with data
```

go for the new CUBLAS library API

will establish CUBLAS context

→ https://docs.nvidia.com/cuda/cublas/index.html

Example cublasDgemm()

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>
#include "cublas_v2.h"

int main(int argc, char **argv)
{
    int N, i, j;
    double alpha, beta, *A, *Adev, *B, *Bdev, *C, *Cdev;
    cublasStatus_t stat;
    cublasHandle_t handle;
    cublasOperation_t Atype, Btype;

    // memory allocation and parameter set up
    N = 5;
    alpha = (double) 1;
    beta = (double) 0;
    Atype = CUBLAS_OP_N;
    Btype = CUBLAS_OP_N;
    A = (double *) malloc(N * N * sizeof(double));
    B = (double *) malloc(N * N * sizeof(double));
    C = (double *) malloc(N * N * sizeof(double));

    // fill matrices A[] and B[] with data
```

go for the new CUBLAS library API

will establish CUBLAS context

CUBLAS way of TRANSA/B='T'∨'N'

→ https://docs.nvidia.com/cuda/cublas/index.html

Example cublasDgemm()

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>
#include "cublas_v2.h"

int main(int argc, char **argv)
{
    int N, i, j;
    double alpha, beta, *A, *Adev, *B, *Bdev, *C, *Cdev;
    cublasStatus_t stat;
    cublasHandle_t handle;
    cublasOperation_t Atype, Btype;

    // memory allocation and parameter set up
    N = 5;
    alpha = (double) 1;
    beta = (double) 0;
    Atype = CUBLAS_OP_N;
    Btype = CUBLAS_OP_N;
    A = (double *) malloc(N * N * sizeof(double));
    B = (double *) malloc(N * N * sizeof(double));
    C = (double *) malloc(N * N * sizeof(double));

    // fill matrices A[] and B[] with data
```

go for the new CUBLAS library API

will establish CUBLAS context

CUBLAS way of TRANSA/B='T'∨'N'

normal memory allocation

→ https://docs.nvidia.com/cuda/cublas/index.html

Example cublasDgemm() cont.

```
    // initiate the CUBLAS context
    cublasCreate(&handle);
    // allocate device memory
    cudaMalloc(&Adev, N * N * sizeof(double));
    cudaMalloc(&Bdev, N * N * sizeof(double));
    cudaMalloc(&Cdev, N * N * sizeof(double));
    // copy contents of arrays A[], B[] into device memory
    cublasSetMatrix(N, N, sizeof(double), &A[0], N, Adev, N);
    cublasSetMatrix(N, N, sizeof(double), &B[0], N, Bdev, N);
    // carry out CUBLAS operation
    cublasDgemm(handle, Atype, Btype, N, N, N, &alpha, Adev, N, Bdev, N, &beta, Cdev, N);
    // retrieve results from device memory
    cublasGetMatrix(N, N, sizeof(double), Cdev, N, &C[0], N);
    // do something with the result
    ... = C[]
    // make clean
    cudaFree(Cdev); cudaFree(Bdev); cudaFree(Adev);
    cublasDestroy(handle);
    // free host memory and return
    free(C); free(B); free(A);
    return(0);
}
```

→ https://docs.nvidia.com/cuda/cublas/index.html

Example cublasDgemm() cont.

```
    // initiate the CUBLAS context
    cublasCreate(&handle);
    // allocate device memory
    cudaMalloc(&Adev, N * N * sizeof(double));
    cudaMalloc(&Bdev, N * N * sizeof(double));
    cudaMalloc(&Cdev, N * N * sizeof(double));
    // copy contents of arrays A[], B[] into device memory
    cublasSetMatrix(N, N, sizeof(double), &A[0], N, Adev, N);
    cublasSetMatrix(N, N, sizeof(double), &B[0], N, Bdev, N);
    // carry out CUBLAS operation
    cublasDgemm(handle, Atype, Btype, N, N, N, &alpha, Adev, N, Bdev, N, &beta, Cdev, N);
    // retrieve results from device memory
    cublasGetMatrix(N, N, sizeof(double), Cdev, N, &C[0], N);
    // do something with the result
    ... = C[]
    // make clean
    cudaFree(Cdev); cudaFree(Bdev); cudaFree(Adev);
    cublasDestroy(handle);
    // free host memory and return
    free(C); free(B); free(A);
    return(0);
}
```

Step 1

→ https://docs.nvidia.com/cuda/cublas/index.html

Example cublasDgemm() cont.

```
    // initiate the CUBLAS context
    cublasCreate(&handle);
    // allocate device memory
    cudaMalloc(&Adev, N * N * sizeof(double));
    cudaMalloc(&Bdev, N * N * sizeof(double));
    cudaMalloc(&Cdev, N * N * sizeof(double));
    // copy contents of arrays A[], B[] into device memory
    cublasSetMatrix(N, N, sizeof(double), &A[0], N, Adev, N);
    cublasSetMatrix(N, N, sizeof(double), &B[0], N, Bdev, N);
    // carry out CUBLAS operation
    cublasDgemm(handle, Atype, Btype, N, N, N, &alpha, Adev, N, Bdev, N, &beta, Cdev, N);
    // retrieve results from device memory
    cublasGetMatrix(N, N, sizeof(double), Cdev, N, &C[0], N);
    // do something with the result
    ... = C[]
    // make clean
    cudaFree(Cdev); cudaFree(Bdev); cudaFree(Adev);
    cublasDestroy(handle);
    // free host memory and return
    free(C); free(B); free(A);
    return(0);
}
```

Step 1

Step 2

→ https://docs.nvidia.com/cuda/cublas/index.html

Example cublasDgemm() cont.

```
    // initiate the CUBLAS context
    cublasCreate(&handle);
    // allocate device memory
    cudaMalloc(&Adev, N * N * sizeof(double));
    cudaMalloc(&Bdev, N * N * sizeof(double));
    cudaMalloc(&Cdev, N * N * sizeof(double));
    // copy contents of arrays A[], B[] into device memory
    cublasSetMatrix(N, N, sizeof(double), &A[0], N, Adev, N);
    cublasSetMatrix(N, N, sizeof(double), &B[0], N, Bdev, N);
    // carry out CUBLAS operation
    cublasDgemm(handle, Atype, Btype, N, N, N, &alpha, Adev, N, Bdev, N, &beta, Cdev, N);
    // retrieve results from device memory
    cublasGetMatrix(N, N, sizeof(double), Cdev, N, &C[0], N);
    // do something with the result
    ... = C[]
    // make clean
    cudaFree(Cdev); cudaFree(Bdev); cudaFree(Adev);
    cublasDestroy(handle);
    // free host memory and return
    free(C); free(B); free(A);
    return(0);
}
```

Step 1

Step 2

Step 3

→ https://docs.nvidia.com/cuda/cublas/index.html

CUBLAS cont.

CUDA SDK cont.

Example cublasDgemm() cont.

```
    // initiate the CUBLAS context
    cublasCreate(&handle);
    // allocate device memory
    cudaMalloc(&Adev, N * N * sizeof(double));
    cudaMalloc(&Bdev, N * N * sizeof(double));
    cudaMalloc(&Cdev, N * N * sizeof(double));
    // copy contents of arrays A[], B[] into device memory
    cublasSetMatrix(N, N, sizeof(double), &A[0], N, Adev, N);
    cublasSetMatrix(N, N, sizeof(double), &B[0], N, Bdev, N);
    // carry out CUBLAS operation
    cublasDgemm(handle, Atype, Btype, N, N, N, &alpha, Adev, N, Bdev, N, &beta, Cdev, N);
    // retrieve results from device memory
    cublasGetMatrix(N, N, sizeof(double), Cdev, N, &C[0], N);
    // do something with the result
    ... = C[]
    // make clean
    cudaFree(Cdev); cudaFree(Bdev); cudaFree(Adev);
    cublasDestroy(handle);
    // free host memory and return
    free(C); free(B); free(A);
    return(0);
}
```

Step 1

Step 2

Step 3

Step 4

→ https://docs.nvidia.com/cuda/cublas/index.html

Example cublasDgemm() cont.

```
    // initiate the CUBLAS context
    cublasCreate(&handle);
    // allocate device memory
    cudaMalloc(&Adev, N * N * sizeof(double));
    cudaMalloc(&Bdev, N * N * sizeof(double));
    cudaMalloc(&Cdev, N * N * sizeof(double));
    // copy contents of arrays A[], B[] into device memory
    cublasSetMatrix(N, N, sizeof(double), &A[0], N, Adev, N);
    cublasSetMatrix(N, N, sizeof(double), &B[0], N, Bdev, N);
    // carry out CUBLAS operation
    cublasDgemm(handle, Atype, Btype, N, N, N, &alpha, Adev, N, Bdev, N, &beta, Cdev, N);
    // retrieve results from device memory
    cublasGetMatrix(N, N, sizeof(double), Cdev, N, &C[0], N);
    // do something with the result
    ... = C[]
    // make clean
    cudaFree(Cdev); cudaFree(Bdev); cudaFree(Adev);
    cublasDestroy(handle);
    // free host memory and return
    free(C); free(B); free(A);
    return(0);
}
```

Step 1

Step 2

Step 3

Step 4

Step 5

→ https://docs.nvidia.com/cuda/cublas/index.html

Example cublasDgemm() cont.

```
    // initiate the CUBLAS context
    cublasCreate(&handle);
    // allocate device memory
    cudaMalloc(&Adev, N * N * sizeof(double));
    cudaMalloc(&Bdev, N * N * sizeof(double));
    cudaMalloc(&Cdev, N * N * sizeof(double));
    // copy contents of arrays A[], B[] into device memory
    cublasSetMatrix(N, N, sizeof(double), &A[0], N, Adev, N);
    cublasSetMatrix(N, N, sizeof(double), &B[0], N, Bdev, N);
    // carry out CUBLAS operation
    cublasDgemm(handle, Atype, Btype, N, N, N, &alpha, Adev, N, Bdev, N, &beta, Cdev, N);
    // retrieve results from device memory
    cublasGetMatrix(N, N, sizeof(double), Cdev, N, &C[0], N);
    // do something with the result
    ... = C[]
    // make clean
    cudaFree(Cdev); cudaFree(Bdev); cudaFree(Adev);
    cublasDestroy(handle);
    // free host memory and return
    free(C); free(B); free(A);
    return(0);
}
```

Step 1

Step 2

Step 3

Step 4

Step 5

Step 6

→ https://docs.nvidia.com/cuda/cublas/index.html

Example cublasDgemm() cont.

```
    // initiate the CUBLAS context
    cublasCreate(&handle);
    // allocate device memory
    cudaMalloc(&Adev, N * N * sizeof(double));
    cudaMalloc(&Bdev, N * N * sizeof(double));
    cudaMalloc(&Cdev, N * N * sizeof(double));
    // copy contents of arrays A[], B[] into device memory
    cublasSetMatrix(N, N, sizeof(double), &A[0], N, Adev, N);
    cublasSetMatrix(N, N, sizeof(double), &B[0], N, Bdev, N);
    // carry out CUBLAS operation
    cublasDgemm(handle, Atype, Btype, N, N, N, &alpha, Adev, N, Bdev, N, &beta, Cdev, N);
    // retrieve results from device memory
    cublasGetMatrix(N, N, sizeof(double), Cdev, N, &C[0], N);
    // do something with the result
    ... = C[]
    // make clean
    cudaFree(Cdev); cudaFree(Bdev); cudaFree(Adev);
    cublasDestroy(handle);
    // free host memory and return
    free(C); free(B); free(A);
    return(0);
}
```

Step 1

Step 2

Step 3

Step 4

Step 5

Step 6

Step 7

→ https://docs.nvidia.com/cuda/cublas/index.html

Words of caution:

- Always start with a small explicit example to check/confirm each individual step
- Formats of matrices are crucial ! 1D-representation, column-wise linearized;
- (double) on consumer grade cards is slower by $\approx 1/64$ than (float)
- Two new APIs, CUBLASXT (for CUDA ≥ 6.0) and cuBLASLt (for CUDA 10.1)
- Pity that we can't use cudaMallocManaged() at this point

→ https://docs.nvidia.com/cuda/cublas/index.html
→ https://devtalk.nvidia.com/default/topic/1047981/b/t/post/5318441

Next level of problems:

$$\underbrace{\begin{pmatrix} 1.96 & -6.49 & -0.47 & -7.20 & -0.65 \\ -6.49 & 3.80 & -6.39 & 1.50 & -6.34 \\ -0.47 & -6.39 & 4.17 & -1.51 & 2.67 \\ -7.20 & 1.50 & -1.51 & 5.70 & 1.80 \\ -0.65 & -6.34 & 2.67 & 1.80 & -7.10 \end{pmatrix}}_{\mathbf{A}}$$

- For example, what are the eigenvalues and corresponding eigenvectors of A ?
- $A\,x = \lambda\,x$
- A typical LAPACK problem — usually tightly coupled to BLAS
- Not on the GPU ! CUBLAS is BLAS only !
- However, there are cuSolver and MAGMA

→ https://docs.nvidia.com/cuda/cusolver
→ https://icl.cs.utk.edu/projectsfiles/magma/doxygen/group__cublas__const.html

Making use of CUSOLVER is schematically very similar to using CUBLAS:

1. Initiate the CUSOLVER context (cusolverDnCreate())
2. Allocate device memory using cudaMalloc() !
3. Transfer content of host arrays to the device (cudaMemcpy())
4. Semi-automatically set up working space required by CUSOLVER routine, e.g. WORK, LWORK in LAPACK jargon
5. Call a specfic CUSOLVER routine, e.g. cusolverDnDsyevd()
6. Transfer back the result from device memory to host (cudaMemcpy())
7. Free device memory
8. Destroy CUSOLVER context

→ https://docs.nvidia.com/cuda/cusolver/index.html#eig_examples

Example cusolverDnDsyevd()

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

int main(int argc, char **argv)
{
    int i, j, lwork, info_gpu, *devInfo;
    double *d_A, *d_W, *d_work;
    cusolverDnHandle_t cusolverH;
    cusolverStatus_t cusolver_status;
    cusolverEigMode_t jobz;
    cublasFillMode_t uplo;
    cudaError_t cudaStat;
    const int m = 5;
    const int lda = m;
    double W[m];
    double V[lda*m];
    double A[lda*m] = { 1.96, -6.49, -0.47, -7.20, -0.65,
                        -6.49, 3.80, -6.39, 1.50, -6.34,
                        -0.47, -6.39, 4.17, -1.51, 2.67,
                        -7.20, 1.50, -1.51, 5.70, 1.80,
                        -0.65, -6.34, 2.67, 1.80, -7.10};
```

Example cusolverDnDsyevd()

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

int main(int argc, char **argv)
{
    int i, j, lwork, info_gpu, *devInfo;
    double *d_A, *d_W, *d_work;
    cusolverDnHandle_t cusolverH;
    cusolverStatus_t cusolver_status;
    cusolverEigMode_t jobz;
    cublasFillMode_t uplo;
    cudaError_t cudaStat;
    const int m = 5;
    const int lda = m;
    double W[m];
    double V[lda*m];
    double A[lda*m] = { 1.96, -6.49, -0.47, -7.20, -0.65,
                       -6.49, 3.80, -6.39, 1.50, -6.34,
                       -0.47, -6.39, 4.17, -1.51, 2.67,
                       -7.20, 1.50, -1.51, 5.70, 1.80,
                       -0.65, -6.34, 2.67, 1.80, -7.10};
```

CUSOLVER header for 'dense' subset

Example cusolverDnDsyevd()

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

int main(int argc, char **argv)
{
    int i, j, lwork, info_gpu, *devInfo;
    double *d_A, *d_W, *d_work;
    cusolverDnHandle_t cusolverH;
    cusolverStatus_t cusolver_status;
    cusolverEigMode_t jobz;
    cublasFillMode_t uplo;
    cudaError_t cudaStat;
    const int m = 5;
    const int lda = m;
    double W[m];
    double V[lda*m];
    double A[lda*m] = { 1.96, -6.49, -0.47, -7.20, -0.65,
                        -6.49, 3.80, -6.39, 1.50, -6.34,
                        -0.47, -6.39, 4.17, -1.51, 2.67,
                        -7.20, 1.50, -1.51, 5.70, 1.80,
                        -0.65, -6.34, 2.67, 1.80, -7.10};
```

CUSOLVER header for 'dense' subset

4 CUSOLVER context

Example cusolverDnDsyevd()

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

int main(int argc, char **argv)
{
    int i, j, lwork, info_gpu, *devInfo;
    double *d_A, *d_W, *d_work;
    cusolverDnHandle_t cusolverH;
    cusolverStatus_t cusolver_status;
    cusolverEigMode_t jobz;
    cublasFillMode_t uplo;
    cudaError_t cudaStat;
    const int m = 5;
    const int lda = m;
    double W[m];
    double V[lda*m];
    double A[lda*m] = { 1.96, -6.49, -0.47, -7.20, -0.65,
                       -6.49, 3.80, -6.39, 1.50, -6.34,
                       -0.47, -6.39, 4.17, -1.51, 2.67,
                       -7.20, 1.50, -1.51, 5.70, 1.80,
                       -0.65, -6.34, 2.67, 1.80, -7.10};
```

CUSOLVER header for 'dense' subset

4 CUSOLVER context

CUSOLVER flags 4 LAPACK's

ASC Austrian Scientific Computing

Example cusolverDnDsyevd() cont.

```
    jobz = CUSOLVER_EIG_MODE_VECTOR;
    uplo = CUBLAS_FILL_MODE_LOWER;
    cusolver_status = cusolverDnCreate(&cusolverH);
    assert(CUSOLVER_STATUS_SUCCESS == cusolver_status);
    cudaStat = cudaMalloc ((void**)&d_A, sizeof(double) * lda * m);
    cudaStat = cudaMalloc ((void**)&d_W, sizeof(double) * m);
    cudaStat = cudaMalloc ((void**)&devInfo, sizeof(int));
    cudaStat = cudaMemcpy(d_A, A, sizeof(double) * lda * m, cudaMemcpyHostToDevice);

    cusolver_status = cusolverDnDsyevd_bufferSize(cusolverH, jobz, uplo, m, d_A, lda, d_W, &lwork);
    cudaStat = cudaMalloc((void**)&d_work, sizeof(double)*lwork);
    cusolver_status = cusolverDnDsyevd(cusolverH, jobz, uplo, m, d_A, lda, d_W, d_work, lwork, devInfo);
    cudaStat = cudaDeviceSynchronize();

    cudaStat = cudaMemcpy(W, d_W, sizeof(double)*m, cudaMemcpyDeviceToHost);
    cudaStat = cudaMemcpy(V, d_A, sizeof(double)*lda*m, cudaMemcpyDeviceToHost);
    cudaStat = cudaMemcpy(&info_gpu, devInfo, sizeof(int), cudaMemcpyDeviceToHost);
    ... do something with W[] and V[] ...
    cudaFree(d_A); cudaFree(d_W); cudaFree(devInfo); cudaFree(d_work);
    cusolverDnDestroy(cusolverH);
    cudaDeviceReset();

    return(0);
}
```

Example cusolverDnDsyevd() cont.

establish CUSOLVER context

```
    jobz = CUSOLVER_EIG_MODE_VECTOR;
    uplo = CUBLAS_FILL_MODE_LOWER;
    cusolver_status = cusolverDnCreate(&cusolverH);
    assert(CUSOLVER_STATUS_SUCCESS == cusolver_status);
    cudaStat = cudaMalloc ((void**)&d_A, sizeof(double) * lda * m);
    cudaStat = cudaMalloc ((void**)&d_W, sizeof(double) * m);
    cudaStat = cudaMalloc ((void**)&devInfo, sizeof(int));
    cudaStat = cudaMemcpy(d_A, A, sizeof(double) * lda * m, cudaMemcpyHostToDevice);

    cusolver_status = cusolverDnDsyevd_bufferSize(cusolverH, jobz, uplo, m, d_A, lda, d_W, &lwork);
    cudaStat = cudaMalloc((void**)&d_work, sizeof(double)*lwork);
    cusolver_status = cusolverDnDsyevd(cusolverH, jobz, uplo, m, d_A, lda, d_W, d_work, lwork, devInfo);
    cudaStat = cudaDeviceSynchronize();

    cudaStat = cudaMemcpy(W, d_W, sizeof(double)*m, cudaMemcpyDeviceToHost);
    cudaStat = cudaMemcpy(V, d_A, sizeof(double)*lda*m, cudaMemcpyDeviceToHost);
    cudaStat = cudaMemcpy(&info_gpu, devInfo, sizeof(int), cudaMemcpyDeviceToHost);
    ... do something with W[] and V[] ...
    cudaFree(d_A); cudaFree(d_W); cudaFree(devInfo); cudaFree(d_work);
    cusolverDnDestroy(cusolverH);
    cudaDeviceReset();

    return(0);
}
```

Example cusolverDnDsyevd() cont.

```
    jobz = CUSOLVER_EIG_MODE_VECTOR;
    uplo = CUBLAS_FILL_MODE_LOWER;
    cusolver_status = cusolverDnCreate(&cusolverH);
    assert(CUSOLVER_STATUS_SUCCESS == cusolver_status);
    cudaStat = cudaMalloc ((void**)&d_A, sizeof(double) * lda * m);
    cudaStat = cudaMalloc ((void**)&d_W, sizeof(double) * m);
    cudaStat = cudaMalloc ((void**)&devInfo, sizeof(int));
    cudaStat = cudaMemcpy(d_A, A, sizeof(double) * lda * m, cudaMemcpyHostToDevice);

    cusolver_status = cusolverDnDsyevd_bufferSize(cusolverH, jobz, uplo, m, d_A, lda, d_W, &lwork);
    cudaStat = cudaMalloc((void**)&d_work, sizeof(double)*lwork);
    cusolver_status = cusolverDnDsyevd(cusolverH, jobz, uplo, m, d_A, lda, d_W, d_work, lwork, devInfo);
    cudaStat = cudaDeviceSynchronize();

    cudaStat = cudaMemcpy(W, d_W, sizeof(double)*m, cudaMemcpyDeviceToHost);
    cudaStat = cudaMemcpy(V, d_A, sizeof(double)*lda*m, cudaMemcpyDeviceToHost);
    cudaStat = cudaMemcpy(&info_gpu, devInfo, sizeof(int), cudaMemcpyDeviceToHost);
    ... do something with W[] and V[] ...
    cudaFree(d_A); cudaFree(d_W); cudaFree(devInfo); cudaFree(d_work);
    cusolverDnDestroy(cusolverH);
    cudaDeviceReset();

    return(0);
}
```

establish CUSOLVER context

error check

Example cusolverDnDsyevd() cont.

```
    jobz = CUSOLVER_EIG_MODE_VECTOR;
    uplo = CUBLAS_FILL_MODE_LOWER;
    cusolver_status = cusolverDnCreate(&cusolverH);
    assert(CUSOLVER_STATUS_SUCCESS == cusolver_status);
    cudaStat = cudaMalloc ((void**)&d_A, sizeof(double) * lda * m);
    cudaStat = cudaMalloc ((void**)&d_W, sizeof(double) * m);
    cudaStat = cudaMalloc ((void**)&devInfo, sizeof(int));
    cudaStat = cudaMemcpy(d_A, A, sizeof(double) * lda * m, cudaMemcpyHostToDevice);

    cusolver_status = cusolverDnDsyevd_bufferSize(cusolverH, jobz, uplo, m, d_A, lda, d_W, &lwork);
    cudaStat = cudaMalloc((void**)&d_work, sizeof(double)*lwork);
    cusolver_status = cusolverDnDsyevd(cusolverH, jobz, uplo, m, d_A, lda, d_W, d_work, lwork, devInfo);
    cudaStat = cudaDeviceSynchronize();

    cudaStat = cudaMemcpy(W, d_W, sizeof(double)*m, cudaMemcpyDeviceToHost);
    cudaStat = cudaMemcpy(V, d_A, sizeof(double)*lda*m, cudaMemcpyDeviceToHost);
    cudaStat = cudaMemcpy(&info_gpu, devInfo, sizeof(int), cudaMemcpyDeviceToHost);
    ... do something with W[] and V[] ...
    cudaFree(d_A); cudaFree(d_W); cudaFree(devInfo); cudaFree(d_work);
    cusolverDnDestroy(cusolverH);
    cudaDeviceReset();

    return(0);
}
```

establish CUSOLVER context

error check

standard procedure

Example cusolverDnDsyevd() cont.

adjust work arrays

establish CUSOLVER context

error check

standard procedure

```
jobz = CUSOLVER_EIG_MODE_VECTOR;
uplo = CUBLAS_FILL_MODE_LOWER;
cusolver_status = cusolverDnCreate(&cusolverH);
assert(CUSOLVER_STATUS_SUCCESS == cusolver_status);
cudaStat = cudaMalloc ((void**)&d_A, sizeof(double) * lda * m);
cudaStat = cudaMalloc ((void**)&d_W, sizeof(double) * m);
cudaStat = cudaMalloc ((void**)&devInfo, sizeof(int));
cudaStat = cudaMemcpy(d_A, A, sizeof(double) * lda * m, cudaMemcpyHostToDevice);

cusolver_status = cusolverDnDsyevd_bufferSize(cusolverH, jobz, uplo, m, d_A, lda, d_W, &lwork);
cudaStat = cudaMalloc((void**)&d_work, sizeof(double)*lwork);
cusolver_status = cusolverDnDsyevd(cusolverH, jobz, uplo, m, d_A, lda, d_W, d_work, lwork, devInfo);
cudaStat = cudaDeviceSynchronize();

cudaStat = cudaMemcpy(W, d_W, sizeof(double)*m, cudaMemcpyDeviceToHost);
cudaStat = cudaMemcpy(V, d_A, sizeof(double)*lda*m, cudaMemcpyDeviceToHost);
cudaStat = cudaMemcpy(&info_gpu, devInfo, sizeof(int), cudaMemcpyDeviceToHost);
... do something with W[] and V[] ...
cudaFree(d_A); cudaFree(d_W); cudaFree(devInfo); cudaFree(d_work);
cusolverDnDestroy(cusolverH);
cudaDeviceReset();

return(0);
}
```

Example cusolverDnDsyevd() cont.

establish CUSOLVER context

error check

standard procedure

adjust work arrays

call solver

```
jobz = CUSOLVER_EIG_MODE_VECTOR;
uplo = CUBLAS_FILL_MODE_LOWER;
cusolver_status = cusolverDnCreate(&cusolverH);
assert(CUSOLVER_STATUS_SUCCESS == cusolver_status);
cudaStat = cudaMalloc ((void**)&d_A, sizeof(double) * lda * m);
cudaStat = cudaMalloc ((void**)&d_W, sizeof(double) * m);
cudaStat = cudaMalloc ((void**)&devInfo, sizeof(int));
cudaStat = cudaMemcpy(d_A, A, sizeof(double) * lda * m, cudaMemcpyHostToDevice);

cusolver_status = cusolverDnDsyevd_bufferSize(cusolverH, jobz, uplo, m, d_A, lda, d_W, &lwork);
cudaStat = cudaMalloc((void**)&d_work, sizeof(double)*lwork);
cusolver_status = cusolverDnDsyevd(cusolverH, jobz, uplo, m, d_A, lda, d_W, d_work, lwork, devInfo);
cudaStat = cudaDeviceSynchronize();

cudaStat = cudaMemcpy(W, d_W, sizeof(double)*m, cudaMemcpyDeviceToHost);
cudaStat = cudaMemcpy(V, d_A, sizeof(double)*lda*m, cudaMemcpyDeviceToHost);
cudaStat = cudaMemcpy(&info_gpu, devInfo, sizeof(int), cudaMemcpyDeviceToHost);
... do something with W[] and V[] ...
cudaFree(d_A); cudaFree(d_W); cudaFree(devInfo); cudaFree(d_work);
cusolverDnDestroy(cusolverH);
cudaDeviceReset();

return(0);
}
```

Example cusolverDnDsyevd() cont.

adjust work arrays

establish CUSOLVER context

error check

standard procedure

```
jobz = CUSOLVER_EIG_MODE_VECTOR;
uplo = CUBLAS_FILL_MODE_LOWER;
cusolver_status = cusolverDnCreate(&cusolverH);
assert(CUSOLVER_STATUS_SUCCESS == cusolver_status);
cudaStat = cudaMalloc ((void**)&d_A, sizeof(double) * lda * m);
cudaStat = cudaMalloc ((void**)&d_W, sizeof(double) * m);
cudaStat = cudaMalloc ((void**)&devInfo, sizeof(int));
cudaStat = cudaMemcpy(d_A, A, sizeof(double) * lda * m, cudaMemcpyHostToDevice);

cusolver_status = cusolverDnDsyevd_bufferSize(cusolverH, jobz, uplo, m, d_A, lda, d_W, &lwork);
cudaStat = cudaMalloc((void**)&d_work, sizeof(double)*lwork);
cusolver_status = cusolverDnDsyevd(cusolverH, jobz, uplo, m, d_A, lda, d_W, d_work, lwork, devInfo);
cudaStat = cudaDeviceSynchronize();

cudaStat = cudaMemcpy(W, d_W, sizeof(double)*m, cudaMemcpyDeviceToHost);
cudaStat = cudaMemcpy(V, d_A, sizeof(double)*lda*m, cudaMemcpyDeviceToHost);
cudaStat = cudaMemcpy(&info_gpu, devInfo, sizeof(int), cudaMemcpyDeviceToHost);
... do something with W[] and V[] ...
cudaFree(d_A); cudaFree(d_W); cudaFree(devInfo); cudaFree(d_work);
cusolverDnDestroy(cusolverH);
cudaDeviceReset();

return(0);
}
```

call solver

back copy results

Example cusolverDnDsyevd() cont.

```
  jobz = CUSOLVER_EIG_MODE_VECTOR;
  uplo = CUBLAS_FILL_MODE_LOWER;
  cusolver_status = cusolverDnCreate(&cusolverH);
  assert(CUSOLVER_STATUS_SUCCESS == cusolver_status);
  cudaStat = cudaMalloc ((void**)&d_A, sizeof(double) * lda * m);
  cudaStat = cudaMalloc ((void**)&d_W, sizeof(double) * m);
  cudaStat = cudaMalloc ((void**)&devInfo, sizeof(int));
  cudaStat = cudaMemcpy(d_A, A, sizeof(double) * lda * m, cudaMemcpyHostToDevice);

  cusolver_status = cusolverDnDsyevd_bufferSize(cusolverH, jobz, uplo, m, d_A, lda, d_W, &lwork);
  cudaStat = cudaMalloc((void**)&d_work, sizeof(double)*lwork);
  cusolver_status = cusolverDnDsyevd(cusolverH, jobz, uplo, m, d_A, lda, d_W, d_work, lwork, devInfo);
  cudaStat = cudaDeviceSynchronize();

  cudaStat = cudaMemcpy(W, d_W, sizeof(double)*m, cudaMemcpyDeviceToHost);
  cudaStat = cudaMemcpy(V, d_A, sizeof(double)*lda*m, cudaMemcpyDeviceToHost);
  cudaStat = cudaMemcpy(&info_gpu, devInfo, sizeof(int), cudaMemcpyDeviceToHost);
  ... do something with W[] and V[] ...
  cudaFree(d_A); cudaFree(d_W); cudaFree(devInfo); cudaFree(d_work);
  cusolverDnDestroy(cusolverH);
  cudaDeviceReset();

  return(0);
}
```

adjust work arrays

establish CUSOLVER context

error check

standard procedure

call solver

back copy results

finalize

- Very straightforward procedure offering significant speed-up for large scale problems at very minor porting effort
- Again, (float) $\leftrightarrow$ (double) gap in terms of performance, $\approx 1/64$ on consumer cards !
- A related problem of considerable interest may be addressed via cusolverEigType_t — generalized symmetric-definite eigenvalue problem (section 2.2.1.4) $A\,x = \lambda\,B\,x$
- Sparse problems covered too — cuSolverSP

→ https://docs.nvidia.com/cuda/cusolver/index.html

More Libraries

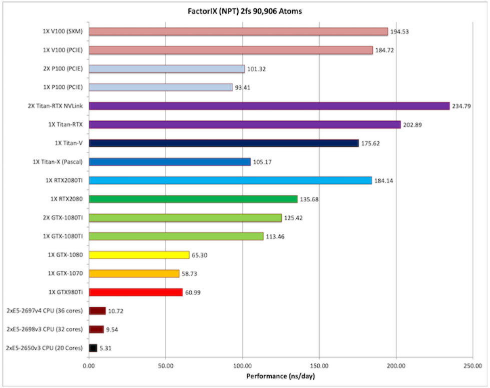
CUDA SDK cont.

- CUFFT — CUDA Fast Fourier Transform
- CURAND — CUDA random number generation library
- CUDNN & TENSORRT — Deep Learning, training & inference
- CUSPARSE — Basic linear algebra for sparse matrices
- NVBLAS — Another replacement for BLAS calls with little effort of porting (re-linking or LD_PRELOAD)
- NVGRAPH — Big Data analytics via graph problems

→ https://docs.nvidia.com/cuda/cufft/index.html
→ https://docs.nvidia.com/cuda/curand/index.html
→ https://docs.nvidia.com/deeplearning/sdk/index.html

CUDA SDK cont.

How come that scientific apps, e.g. AMBER's pmemd.cuda are doing so well on consumer grade GPUs ?



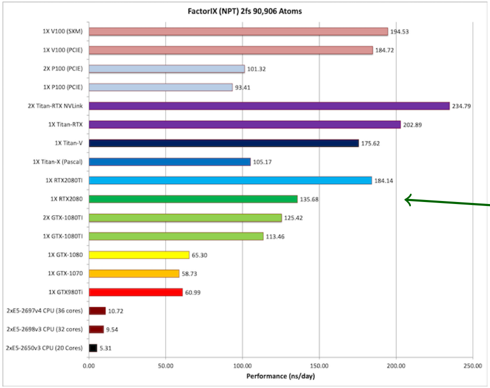
→ http://ambermd.org/GPUPerformance.php
→ https://www.sciencedirect.com/science/article/pii/S0010465512003098

ASC Austrian Scientific Computing

Numerical Accuracy & Performance

CUDA SDK cont.

How come that scientific apps, e.g. AMBER's pmemd.cuda are doing so well on consumer grade GPUs ?



Combine several 32bit types (float, int) to approximate double

→ http://ambermd.org/GPUPerformance.php

→ https://www.sciencedirect.com/science/article/pii/S0010465512003098

Dekker (1971) and Knuth (1969)

Quasi-double precision method

- Combine two floating-point numbers to express one variable

x + y = z + zz	x = 10.2, y = 0.345
(where | z / zz | < 2^-24)	(three significant digits)
z = fl(x + y)	z = 10.5
w = fl(z – x)	w = 0.3
zz = fl(y – w)	zz = 0.045
	(z + zz = 10.545, no cancellation)

x, y, z, zz : FP32 variable

fl(operation) : FP32 arithmetic operation

- More than twice operations are needed for similar accuracy to double-precision

$\Rightarrow$ Only works for consumer GPUs

(slide courtesy of Prof Tetsu Narumi, UEC, Tokyo)

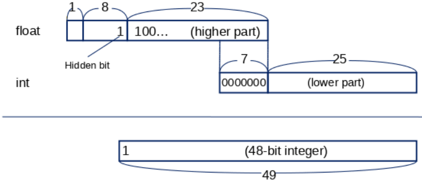
→ https://link.springer.com/article/10.1007/BF01397083
→ https://en.wikipedia.org/wiki/The_Art_of_Computer_Programming

Numerical Accuracy & Performance cont.

CUDA SDK cont.

Combine floating- and fixed-point

- FP32 and INT32 for one variable
- Initialized with a large number beforehand
- Similar to 48-bit integer



Tetsu Narumi et al., International Journal of Computational Methods, vol. 8, No. 3, pp. 561-581 (2011)

8

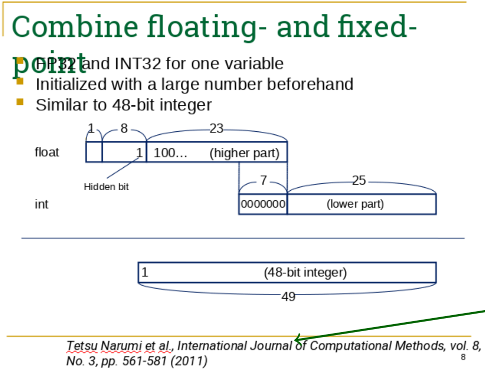
(slide courtesy of Prof Tetsu Narumi, UEC, Tokyo)

→ doi:10.1142/S0219876211002708
→ doi:10.1016/j.cpc.2012.09.022

ASC Austrian Scientific Computing

Numerical Accuracy & Performance cont.

CUDA SDK cont.



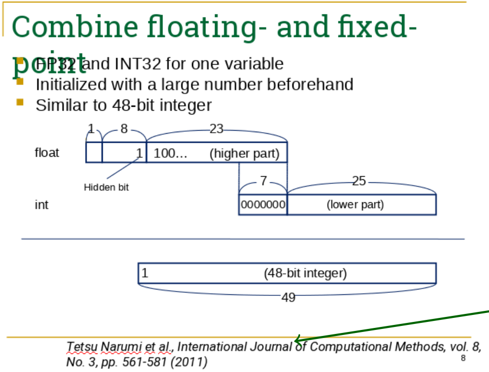
All elementary operations, +, -, *

(slide courtesy of Prof Tetsu Narumi, UEC, Tokyo)

→ doi:10.1142/S0219876211002708
→ doi:10.1016/j.cpc.2012.09.022

Numerical Accuracy & Performance cont.

CUDA SDK cont.



(slide courtesy of Prof Tetsu Narumi, UEC, Tokyo)

Similar strategy in AMBER's pmemd.cuda

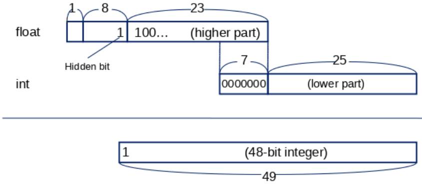
All elementary operations, +, -, *

→ doi:10.1142/S0219876211002708

→ doi:10.1016/j.cpc.2012.09.022

Combine floating- and fixed-point

- FP32 and INT32 for one variable
- Initialized with a large number beforehand
- Similar to 48-bit integer



Tetsu Narumi et al., International Journal of Computational Methods, vol. 8, No. 3, pp. 561-581 (2011) 8

Also see Mandelbrot example in SDK, 5_Domain_Specific

All elementary operations, +, -, *

Similar strategy in AMBER's pmemd.cuda

(slide courtesy of Prof Tetsu Narumi, UEC, Tokyo)

→ doi:10.1142/S0219876211002708
→ doi:10.1016/j.cpc.2012.09.022

5_Domain_Specific/marchingCubes

CUDA SDK cont.

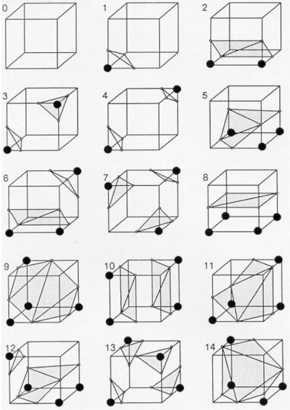
Practical example, of frequent use in biophysical chemistry

- Extracts an isosurface from a volumetric dataset using the 'marching cubes algorithm'
- OpenGL interoperation
- Chosen example is the electron density of C_{60}

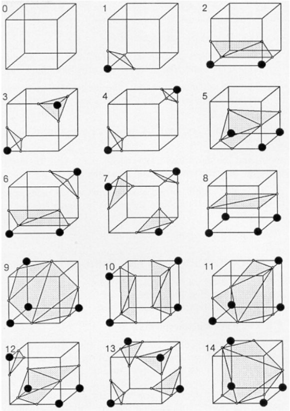
→ `./run_marchingCubes.sh`

5_Domain_Specific/marchingCubes cont.

CUDA SDK cont.



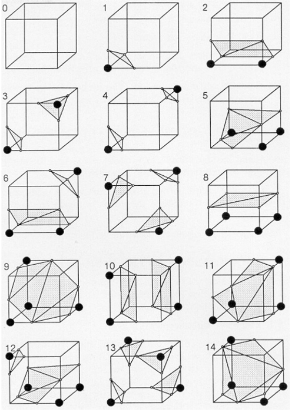
→ https://dl.acm.org/citation.cfm?doid=37402.37422



Assign vertices 0 if below iso-value, 1 if above

→ https://dl.acm.org/citation.cfm?doid=37402.37422

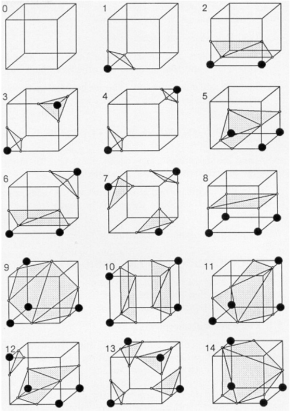
Neighbouring vertices of 0, 1 assignment determine iso-surface edges



Assign vertices 0 if below iso-value, 1 if above

→ https://dl.acm.org/citation.cfm?doid=37402.37422

Neighbouring vertices of 0, 1 assignment determine iso-surface edges



14 elementary cases, the rest (256) from symmetry

Assign vertices 0 if below iso-value, 1 if above

→ https://dl.acm.org/citation.cfm?doid=37402.37422

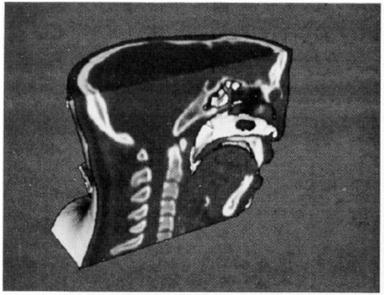


Figure 11. Sagittal Cut with Texture Mapping.

CT data, shows the slice data in relation to the constructed

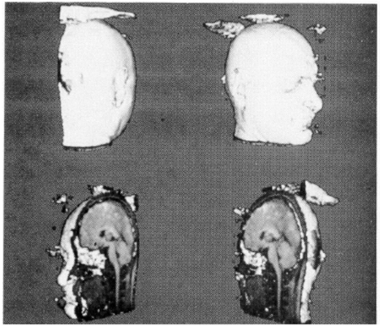


Figure 12. Rotated Sequence of Cut MR Brain.

Interesting for medical images (CT,MR)

→ https://dl.acm.org/citation.cfm?doid=37402.37422

The main profiling tool in CUDA nowadays is NVIDIA Nsight Compute

- nsys nvprof (quick and easy)
- nsys/nsys-ui (for system traces and timeline analysis – streams)
- ncu/ncu-ui (for detailed kernel optimization)

→ https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html

The main profiling tool in CUDA nowadays is NVIDIA Nsight Compute

- nsys nvprof (quick and easy)
- nsys/nsys-ui (for system traces and timeline analysis – streams)
- ncu/ncu-ui (for detailed kernel optimization)

```
cuda—zen sh@n3073—004:~$  nvcc ./mmm_example_1.cu
cuda—zen sh@n3073—004:~$  ncu -f --set full -o profile ./a.out
cuda—zen sh@gui3068—009:~$  ncu-ui ./profile.ncu-rep
```

→ https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html

The main profiling tool in CUDA nowadays is NVIDIA Nsight Compute

- nsys nvprof (quick and easy)
- nsys/nsys-ui (for system traces and timeline analysis – streams)
- ncu/ncu-ui (for detailed kernel optimization)

1. Compilation

```
cuda—zen sh@n3073—004:~$  nvcc ./mmm_example_1.cu
cuda—zen sh@n3073—004:~$  ncu -f –set full -o profile ./a.out
cuda—zen sh@gui3068—009:~$  ncu-ui ./profile.ncu-rep
```

→ https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html

The main profiling tool in CUDA nowadays is NVIDIA Nsight Compute

- nsys nvprof (quick and easy)
- nsys/nsys-ui (for system traces and timeline analysis – streams)
- ncu/ncu-ui (for detailed kernel optimization)

1. Compilation

```
cuda-zen sh@n3073-004:~$  nvcc ./mmm_example_1.cu
cuda-zen sh@n3073-004:~$  ncu -f --set full -o profile ./a.out
cuda-zen sh@gui3068-009:~$  ncu-ui ./profile.ncu-rep
```

2. Write profile

→ https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html

Profiling CUDA Code
CUDA SDK cont.

The main profiling tool in CUDA nowadays is NVIDIA Nsight Compute

- nsys nvprof (quick and easy)
- nsys/nsys-ui (for system traces and timeline analysis – streams)
- ncu/ncu-ui (for detailed kernel optimization)

1. Compilation

```
cuda-zen sh@n3073-004:~$  nvcc ./mmm_example_1.cu
cuda-zen sh@n3073-004:~$ ncu -f --set full -o profile ./a.out
cuda-zen sh@gui3068-009:~$  ncu-ui ./profile.ncu-rep
```

2. Write profile

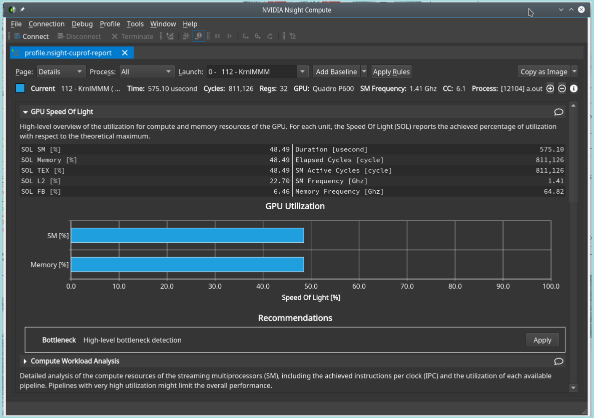
3. Visualize profile

→ https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html

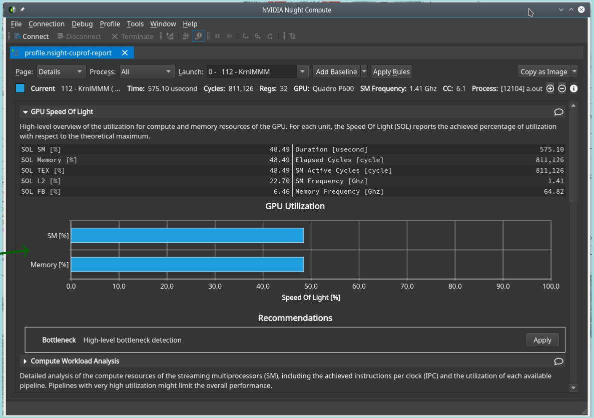
Profiling CUDA Code cont.

CUDA SDK cont.

```
cuda-zen sh@gui3068-009:~$ ncu-ui ./profile.ncu-rep
```

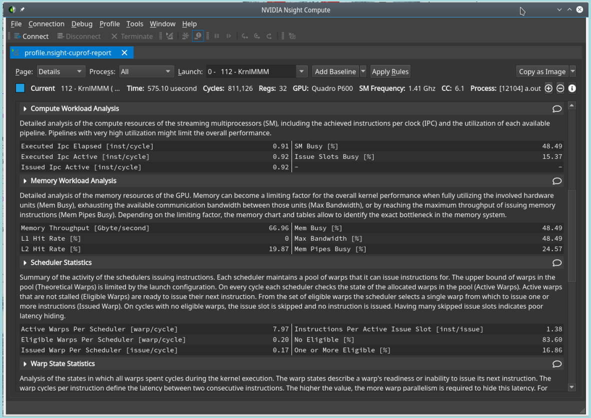


cuda–zen sh@gui3068–009:~$ ncu-ui ./profile.ncu-rep

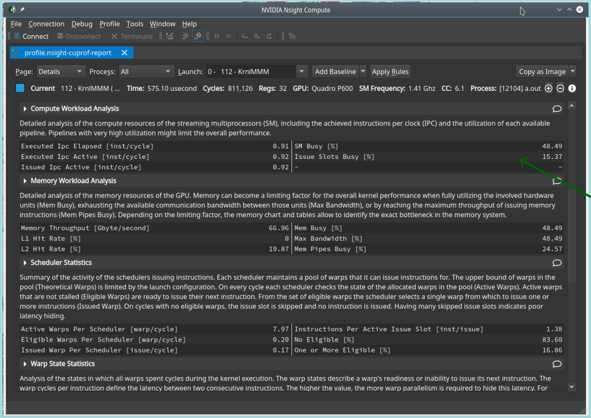


GPU features evaluated w.r.2 theoretical max (SOL)

cuda–zen sh@gui3068–009:~$ ncu-ui ./profile.ncu-rep

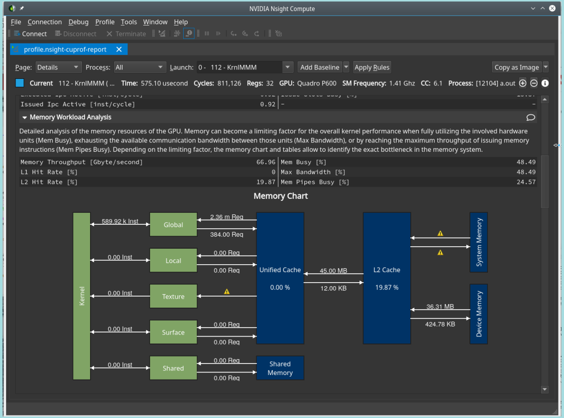


cuda–zen sh@gui3068–009:~$ ncu-ui ./profile.ncu-rep



Different categories' evaluation

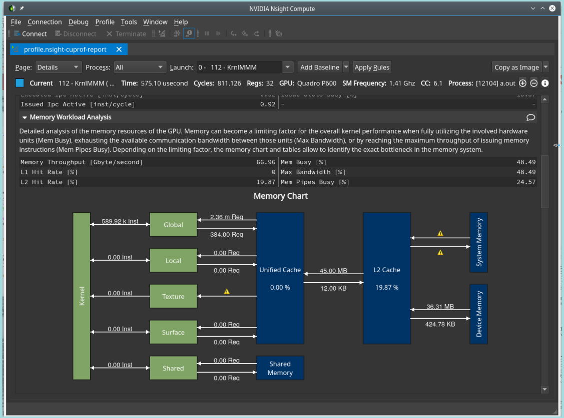
```
cuda-zen sh@gui3068-009:~$ ncu-ui ./profile.ncu-rep
```



Profiling CUDA Code cont.

CUDA SDK cont.

cuda–zen sh@gui3068–009:~$ ncu-ui ./profile.ncu-rep



Expandable detail sections

HIP

- The Heterogeneous Interface for Portability (HIP) is AMD's dedicated GPU programming environment (e.g. to program MI250X/MI300X devices)
- Very similar to CUDA
- HIP code will also run on NVIDIA platforms
- HIP forms another ecosystem with tools, libraries, etc
- Every basic CUDA construct has a direct counterpart in HIP

→ https://developer.amd.com/wp-content/resources/ROCm%20Learning%20Centre/chapter3/HIP-Coding-3.pdf

Vector Addition with HIP

```
// HIP kernel. Each thread takes care of one element of c;
__global__ void vecAdd(double *a, double *b, double *c, int n)
{
    // Get our global thread ID
    int id = (blockIdx.x * blockDim.x) + threadIdx.x;
    // Make sure we do not go beyond bounds
    if (id < n)
        c[id] = a[id] + b[id];
}

int main()
{
    ...
    // Execute the kernel with n threads
    hipLaunchKernelGGL(vecAdd, dim3(gridSize), dim3(blockSize), 0, 0, d_a, d_b, d_c, n);
    hipDeviceSynchronize();
}
```

→ https://developer.amd.com/wp-content/resources/ROCm%20Learning%20Centre/chapter3/HIP-Coding-3.pdf

HIP CONT.

HIP Example

same kernel declaration specifiers

Vector Addition with HIP

```
// HIP kernel. Each thread takes care of one element of c;
__global__ void vecAdd(double *a, double *b, double *c, int n)
{
    // Get our global thread ID
    int id = (blockIdx.x * blockDim.x) + threadIdx.x;
    // Make sure we do not go beyond bounds
    if (id < n)
        c[id] = a[id] + b[id];
}

int main()
{
    ...
    // Execute the kernel with n threads
    hipLaunchKernelGGL(vecAdd, dim3(gridSize), dim3(blockSize), 0, 0, d_a, d_b, d_c, n);
    hipDeviceSynchronize();
}
```

→ https://developer.amd.com/wp-content/resources/ROCm%20Learning%20Centre/chapter3/HIP-Coding-3.pdf

Vector Addition with HIP

```
// HIP kernel. Each thread takes care of one element of c;
__global__ void vecAdd(double *a, double *b, double *c, int n)
{
    // Get our global thread ID
    int id = (blockIdx.x * blockDim.x) + threadIdx.x
    // Make sure we do not go beyond bounds
    if (id < n)
        c[id] = a[id] + b[id];
}

int main()
{
    ...
    // Execute the kernel with n threads
    hipLaunchKernelGGL(vecAdd, dim3(gridSize), dim3(blockSize), 0, 0, d_a, d_b, d_c, n);
    hipDeviceSynchronize();
}
```

same kernel declaration specifiers

same built-in variables, e.g. threadIdx.x=0,1,2...

→ https://developer.amd.com/wp-content/resources/ROCm%20Learning%20Centre/chapter3/HIP-Coding-3.pdf

Vector Addition with HIP

```
// HIP kernel. Each thread takes care of one element of c;
__global__ void vecAdd(double *a, double *b, double *c, int n)
{
    // Get our global thread ID
    int id = (blockIdx.x * blockDim.x) + threadIdx.x;
    // Make sure we do not go beyond bounds
    if (id < n)
        c[id] = a[id] + b[id];
}

int main()
{
    ...
    // Execute the kernel with n threads
    hipLaunchKernelGGL(vecAdd, dim3(gridSize), dim3(blockSize), 0, 0, d_a, d_b, d_c, n);
    hipDeviceSynchronize();
}
```

same kernel declaration specifiers

same built-in variables, e.g. threadIdx.x=0,1,2...

different kernel execution configuration

→ https://developer.amd.com/wp-content/resources/ROCm%20Learning%20Centre/chapter3/HIP-Coding-3.pdf

HIP cont.

Compile and Run and Monitor

```
(zen3) [sh@some – mi250x ~]$ hipcc vadd_hip.cpp
(zen3) [sh@some – mi250x ~]$ ./a.out

0 100.000000
1 100.000000
2 100.000000
3 100.000000
4 100.000000
5 100.000000
6 100.000000
7 100.000000
8 100.000000
9 100.000000
10 100.000000
11 100.000000
...
99 100.000000
```

→ https://developer.amd.com/wp-content/resources/ROCm%20Learning%20Centre/chapter3/HIP-Coding-3.pdf
→ https://rocmdocs.amd.com/en/latest/Programming_Guides/HIP-porting-guide.html
→ https://forums.extremehw.net/topic/782-solvedimage-upload-broken-on-firefox

HIP cont.

Compile and Run and Monitor

(zen3) [sh@some − mi250x ~]$ hipcc vadd_hip.cpp
(zen3) [sh@some − mi250x ~]$ ./a.out

```
0 100.000000
1 100.000000
2 100.000000
3 100.000000
4 100.000000
5 100.000000
6 100.000000
7 100.000000
8 100.000000
9 100.000000
10 100.000000
11 100.000000
...
99 100.000000
```

```
~
tictoc@TickTockArch $ rocm-smi


=======================ROCm System Management Interface=======================
==============================================================================
GPU  Temp   AvgPwr  SCLK     MCLK     Fan   Perf  PwrCap  VRAM%  GPU%
0    42.0c  294.0W  1925Mhz  1000Mhz  0.0%  high  450.0W   17%   100%
1    50.0c  219.0W  1772Mhz  1000Mhz  0.0%  high  220.0W    4%   100%
2    45.0c  273.0W  1925Mhz  1000Mhz  0.0%  high  300.0W    4%   100%
3    55.0c  163.0W  1700Mhz  1000Mhz  0.0%  high  170.0W    4%   100%
==============================================================================
============================End of ROCm SMI Log ============================
~
tictoc@TickTockArch $ _
```

→ https://developer.amd.com/wp-content/resources/ROCm%20Learning%20Centre/chapter3/HIP-Coding-3.pdf
→ https://rocmdocs.amd.com/en/latest/Programming_Guides/HIP-porting-guide.html
→ https://forums.extremehw.net/topic/782-solvedimage-upload-broken-on-firefox

Take Home Messages

- CUDA can be easily used in Fortran programs

Take Home Messages

- CUDA can be easily used in Fortran programs
- BLAS calls may be substituted with calls to CUBLAS potentially offering significant acceleration

Take Home Messages

- CUDA can be easily used in Fortran programs
- BLAS calls may be substituted with calls to CUBLAS potentially offering significant acceleration
- Basic LAPACK calls may be replaced with CUSOLVER routines

Take Home Messages

- CUDA can be easily used in Fortran programs
- BLAS calls may be substituted with calls to CUBLAS potentially offering significant acceleration
- Basic LAPACK calls may be replaced with CUSOLVER routines
- Sparse problems are supported with specific libraries, other basic tasks as well, CUFFT, CURAND, CUDNN

Take Home Messages

- CUDA can be easily used in Fortran programs
- BLAS calls may be substituted with calls to CUBLAS potentially offering significant acceleration
- Basic LAPACK calls may be replaced with CUSOLVER routines
- Sparse problems are supported with specific libraries, other basic tasks as well, CUFFT, CURAND, CUDNN
- Taking advantage of fast FP32 operations with quasi-FP64 accuracy on consumer grade cards requires a couple of algorithmic changes

Take Home Messages

- CUDA can be easily used in Fortran programs
- BLAS calls may be substituted with calls to CUBLAS potentially offering significant acceleration
- Basic LAPACK calls may be replaced with CUSOLVER routines
- Sparse problems are supported with specific libraries, other basic tasks as well, CUFFT, CURAND, CUDNN
- Taking advantage of fast FP32 operations with quasi-FP64 accuracy on consumer grade cards requires a couple of algorithmic changes
- NVIDIA's NSIGHT compute is an advanced profiling tool for in-depth kernel optimization

Take Home Messages

- CUDA can be easily used in Fortran programs
- BLAS calls may be substituted with calls to CUBLAS potentially offering significant acceleration
- Basic LAPACK calls may be replaced with CUSOLVER routines
- Sparse problems are supported with specific libraries, other basic tasks as well, CUFFT, CURAND, CUDNN
- Taking advantage of fast FP32 operations with quasi-FP64 accuracy on consumer grade cards requires a couple of algorithmic changes
- NVIDIA's NSIGHT compute is an advanced profiling tool for in-depth kernel optimization
- AMD's HIP is very similar to CUDA and kernel code can be directly interchanged