GPU PROGRAMMING WITH CUDA

An Introduction to CUDA Fortran

April 29, 2022 | Kaveh Haghighi Mood | JSC

JÜLICH Forschungszentrum

OVERVIEW

- Introduction
- CUDA Fortran basics
- Kernel loop directives (CUF kernels)
- Drop-in Fortran array intrinsics acceleration with CuTENSOR
- CUDA Fortran Limitations
- ISO standard Fortran + GPUs
- Resources

WHY CUDA FORTRAN?

- GPU support in native Fortran language
- Libraries and directive-based programming models are not flexible enough
- Not so difficult!
- Interoperable with OpenACC
- Similar to CUDA C
- CUDA Libraries

FORTRAN VS CUDA FORTRAN

Fortran

```fortran
program testVecAdd
use mathOps
implicit none

 integer, parameter :: N = 40000
 real :: a(N)

 a = 10.0
 call vecAdd(a,1.0)
 print*,"max_diff=", maxval(a-11.0)

end program testVecAdd
```

```fortran
module mathOps
contains

 subroutine vecAdd(a,b)
 implicit none

  real :: a(:)
  real :: b
  integer :: i, n

  n = size(a)
  do i=1,n
    a(i)=a(i)+b
  enddo

 end subroutine vecAdd
end module mathOps
```

FORTRAN VS CUDA FORTRAN

CUDA Fortran

```fortran
program testVecAdd
use mathOps
use cudafor
implicit none

 integer, parameter :: N = 40000
 real :: a(N)
 real,device :: a_d(N)
 integer tBlock, grid

 a = 10.0
 a_d = a
 tBlock = 256
 grid = ceiling(real(N)/tBlock)
 call vecAdd<<<grid,tBlock>>>(a_d,1.0)
 a = a_d
 print*,"max_diff=", maxval(a-11.0)

end program testVecAdd
```

```fortran
module mathOps
contains
 attributes(global) subroutine vecAdd(a,b)
 implicit none

  real :: a(:)
  real,value :: b
  integer :: i, n

  n = size(a)
  i= blockDim%x*(blockIdx%x-1)+threadIdx%x
  if (i=<n) then
    a(i)=a(i)+b
  endif

 end subroutine vecAdd
end module mathOps
```

CUDA FORTRAN BASICS

Data management

- Fortran enabled for CUDA
  - `device` attribute ⟶ declare variables in the device memory
  - `managed` attribute ⟶ declare unified memory arrays
  - Standard Fortran array assignment ⟶ data copies between host and device + sync
  - Standard Fortran `allocate` and `deallocate` ⟶ for both host and device allocations
- CUDA API calls ⟶ memory copy functions (cudaMemcpy, cudaMemcpy2D,...) are also available
- Scalars ⟶ CUDA runtime responsibility, if passed by value

JÜLICH Forschungszentrum

CUDA FORTRAN BASICS

Kernel launch

- Fortran enabled for CUDA
  - triple chevron notation:

    ```
    call kernel<<<grid,block[,bytes][,streamid]>>>(arg1,arg2,...)
    ```
  - `attributes(global)` ⟶ mark kernel subroutines
  - `use cudafor` ⟶ CUDA Fortran types (`blockDim%x, blockIdx%x`)
- Similar to CUDA C loops are replaced with bound checks
- Launch parameters can be extended to two and three dimensions with `dim3` derived type:

```
type(dim3) :: gridDim, blockDim

 blockDim = dim3(32,32,1)
 gridDim = dim3(ceiling(real(NN)/tBlock%x), ceiling(real(NM)/tBlock%y), 1)
 call calcKernel<<<gridDim,blockDim>>>(A_dev,Anew_dev)
```

TASK1

The first CUDA Fortran program

In this exercise, we'll scale a vector (array) of single-precision numbers by a scalar.

- Navigate to:

  `~/CUDA-Course/11-CUDA-Fortran/exercises/tasks/scale_vector`

- Look at `Instructions.ipynb` for instructions
- Call `source setup.sh` to load the modules of this task into your environment

IMPORTANT NOTES

- `use cudafor` is necessary to use CUDA Fortran types
- The Fortran array notation should be used for simple data transfers not complicated calculations
- Only one device array is allowed on the right hand side. Following statement is not legal: $A = C\_dev + B\_dev$
- CUDA Fortran source code should have .cuf or .CUF extension or you can add "-cuda" to compiler flags

TASK2

Jacobi solver with explicit kernel

- Navigate to:

  `~/CUDA-Course/11-CUDA-Fortran/exercises/tasks/jacobi-explicit`

- Look at `Instructions.ipynb` for instructions
- Call `source setup.sh` to load the modules of this task into your environment

CUF KERNELS

- To many loops? Reductions? Writing kernels is difficult?
- Compiler can write kernels for you, using `!$CUF` directive:

```
!$cuf kernel do[(n)] <<< grid, block, stream=streamid >>>
  do i=1,N
   do j=1,M
    do k=1,P
      ...
    enddo
   endo
  enddo
```

CUF KERNELS

- Compiler can choose launch parameters, if "∗" is used
- The n parameters after do, denotes the minimum debt of nested loops
- DO loops must have invariant loop limits
- GOTO or EXIT statements are not allowed
- Array syntax are not allowed

TASK3

Jacobi solver with kernel loop directives

- Navigate to:

  `~/CUDA-Course/11-CUDA-Fortran/exercises/tasks/jacobi-cuf`

- Look at `Instructions.ipynb` for instructions
- Call `source setup.sh` to load the modules of this task into your environment
- Compare the results with the explicit kernel version

FORTRAN ARRAY INTRINSICS WITH CUTENSOR

- `nvfortran` compiler can map Fortran intrinsic to to CuTENSOR
- Close to zero efforts for acceleration of `matmul`, `transpose`, `reshape` functions!
- Just add `use cutensorex` and recompile with `-cudalib=cutensor` !

TASK4

Fortran array intrinsics using Tensor Cores

- Navigate to:

  `~/CUDA-Course/11-CUDA-Fortran/exercises/tasks/matmul-cutensor`

- Look at `Instructions.ipynb` for instructions
- Call `source setup.sh` to load the modules of this task into your environment
- Compare the calculation time with and without the cuTENSOR

JÜLICH Forschungszentrum

TASK4

Fortran array intrinsics using Tensor Cores

Results on JUWELS Booster (gflops):

Size	8192	16384
Naive CUDA shared mem implementation	1945	2205
cuTENSOREX	16083	16435

CUDA FORTRAN LIMITATIONS

- Not portable! You have to use Nvidia GPUs
- Supported only by Nvidia HPC SDK (formerly known as PGI) and IBM XL Fortran compilers
- For some CUDA libraries, you have to write interfaces
- Small community

ISO STANDARD FORTRAN + GPUS!

- Non-standard libraries, directives or language extensions are not attractive enough?
- Standard portable acceleration is possible now!
- Fortran 2008 `DO CONCURRENT` supported by *nvfortran*:

```fortran
subroutine vecAdd(a,b)
implicit none

real :: a(:)
real :: b
integer :: i, n

n = size(a)
do i=1,n
   a(i)=a(i)+b
enddo

end subroutine vecAdd
```

```fortran
subroutine vecAdd(a,b)
implicit none

real :: a(:)
real :: b
integer :: i, n

n = size(a)
do concurrent (i = 1: n)
  a(i)=a(i)+b
enddo

end subroutine vecAdd
```

ISO STANDARD FORTRAN ON GPUS!

- Correctness? $\longrightarrow$ You are responsible
- Data transfer? $\longrightarrow$ Compiler and runtime env
- Additional `-stdpar` compilation flag is necessary
  - `-stdpar=multicore` $\longrightarrow$ compiles for CPU
  - `-stdpar=gpu,multicore` $\longrightarrow$ compiles for GPU or CPU

ISO STANDARD FORTRAN ON GPUS!

- Nested loop example:

```
do i = 1, n
  do j =1,m
    C(i,j)=a(i)+b(j)
  enddo
enddo
```

```
do concurrent (i = 1: n, j=1: m)
  C(i,j)=a(i)+b(j)
enddo
```

- Data privatization:

```
DO CONCURRENT (...) [locality-spec]
```

`locality-spec` options:

```
local(list)
local_init(list)
share(list)
```

ISO STANDARD FORTRAN TASK

Jacobi solver with do concurrent

- Navigate to:

  `~/CUDA-Course/11-CUDA-Fortran/exercises/tasks/jacobi-std`

- Look at `Instructions.ipynb` for instructions
- Call `source setup.sh` to load the modules of this task into your environment
- Compare the results with the explicit and CUF kernel versions

RESOURCES

- CUDA Fortran for Scientists and Engineers by Ruetsch and Fatica 2013
- CUDA Fortran Porting Guide
- CUDA Fortran Programming Guide and Reference
- Examples:
  `NVHPC-INSTALLDIR/arch/version/examples`

RESOURCES

- CUDA Fortran for Scientists and Engineers by Ruetsch and Fatica 2013
- CUDA Fortran Porting Guide
- CUDA Fortran Programming Guide and Reference
- Examples:
  `NVHPC-INSTALLDIR/arch/version/examples`

Thank you for your attention!