NATESM GPU SESSION
DKRZ USER WORKSHOP

13 October 2022 | Andreas Herten, Kaveh Haghighi-Mood | Forschungszentrum Jülich

Outline

Exercises on Levante

- Everyone should have personal account on Levante!
- Please use own budgeting allocation (i.e. Slurm's `--account`, like `kg0166`)
- Our course environment: Shortcuts to *normalize* work in this hands-on session
- Please login to Levante and source course environment:

```
$ account=kg0166 source /scratch/workshop/source-levante.sh
```

  (Replace `kg0166` with your budget.)
- Available afterwards
  - `material-sync` command to sync material to `$HOME` (also: `material-sync-force`)
  - Pre-populated `make run` target in exercises which will submit jobs to Slurm
  - Loaded environment modules: `nvhpc`

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

```
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
```

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

CUDA Fortran Basics

Kernel launch

- Fortran enabled for CUDA
  - triple chevron notation:

    ```
    call kernel<<<grid,block[,bytes][,streamid]>>>(arg1,arg2,...)
    ```
  - `attributes(global)` $\longrightarrow$ mark kernel subroutines
  - `use cudafor` $\longrightarrow$ CUDA Fortran types (`blockDim%x, blockIdx%x`)
- Like in CUDA C: replace loops with bound checks
- Extend launch parameters to 2 or 3 dimensions: use `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)
```

Task: Simple CUDA Fortran

TASK

Scale Vector

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

- Location: `CUDA-Fortran/tasks/scale_vector`
- Operation: $y_i = \alpha * x_i$
- Look at Instructions document, pick either Unified Memory version or explicit version
- Make sure to have the Levante setup sourced, `source levante-setup.sh` in the root!

Important Notes

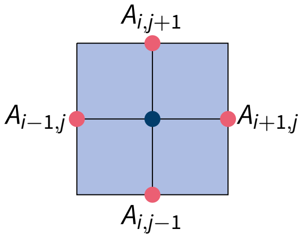
- `use cudafor` necessary for CUDA Fortran types
- Fortran array notation: use only for simple data transfers, not complicated calculations
- Only one device array is allowed on right hand side. Following statement is not legal:
  $A = C\_dev + B\_dev$
- CUDA Fortran source: .cuf or .CUF extension; or add `-cuda` to compiler flags

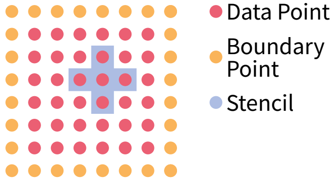
JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

Task: (Little) Advanced CUDA Fortran

Jacobi solver with explicit kernel

TASK





- Location: `CUDA-Fortran/tasks/jacobi-explicit`
- Operation: $y_{i,j} = (x_{i+1,j} + x_{i-1,y} + x_{i,j+1} + x_{i,j-1})/4$
- Look at Instructions document
- Make sure to have the Levante setup sourced, `source levante-setup.sh` in the root!

CUF Kernels *(Kernel Loop Directive)*

Overview

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

```fortran
!$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

Options

- Compiler can choose launch parameters, if "`*`" is used for launch configuration
- n parameter after do denotes the minimum depth of nested loops
- Limits
  - `do` loops must have invariant loop limits
  - `goto` or `exit` statements not allowed
  - Array syntax not allowed

Task: CUF Kernels

TASK

Jacobi Solver with Kernel Loop Directives

- Location: `CUDA-Fortran/tasks/jacobi-cuf`
- Look at Instructions document
- Compare results with explicit kernel version

Libraries in Fortran

cuTENSOR Example

- NVHPC provides pre-made Fortran interfaces to CUDA libraries, like cuBLAS, cuFFT, cuRand, cuSPARSE, …
  $\rightarrow$ `docs.nvidia.com/hpc-sdk/compilers/fortran-cuda-interfaces/`
- Also, special extensions for Fortran in cuTENSOR: `cutensorEx`
- `nvfortran` compiler can map Fortran intrinsic natively to cuTENSOR
- Nearly zero efforts for acceleration of `matmul`, `transpose`, `reshape` functions!
- Just add `use cutensorEx` and recompile with `-cudalib=cutensor`!

Task: cuTENSOR Library

TASK

Fortran Array Intrinsics using Tensor Cores

- Location: `CUDA-Fortran/tasks/matmul-cutensor`
- Different example (because Tensor Cores): Matrix Multiplication
- Look at Instructions document
- Compare the calculation time with and without the cuTENSOR

Task: cuTENSOR Library

TASK

Fortran Array Intrinsics using Tensor Cores

- Location: `CUDA-Fortran/tasks/matmul-cutensor`
- Different example (because Tensor Cores): Matrix Multiplication
- Look at Instructions document
- Compare the calculation time with and without the cuTENSOR
- Results on JUWELS Booster (GFLOP/s):

Size	8192	16 384
Naive CUDA shared mem implementation	1945	2205
cuTENSOR	16 083	16 435

JÜLICH Forschungszentrum JÜLICH SUPERCOMPUTING CENTRE

CUDA Fortran Limitations

- Not portable; only Nvidia GPUs
- Only via Nvidia HPC SDK (formerly known as PGI) *and IBM XL Fortran compilers*
- For some CUDA libraries, need to write interfaces
- Small community

ISO Standard Fortran with 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` and Intel oneAPI 2022.3:

```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 with 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` and Intel oneAPI 2022.3:

```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
```

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

ISO GPU Details

- You are responsible for **correctness**
- Data transfer: Compiler and runtime env
- Additional `-stdpar` compilation flag necessary

`-stdpar=multicore`	Compile for CPU
`-stdpar=gpu,multicore`	Compile for GPU or CPU

ISO GPU Details

- You are responsible for **correctness**
- Data transfer: Compiler and runtime env
- Additional `-stdpar` compilation flag necessary

`-stdpar=multicore`	Compile for CPU
`-stdpar=gpu,multicore`	Compile for GPU or CPU

- Definition: `do concurrent (...) [locality-spec]`
  Locality options: `local(list)`, `local_init(list)`, `share(list)`

ISO GPU Details

- You are responsible for **correctness**
- Data transfer: Compiler and runtime env
- Additional `-stdpar` compilation flag necessary

`-stdpar=multicore`	Compile for CPU
`-stdpar=gpu,multicore`	Compile for GPU or CPU

- Definition: `do concurrent (...) [locality-spec]`
  Locality options: `local(list)`, `local_init(list)`, `share(list)`
- 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
```

Task: ISO GPU

TASK

Jacobi Solver with `do concurrent`

- Location: `STD-Fortran/tasks/jacobi-std`
- Look at Instructions document
- Compare results with explicit and CUF kernel versions

OpenACC

- High-level programming model for GPUs *et al.*
- Similar to OpenMP, but better GPU support earlier
- OpenACC: more descriptive compared to OpenMP
- Needed: OpenACC-capable compiler; **NVIDIA HPC SDK**, GCC, Clang

OpenACC

- High-level programming model for GPUs *et al.*
- Similar to OpenMP, but better GPU support earlier
- OpenACC: more descriptive compared to OpenMP
- Needed: OpenACC-capable compiler; **NVIDIA HPC SDK**, GCC, Clang
- Directives Overview:

`parallel`	Start parallel region
`loop`	Create loop parallelism; usually `parallel loop` combination
`kernels`	Accelerate a full region, with much leeway for compiler
`data`	Create region for which data is transferred to and resides on GPU

OpenACC

- High-level programming model for GPUs *et al.*
- Similar to OpenMP, but better GPU support earlier
- OpenACC: more descriptive compared to OpenMP
- Needed: OpenACC-capable compiler; **NVIDIA HPC SDK**, GCC, Clang
- Directives Overview:

`parallel`	Start parallel region
`loop`	Create loop parallelism; usually `parallel loop` combination
`kernels`	Accelerate a full region, with much leeway for compiler
`data`	Create region for which data is transferred to and resides on GPU

- All directives have clauses (*options*), like

```
!$acc parallel loop reduction(max:C) gang vector copy(A)
```

OpenACC

- High-level programming model for GPUs *et al.*
- Similar to OpenMP, but better GPU support earlier
- OpenACC: more descriptive compared to OpenMP
- Needed: OpenACC-capable compiler; **NVIDIA HPC SDK**, GCC, Clang
- Directives Overview:

`parallel`	Start parallel region
`loop`	Create loop parallelism; usually `parallel loop` combination
`kernels`	Accelerate a full region, with much leeway for compiler
`data`	Create region for which data is transferred to and resides on GPU

- All directives have clauses (*options*), like

```
!$acc parallel loop reduction(max:C) gang vector copy(A)
```

→ `www.openacc.org/specification`

OpenACC Example

```c
#pragma acc data copyout(y[0:N]) create(x[0:N])
{
double sum = 0.0;
#pragma acc parallel loop
for (int i=0; i<N; i++) {
    x[i] = 1.0;
    y[i] = 2.0;
}

#pragma acc parallel loop
for (int i=0; i<N; i++) {
    y[i] = i*x[i]+y[i];
}
}
```

```fortran
!$acc data copyout(y(1:N)) create(x(1,N))

sum = 0.0;
!$acc parallel loop
do i = 1, N
    x(i) = 1.0
    y(i) = 2.0
end do
!$acc end parallel loop
!$acc parallel loop
do i = 1, N
    y(i) = i*x(i)+y(i)
end do
!$acc end parallel loop
!$acc end data
```

Task: OpenACC

TASK

Jacobi Solver with `do concurrent`

- Location: `OpenACC/tasks/`
- Look at Instructions document
- Compare results to other experiments

Exercises on Levante

- Everyone should have personal account on Levante!
- Please use own budgeting allocation (i.e. Slurm's `--account`, like `kg0166`)
- Our course environment: Shortcuts to *normalize* work in this hands-on session
- Please login to Levante and source course environment:

  ```
  $ account=kg0166 source /scratch/workshop/source-levante.sh
  ```

  (Replace `kg0166` with your budget.)
- Available afterwards
  - `material-sync` command to sync material to `$HOME` (also: `material-sync-force`)
  - Pre-populated `make run` target in exercises which will submit jobs to Slurm
  - Loaded environment modules: `nvhpc`

Ressources, Conclusion

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`
- Intel OneAPI 2022.3 Release Notes
- AMD’s gpufort (source-to-source converter to OpenMP)

Conclusion

- Many ways to use Fortran with Nvidia GPUs, all with NVHPC
  - CUDA Fortran
  - CUF Kernels
  - Libraries
  - ISO Standard
  - OpenACC
- Few ways to use Fortran with other GPUs

Conclusion

- Many ways to use Fortran with Nvidia GPUs, all with NVHPC
  - CUDA Fortran
  - CUF Kernels
  - Libraries
  - ISO Standard
  - OpenACC
- Few ways to use Fortran with other GPUs

Thank you for your attention!

a.herten@fz-juelich.de

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

Appendix

References: Images, Graphics I

[1] Forschungszentrum Jülich GmbH (Ralf-Uwe Limbach). *JUWELS Booster*.