Advanced Fortran Topics - Hands On Sessions

Inhalt

Introductory Comments .................................................................................................................................................. 2
Exercises for Day 1 ......................................................................................................................................................... 3
  Session 1 – Reverse communication interface .............................................................................................................. 3
  Session 1 – Dynamic memory management for the heat example .......................................................................................... 3
  Session 2 – some performance tunings for the heat example .............................................................................................. 4
  Session 2 – Operations on sparse matrices ..................................................................................................................... 4
  Session 3 – Sparse matrices continued ............................................................................................................................ 5
  Session 4 – Physical bodies .............................................................................................................................................. 5
  Session 4 – Implement the date class (part 1) ..................................................................................................................... 5
Exercises for Day 2 ......................................................................................................................................................... 6
  Session 5 – Implement the date class (part 2) ..................................................................................................................... 6
  Session 5 – Lazy deletion of function values ................................................................................................................... 6
  Session 5 – Geometric objects ........................................................................................................................................... 6
  Session 6 – improving an interface class ............................................................................................................................ 7
  Session 6 – Collision of physical bodies ........................................................................................................................... 7
  Session 7 – Using asynchronous I/O ................................................................................................................................ 7
  Session 7 – User-defined derived type I/O for sparse matrices ........................................................................................ 7
Exercises for Day 3 ......................................................................................................................................................... 8
  Session 8 – Using IEEE exceptions ................................................................................................................................... 8
  Session 9 – Parameterized derived types ........................................................................................................................... 8
  Session 10 (easy) – Calling GSL from Fortran .................................................................................................................. 8
  Session 10 (more difficult) – Using a POSIX call from Fortran .......................................................................................... 9
  Session 11 – Dynamic dummy arguments ........................................................................................................................ 9
  Session 12 – C library interface with function argument ................................................................................................ 9
Exercises for Day 4 ......................................................................................................................................................... 11
  Session 13 – Trajectories of Physical Bodies .................................................................................................................. 11
  Session 14 – Integration example ...................................................................................................................................... 11
  Session 15 – Reverting an object to its earlier state ....................................................................................................... 11
  Session 16 – Implementing the observer simulation .................................................................................................... 12
  Session 17 – Data distribution ........................................................................................................................................ 12
Exercises for Day 5........................................................................................................................................... 13
Session 18 – Heat conduction parallelized .................................................................................................. 13
Session 19 (alternative 1) – triangular matrix-vector multiplication.......................................................... 13
Session 19 (alternative 2) – parallelizing a ray tracer.................................................................................. 14
Session 20 – Completing the heat example................................................................................................ 14
Session 21 (optional) – parallel library call ................................................................................................. 14

Introductory Comments

• Before you start the first exercise, please read the appropriate chapter from the organizational document to perform setup of the course environment
• There is no need to do all the exercises.
• You are bound to make mistakes. Therefore, writing test programs (comparing expected with actual results) is a vitally important step in the development cycle. And one learns more from the mistakes than from the successes.
• Source code including examples from the talk and solutions for the exercises will be provided at the end of each day via the download URL: https://doku.lrz.de/display/PUBLIC/Materials+-+Advanced+Fortran+Topics

(You can use the wget command to perform downloads via UNIX command line by doing copy and paste of an archive name URL).
Exercises for Day 1

Session 1 – Reverse communication interface
Some knowledge of OpenMP is useful for doing this exercise.

The directory `hands_on/skel/reverse_communication` contains example code for the integration routine that uses reverse communication, as discussed in the slides.

a. Why is that code (still) not thread-safe? You can check the fact by activating the OpenMP clauses in the calling program and running with OMP_NUM_THREADS set to a value larger than 1.

b. Re-design the code to be thread-safe without adding any OpenMP directives in the module `mod_integration`. Hint: Introduce a derived type.

Session 1 – Dynamic memory management for the heat example
The directory `hands_on/skel/heat` contains source code for the heat conduction example from the slide talk. This version of the program is rather inflexible, because you need to recompile it if you want to run a different problem size. Make changes to the source files that allow you to determine the problem size at **run time** by reading the array dimensions `nx` and `ny` from standard input via the `GET_COMMAND_ARGUMENT` and `GET_COMMAND_ARGUMENT_COUNT` intrinsics.

**Note:** A simple example for how to parse command arguments is available in the folder `examples/day1/command_line`. Also, you should consult the Intel and the gfortran compiler documentation for complete information on how to use the intrinsics.

The executable should then be invoked e.g., via

```
./heat.exe -nx 50 -ny 100
```

for a problem size 50 x 100. The following steps will be also needed:

- Make the necessary changes to the declarations of the fields `phi` and `phinew`. What else is needed to assure the fields are set up correctly?
- Note that the x and y directions now may be differently discretized, and the space differentials must be set up at run time.

The correctness can be checked by running a small problem size (10 x 10) to conclude fast, and activating a printout done while the iteration still executes. When you are done, perform comparative performance measurements for the problem size 200 x 200. Note that adding suitable vectorization options can significantly improve performance.

The solution for this exercise will be in the folder `hands_on/solutions/heat`, in the files `mod_heat.f90` and `heat.f90`. 
Session 2 – some performance tunings for the heat example

Return to the heat conduction example from session 1 and do some performance evaluations.

a. first, add the TARGET attribute to the phi and phinew arrays and observe the impact on performance for \( nx = ny = 200 \).

b. Due to copying of phinew to phi, a factor of two more memory accesses are needed than would be the case if the fields had the POINTER attribute and the pointers are simply switched between iterations. Make a copy of the module within which just that is done and check the performance (possibly for different compilers).

c. Does adding the CONTIGUOUS attribute to the pointer fields improve performance? If not, consider possibilities to eliminate the POINTER attribute for the duration of the calculations.

The solution for this exercise will be in the folder hands_on/solutions/heat, in the files mod_heat_ptr.f90 and heat_ptr.f90.

Session 2 – Operations on sparse matrices

The derived type sparse mentioned in the slides can be used to implement sparse matrices, and thus a sparse matrix-vector multiplication. A sparse matrix is represented by a rank-1 array

```
type( sparse ), allocatable :: sa(:)
```

which is allocated and subsequently has its (few) matrix elements set via repeated invocation of a module procedure set_element() on each array element of sa (the latter is intended to correspond to a matrix row in form of a linked list ordered by ascending index values). The folder hands_on/skel/sparse contains files mod_sparse.f90 and test_sparse.f90 copies of which you will need to suitably modify:

a. Add a module function that implements the multiplication of a sparse matrix by a real(dk) vector and overload the multiplication operator.

b. Run the resulting program and check the results. Then, check for memory leaks by building with debugging options and running the command through Valgrind:

```
valgrind --tool=memcheck --leak-check=yes --track-origins=yes --show-reachable=yes ./test_my_sparse.exe
```

Where does the observed problem come from?

c. Consider further changes to your program to avoid the problem.

Two slightly different solutions are provided in the hands_on/solutions/sparse folder as

- mod_sparse_simple.f90 and test_sparse_simple.f90
- mod_sparse.f90 and test_sparse.f90

Note: the sparse implementation demonstrated here is horribly inefficient, because pointer chasing causes loss of spatial locality. A number of much better-performing storage schemes for sparse matrices exist, which you should use in productive environments.
Session 3 – Sparse matrices continued

In the slides for session 2, an overloaded structure constructor was shown for an object of type `sparse`. This suggests a much more efficient way of storing the row of a sparse matrix. Create a new folder, copy the module to it and improve the type definition. Apart from adding a constructor overload, you will also need to modify the implementations of the module procedures, of course. Then check that the main program still executes with the same results without any changes to it. Finally, create a new main program that uses the overloaded constructor instead of setting elements individually. The solution to this exercise will be contained in `hands_on/solutions/sparse_crs`.

Session 4 – Physical bodies

Starting from the type definition:

```fortran
type :: body
  real(kind=dk) :: mass
  real(kind=dk) :: pos(3), vel(3)
end type
```

please define two extensions of that type (in the same module that defines the above type) which describe

- a body that has an electrical charge,
- a body that rotates.

Then write a procedure `print_body` that prints out all components of an object whose dynamic type may be any of the three types described above.

In a main program, please declare an allocatable polymorphic variable of declared type `body`, allocate it to be in turn any of the three types above, define reasonable values, and invoke `print_body` on that object.

Suppose a physical body starts out being of `type(body)` but later acquires a charge (for example, by being struck by lightning). How can the previous values of the non-charge type components be retained when the object changes its dynamic type?

Finally, declare an unlimited polymorphic entity in the main program and allocate it to be of one of the above types. How can you invoke `print_body` on that object?

You can start out from the skeleton code given in `hands_on/skel/polymorphic_body`. The solution will be available in `hands_on/solutions/polymorphic_body`.

Session 4 – Implement the date class (part 1)

Fill in all necessary details needed for implementation of the date and datetime types discussed in the lecture. Some subroutines which do the necessary computational stuff are provided in `hands_on/skel/datetime` (please ignore the file `person.f90` for now). Please start out by implementing the missing subroutines `inc_date` and `inc_datetime` as well as overloading the structure constructor for the above types.
Exercises for Day 2

Session 5 – Implement the date class (part 2)
Bind the procedures written in part 1 of this exercise as well as write_date to their appropriate types and uncomment the statements in the test program that execute them, to check that your code works correctly. The solution for this is in the folder hands_on/solutions/datetime.

Session 5 – Lazy deletion of function values
Suppose you have a type definition

```fortran
type :: person
  private
  character(len=nmx) :: name
  class(date), pointer :: birthday => null()
end type
```

and functions that initialize objects of that type, or return the name or birthday of a person. How can you print out the birthday of a person without needing to copy an object? The datetime skeleton from the exercise above also contains a file person.f90 that you can start out from.

The solution for this is in the folder hands_on/solutions/datetime. Note: Because many compilers mishandle this, please check against the result produced by the NAG compiler.

Session 5 – Geometric objects
A rectangle might be characterized by the type definition

```fortran
type :: rectangle
  private
  real(dk) :: length = 0.0_dk
  real(dk) :: breadth = 0.0_dk
end type
```

Implement a type-bound procedure which calculates the area of a rectangle, as well as an initialization procedure and a procedure to perform a stretching of a rectangle along a chosen direction. Since a square is a special kind of rectangle, one might consider the following type definition:

```fortran
type, extends(rectangle) :: square
... ! fill in missing bits
end type
```

How do you handle the inheritance behavior, especially that of the type-bound procedures? Assume that a function adjust exists that takes an object of type(rectangle) as an argument and increases the size of the object if its area is too small. What happens if an entity of type(square) is used as an actual argument in an invocation of this function? Try to improve on the type design.

Please consult the two Fortran codes in hands_on/solutions/shapes for the “bad” and “good” versions of the design.
Session 6 – improving an interface class
The folder examples/day2/interface_class contains a runnable example for the interface class discussed in the lecture. Unfortunately, the main program still contains a dependency on the module that declares the type extension. What is the cause of the problem? Try to find a solution such that the statement “use mod_file_handle” can be replaced by “use mod_handle”.

The solution will be available in hands_on/solutions/interface_class.

Session 6 – Collision of physical bodies
Write a type-bound procedure that calculates the front-on collision of physical bodies, starting from the code written for the above exercise. The following assumptions may be made: For the base type, the conservation of energy and momentum defines the result. In the center-of-mass system that moves with velocity

\[ \vec{V} = \frac{(m_1\vec{v}_1 + m_2\vec{v}_2)}{(m_1 + m_2)} \]

the velocities \( \vec{w}_i' = \vec{v}_i - \vec{V} \) after the collision can be obtained from the corresponding \( \vec{w}_i \) before the collision by

\[ m_1\vec{w}_1' + m_2\vec{w}_2' = 0 = m_1\vec{w}_1 + m_2\vec{w}_2 \quad m_1w_1^2 + m_2w_2^2 = m_1w_1'^2 + m_2w_2'^2 \]

For two charged bodies that collide the charges after the collision are defined by

\[ \frac{q_1'}{R_1} = \frac{q_2'}{R_2} \]

and you can assume that the electrostatic interaction is negligible. The radii \( R_1, R_2 \) of the objects shall be supplied through the interface.

Session 7 – Using asynchronous I/O
The directory hands_on/skel/aio contains source code for a ray tracer. This code performs I/O of the complete picture array at the end of the calculation. The resulting file can be viewed with the display command. Convert this program to use asynchronous I/O by putting the data transfer statements inside the outer loop that processes the tiles in the picture. In particular, you can reduce the needed amount of storage from \( \text{size}^2 \) to \( \text{size}*\text{nbuf} \), where \( \text{nbuff} \) is the number of I/O buffers available for asynchronous I/O. For which picture size do you observe a performance advantage? For good I/O performance it is recommended to change to the $SCRATCH directory and execute the program there.

Session 7 – User-defined derived type I/O for sparse matrices
Starting from the solutions for the sparse matrix multiplication from session 2, add a generic binding and module procedures that support formatted as well as list-directed output of objects of type sparse. Modify the test program to exercise these features.

The solution for this exercise is contained in the folder hands_on/solutions/sparse (files mod_sparse_io.f90 and test_sparse_io.f90).
Exercises for Day 3

Session 8 – Using IEEE exceptions

Direct evaluation of the expression \(|x| = \sqrt{\sum x_i^2}\) (euclidean norm of a vector) may cause avoidable overflows. Use the IEEE overflow flag to deal with this situation in a function subprogram that evaluates the above expression. Please test your code with at least the following input:

\[
\begin{align*}
\text{v} & = [2.0, 2.0, 2.0, 2.0] \\
\text{v} & = [0.0, 0.0, \text{sqrt}(2.0), \text{sqrt}(2.0)] \\
\text{v} & = [0.0, 0.0, 0.0, 0.0] \\
\text{v} & = [0.0, 0.0, 0.0, 2.0*\text{tiny}(1.0)] \\
\text{v} & = [0.0, 0.0, 0.0, \text{tiny}(1.0)] \\
\text{v} & = [0.0, 0.0, \text{tiny}(1.0), \text{tiny}(1.0)] \\
\text{v} & = [2.0, 2.0, 2.0, \text{sqrt}(\text{huge}(1.0))] \\
\text{v} & = [2.0, 2.0, 2.0, 1.1*\text{sqrt}(\text{huge}(1.0))] \\
\end{align*}
\]

The solution will be provided in `hands_on/solutions/ieee_exceptions`.

Session 9 – Parameterized derived types

Take a look at the skeleton code you find inside `hands_on/skel/pdt/`.

1. As indicated, define an abstract parameterized derived type `matT` with kind parameter `k` and len parameters `m`, `n`. Then, define a type extension with a rank-2-real-array component `mat`. In the main program, declare allocatable objects of the kind indicated in the skeleton main program. Further, define an assignment that can handle different kind-parameters for `rmatT`, e.g. `real32` and `real64` and any len parameter. Also define a matrix multiplication that can handle different kind-parameters for `rmatT`, e.g. `real32` and `real64`. Try to get the indicated code fragments in the main program running.

2. Now add the private attribute to the parameterized type `rmatT`. What is required to get the main program to compile and run again?

Session 10 (easy) – Calling GSL from Fortran

Write a program that calculates the value of the error function

\[
erf(x) := \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt
\]

for values of the argument between 0 and 2. Use the implementation supplied by the GNU scientific library, and use the variant of the function that also calculates the error estimate. Documentation is available at

and the environment module `gsl` provides the necessary variables (`$GSL_LIB`) to link against the library. The solution for this exercise will be contained in `hands_on/solutions/interop_gsl`.

Note that a nearly complete Fortran interface to the library is available via [https://github.com/reinh-bader/fgsl/](https://github.com/reinh-bader/fgsl/)

### Session 10 (more difficult) – Using a POSIX call from Fortran

The API call `getpwnam()` allows to extract information about a specific user name from the authentication databases. Write a Fortran module and program that contains the necessary type definition, BIND(C) interfaces and module procedures to produce and print out a user's HOME directory. What must be taken care of in case a user name does not exist? Study the man page for `getpwnam()` for information on its interface.

Skeleton code for the exercise is in `hands_on/skel/interop_struct`. A fully working main program is available, but the module `mod_passwd` must be implemented.

The solution for this exercise will be found in `hands_on/solutions/interop_struct`.

### Session 11 – Dynamic dummy arguments

In the lecture, we've seen how C-defined types with pointer components can be handled in Fortran 2003. Now consider the reverse: How can the Fortran interface `subroutine generate_data(arr)`

```fortran
real(c_float), allocatable, intent(out) :: arr(:)
end subroutine
```

- which is non-interoperable (why?) - be used from C? Assume that the data for `arr` is generated within the subroutine and that these data should be accessible within C. See `hands_on/skel/interop_deferred_shape` for skeleton code.

With TS 29113 (extensions to C interoperability, currently only supported by the Intel compiler), it is permitted to directly use interfaces such as the above, i.e. it is possible to add BIND(C) to the above interface. This removes the need to introduce a “handle” derived type. Implement a C main program that uses the TS' facilities to do such a call after correctly setting up a descriptor for the allocatable object.

### Session 12 – C library interface with function argument

The folder `hands_on/skel/interop_interface` contains source files `c_libcall.[c,h]` implementing a library call with prototype

```c
float Sum_Fun( float (*fun)(float x, void *params), void *params );
```

which takes a function argument to be provided by the client. Furthermore, the following files are supplied:

- an example main program in C (`cmain.exe`) which you can build using the Makefile – you'll need to inspect this to see how the argument function is implemented.
- skeletons for a Fortran library module `f_lib.f90` and a Fortran main program `fmain.f90`, which will not successfully compile ...

Here's what you are required to do:
1. Please add the missing functionality in the Fortran code using only Fortran 2003-style interoperability features, so it performs the same function as the C code. What limitations apply?

2. Using the additional features from the interop TS 29113 (currently only supported in the Intel compiler), extend the semantics of the interface so that also non-interoperable arguments can be used from Fortran. Check that this works by calling the function with a polymorphic actual argument.

The solutions will be provided in hands_on/solutions/Interop_Interface.
Exercises for Day 4

Session 13 – Trajectories of Physical Bodies
Define a type that – reusing code from the Session 4 exercise “Physical Bodies” - is capable of describing the trajectory of a particle of class(body). This means that the physical state of the particle at some number of points in time must be stored inside an entity of such a type.

1. What type components must be defined to fully describe such a trajectory? Write an overloaded (generic) structure constructor for an entity of that type that defines an initial condition for the trajectory.

2. Write a procedure that updates the trajectory by adding an additional time step, assuming that mass or charge of the particle do not change. Deal with the storage limitation of an entity of type(trajectory) by writing data to disk if necessary, without changing existing interfaces.

3. What do you need to do if an entity of type(trajectory) goes out of scope or is deallocated?

Please start out with the skeleton code in hands_on/skel/trajectory. The entries marked FIXME indicate where missing bits need to be added; a test program that runs the code is also supplied. Reference output is provided in a subdirectory.

Session 14 – Integration example
Fill in the details on implementing the integration example from today's lecture. Some code which provides a simple default integrator is available in hands_on/skel/integration, as well as code that indicates how Fortran GSL interface functions (which provide more refined capabilities) should be used. Please note that the main program needs to change (read the FIXME entries there), and the qdr.f90 module needs appropriate updates.

Session 15 – Reverting an object to its earlier state
Define a type and associated type-bound procedures that can be used for an object that is intended to describe the solution of a system of linear equations, \( Ax = b \). The first step used should be invocation of the LAPACK routine SGESV. If this routine fails, it will be necessary to restore the matrix \( A \) to its earlier state before calling a procedure that calculates the singular value decomposition (in which case one can at best obtain a least squares deviate). Implement the Memento pattern necessary to perform this restoration in a well-encapsulated way. The skeleton code you can start out from for this exercise is in the folder hands_on/skel/linear_solution_space.
Session 16 – Implementing the observer simulation

Starting from the skeleton code in `hands_on/skel/observer`, complete the implementation of the observer simulation. The energy value and plotting slices should be updated in intervals corresponding to a noticeable change in the function $y(x, t)$; corresponding values of the simulation time should also be printed out. The skeleton code for this is in `hands_on/skel/observer`. Reference printout is also provided as `stdout.res`.

Session 17 – Data distribution

Please read the web page at [http://www.lrz.de/services/software/parallel/pgas/caf/](http://www.lrz.de/services/software/parallel/pgas/caf/) to inform yourself about how parallel Fortran programs are compiled and started. Then, consider a triangular matrix:

![Diagram of triangular matrix]

Typically $n \gg$ number of tasks

Make a copy of the serial program `hands_on/skel/triangle/triangular.f90` into your working directory. The program reads in matrix size and a row index from the command line. It then sets up $A(i, j) = i + j$

and prints out the specified row. Parallelize this program in a manner that distributes data as evenly as possible across images.
Exercises for Day 5

Session 18 – Heat conduction parallelized
Starting out from the skeletons `mod_heat.f90` and `heat.f90` introduced on day 1, parallelize this code using coarrays.

a. Introduce a one-dimensional domain decomposition along the y direction. One method to deal with the boundary cell problem consists in assigning each image an additional column (“halo” or “ghost” cells, colored grey in the figure below) at its boundaries to another domain which receives data from the task that hosts that domain:

Only the halo cells need to be involved in communication; in this example, these form contiguous arrays. For now, please only run a fixed (sufficiently large) number of iterations, omitting the termination criterion.

b. Start with a small problem size to check whether correct results (the printout from multiple tasks should be collected on image 1) are obtained.

c. When running with a problem size of 200 x 200, up to how many images does your code scale?

The solutions for this problem will be available in the folder `hands_on/solutions/heat_caf` (files `mod_heat_static.f90` and `heat_static.f90`).

Session 19 (alternative 1) – triangular matrix-vector multiplication
Continue the session 17 exercise by implementing the full matrix-vector multiplication for triangular matrices. Use the most efficient access pattern for calculation of the scalar products. The coarray used to store one of the vectors should be allocatable. Do not forget to implement the necessary communication to assure the complete result is available on all images. The solutions for this problem will be available in the folder `hands_on/solutions/triangle_full`. 
Session 19 (alternative 2) – parallelizing a ray tracer
The subdirectory hands_on/skel/ray contains a serial ray-tracer code (the same we've seen before in a different context), which computes a pretty picture. The central function is calc_tile(), which computes one tile of the picture. The size of one tile and of the whole picture is hardcoded at the start of the main program. Note that the code assumes that the picture size is a multiple of the tile size. In the version given, the picture size is 4000 x 4000 and the tile size is 200 x 200.

Parallelize the code using coarrays. You can deactivate the output for testing, but make sure that your parallel code computes the correct result (this is easy since you can always display the picture). What speedup does your code obtain going from 1 to 16 images? Also, compare with the baseline performance from the serial code.

Session 20 – Completing the heat example
Complete the parallel version of the heat conduction program by adding the necessary reduction call that enables consistent evaluation of the termination condition. Note that currently only gfortran supports the collectives defined in TS18508. For Intel Fortran, either a critical region must be used, or the Fortran collective must be replaced by an MPI call.

Session 21 (optional) – parallel library call
For the matrix-vector multiply, rewrite the main program to use a library call with a coarray dummy argument.