Fourth edition new examples and updated material

Authors

Ian D Chivers &
Jane Sleightholme

Overview

This document has coverage of

- updates of some of the examples in the fourth edition;
- updates of some of the timing runs with current versions of compilers (see the compiler chapter for individual details);
- new examples;
- additional material aimed at people attending our courses, but also of use more generally;

The fourth edition is available at

`https://www.fortranplus.co.uk/`

This is our primary Fortran site.

We will be providing the new and changed material in these notes, which will be available from our web site.

All files are available on our web sites. Our secondary web site Fortran home page is

`https://www.rhymneyconsulting.co.uk/fortran/`

The 4_edition_update.tar file contains all of the original examples, new examples, and a copy of these notes. A spreadsheet is also available covering additional information about the fourth edition examples and new examples. Two additional tabs are available in the spreadsheet giving details of module usage, and include file usage.

Acknowledgements

One of the places where ideas for new or updated examples comes from are the people we teach.

Thanks to everyone we've taught since the publication of the fourth edition for their input.

Attendees at the course in May 2023 include Ryan, Charlotte, Dimitrios, Lianne, Benjamin, Veryan, Lucy and Lucy. Thanks for your comments.

Recent history

April 2024

- Updated the shell scripts and batch files to bring in line with the latest compiler releases.
- Updated to reflect the latest Intel offerings.
- Added a short coverage of Fortran 2023.
- Updated with details of running the Nvidia Linux compiler suite under WSL 2.

December 2023

- Added a statistics module that supports 80 bit reals - gfortran only;
- Added a statistics module that supports 16 bit reals - nag only;

November and December 2023

- Updated the Intel oneAPI chapter with details of the 2024 release. This release provides support for Nvidia gpu programming.
- Updated the compiler chapter with updated timing for ch3801, ch3802, ch3803 and ch3806 with current compilers;

October and November 2023

- updated shell scripts and batch files that compile the examples;
- added draft template examples based on proposals for the next standard F202Y
- updated chapter 38 example 6 with additional tables for runs on two more hardware platforms and for additional compilers;
- reorganisation of chapters and examples.

June 2023

- spelling check;
- Updated the C interop chapter to have details of current compilation options. Intel currently offer 2 Fortran compilers and 2 C++ compilers, and we have details of the 4 combinations possible.
- Updated the compilers used chapter to have details of current compiler versions and options.
- Added new example to the pointer chapter looking at memory allocation for arrays by array type and by compiler.

Table of Contents

'The first thing we do, let's kill all the language lawyers.'
Henry VI, part II

1 Fourth edition update changes

1.1 Introduction

Here is the list of chapters with changes:

- General changes
  - Added use of the two iso_fortran_env functions compiler_version() and compiler_options() to some examples.
- Several new chapters. More details are given below.
  - Chapter 43 - New module to add commas to integers when printing. The original version only handled 64 bit integers. The current version handles 32 and 64 bit integers, and negative integers.
  - Chapter 43 - New example illustrating Kahan summation, with timing.
  - Chapter 44 - New C interop example which provides function access to the Windows API for memory usage - GlobalMemoryStatusEx function which is in sysinfoapi.h
  - Chapter 44 - New C interop example which provides function access to the Linux API for memory usage - provided in the <sys/sysinfo.h> header file.
  - Chapter 44 - Two modified Kahan summation example illustrating the use of the memory api functions on Windows and Linux.
  - Chapter 44 - 2 examples illustrating memory leaks under Windows and Linux.
  - Chapter 45 - Basic coverage of the Nvidia HPC toolkit and gpu programming. There are several examples on using an Nvidia gpu with timing figures.
  - Chapter 46 - Basic introduction to the Intel oneapi toolkits. No examples at this time.
  - Chapter 47 - Two examples of generic sorting modules using syntax from the Fortran 202Y standard. The first is based on a proposal from Japan, and the second is based on a proposal from a J3 work group. These are drafts.
  - Chapter 48 - Introduction to some of the new features in the Fortran 2023 standard. WORK IN PROGRESS
- 1 - Overview - None

- 2 - Introduction to Problem Solving - None
- 3 - Introduction to programming languages - Updates to the C and C++ entries to bring in line with the current standard situation. Added a section on the Fortran 2023 standard. The C++ 17 standard now supports parallel programming. A new example has been written to call the C++ parallel STL sort routines.
- 4 - Introduction to programming - None
- 5 - Arithmetic
  - Added an example in chapter 5 about the use of undefined variables.
  - Added coverage of the NAG compiler flag -C=undefined and the Intel flag /Qtrapuv
  - Modified ch0504p.f90 to calculate in seconds.
- 6 - Arrays 1: Some Fundamentals
  - Added a problem in chapter 6 about the use of an undefined value, and a repeat of the use of the NAG and Intel flags.
- 7 - Arrays 2: Further Examples - None
- 8 - Whole Array and Additional Array Features
  - Added a new 3 d reshape example.
- 9 - Output of Results - None
- 10 - Reading in data - None
- 11 - Summary of I/O concepts - None
- 12 - Functions - None
- 13 - Control Structures and execution control
  - Added an explicit forward reference in chapter 13 to the do concurrent example ch3305.f90 in the openMP chapter.
- 14 - Characters - None
- 15 - Complex - None
- 16 - Logical - None
- 17 - Introduction to Derived Types
  - Added an example in chapter 17 showing default constructor usage.
- 18 - An Introduction to Pointers
  - Additional coverage of the differences between variable and pointer status types
  - Deleted example seven
  - Duplicated versions of the first six examples to use the c_loc c interop function to provide details of what is happening behind the scenes.

    - New example looking at where arrays are allocated in memory
- 19 - Introduction to Subroutines - None
- 20 - Subroutines: 2 - None
- 21 - Modules - None
- 22 - Data structuring in Fortran
    - Added three new examples to chapter 22.
    - The first 2 are rewrites of the linked list examples to use allocatable components rather than pointers.
    - The third linked list example removes pointer usage altogether and uses move_alloc.
- 23 - An Introduction to Algorithms and the Big O notation - None
- 24 - Operator overloading - None
- 25 - Generic programming
    - added 2 new examples
    - generic stats module with 16 bit support - Nag only
    - generic stats module with 80 bit support - gfortran only
- 26 - Mathematical and numerical examples - None
- 27 - Parameterised derived types (PDTs) in Fortran - None
- 28 - Introduction to Object Oriented Programming - None
- 29 - Additional Object Oriented examples - None
- 30 - Introduction to submodules - None
- 31 - Introduction to parallel programming - None
- 32 - MPI - Message Passing Interface - None
- 33 - OpenMP - None
- 34 - Coarray Fortran - None
- 35 - C Interop
    - Added an example of passing a one d <vector> from C++ to Fortran. Idea came from some of the people from the UK Met Office attending a Fortran course in June 2022. Many thanks.
    - Added an example of passing a one d <array> for completeness.
- 36 - IEEE Arithmetic
    - Additional explanation in chapter 36 (IEEE arithmetic) of example ch3605 showing incorrect summation by the Intel compiler.

    - Additional material in the IEEE chapter to bring it up to date with the latest IEEE standards.
- 37 - Derived type I/O - None
- 38 - Sorting and searching
    - New sorting example calling the C++ STL parallel sorting routines from Fortran.
- 39 - Handling missing data in statistics calculations
    - Updated C# example in chapter 39 to get the Met Office station files
    - Added new Python program to get the Met Office station files.
    - New example doing missing data calculations using IEEE nans in chapter 39.
      This involves multiple versions of some of the files.
    - Added additional explanation in chapter 39 to cover the special processing required for the 3 closed stations
- 40 - Converting from Fortran 77 - None
- 41 - Graphics libraries - simple dislin usage - None
- 42 - Abstract interfaces and procedure pointers - None
- 43 - Miscellaneous additional examples - New chapter
- 44 - 6 new examples using the Windows and Linux memory apis.
- 45 - New chapter on GPU programming using Nvidia GPUs. There are currently 6 Nvidia GPU examples.
- 46 - New chapter on the Intel oneapi toolkit. No examples at this time.
- 47 - Two draft generic sorting modules based on proposed syntax from the Fortran 202Y draft standard.

1.2 Example list

A separate spreadsheet is available which documents all of the examples from the 4th edition, includes the new examples from the 4th edition update and has summary information on module usage for each example. It is included in the tar file.

1.3 Compiling the examples

We have written a set of Windows batch files and shell scripts to compile the examples. A later chapter has more information, and the tar file contains the various batch files and shell scripts.

2 Introduction to programming languages

2.1 Chapter 3 - changes

There have been changes to the C, C++ and Fortran entries.

2.2 C

There have been several versions of C. Before the language was standardised most people relied on an informal specification contained in the book by Dennis Ritchie and Brian Kernighan, and this version is called K&R C.

The following table summarises the C publication and standardisation history.

Common name	Date	Standard name and version
Initial release	1972	
K and R C	1978	
ANSI C, C89	1989-1990	ANSI X3.159-1989
ISO C, C90	1990	ISO/IEC 9899:1990
C99, C9X	1999	ISO/IEC 9899:1999
C11, C1X	2011	ISO/IEC 9899:2011
C17	2018	ISO/IEC 9899:2018
C23, C2X	2024	ISO/IEC 9899:2024

C99 introduced several new features, including inline functions, several new data types (long long int and complex), variable-length arrays, improved support for IEEE 754 floating point, and support for one-line comments beginning with //, as in C++.

The C11 standard added new features to C and the library, including type generic macros, anonymous structures, improved Unicode support, atomic operations, multi-threading, and bounds-checked functions, improved compatibility with C++.

C17 introduced no new language features, only technical corrections, and clarifications
to defects in C11.

C23 is a major update. Here is a link to a comprehensive coverage.

https://en.wikipedia.org/wiki/C23_(C_standard_revision)

C2Y is a work in progress.

2.3 C++

The following tablel summarises the C++ standardisation versions.

Year	C++ Standard	Informal name
1998	ISO/IEC 14882:1998	C++98
2003	ISO/IEC 14882:2003	C++03
2007	ISO/IEC TR 19768:2007	C++TR1
2011	ISO/IEC 14882:2011	C++11
2014	ISO/IEC 14882:2014	C++14
2017	ISO/IEC 14882:2017	C++17
2020	ISO/IEC 14882:2020	C++20

In 1998, C++98 was released, standardizing the language, and a minor update (C++03) was released in 2003.

After C++98, C++ evolved relatively slowly until, in 2011, the C++11 standard was released, adding numerous new features, enlarging the standard library further, and providing more facilities to C++ programmers. After a minor C++14 update released in December 2014, various new additions were introduced in C++17. After becoming finalized in February 2020, a draft of the C++20 standard was approved on 4 September 2020, and officially published on 15 December 2020.

Wikipedia has a good coverage.

`https://en.wikipedia.org/wiki/C%2B%2B`

2.4 Fortran 2023

The following has been taken from the Fortran 2023 standard.

2.4.1 Source form:

The maximum length of a line in free form source has been increased. The maximum length of a statement

has been increased. The limit on the number of continuation lines has been removed.

2.4.2 Data declaration:

A data object with a coarray component can be an array or allocatable. BIND(C) ENUM are now referred to as interoperable enumerations, and noninteroperable enumeration types are available. An interoperable enumeration can be given a type name. TYPEOF and CLASSOF type specifiers can be used to declare one or more entities to have the same type and type parameters as another entity. A PUBLIC namelist group can have a PRIVATE namelist group object. The DIMENSION attribute can be declared with a syntax that does not depend on the rank (8.5.8, 8.5.17).

2.4.3 Data usage and computation:

Binary, octal, and hexadecimal literal constants can be used in additional contexts. A deferred-length allocatable errmsg-variable is allocated by the processor to the length of the explanatory message. An ALLOCATE statement can specify the bounds of an array allocation with array expressions. A pointer assignment statement can specify lower bounds or rank remapping with array expressions. Arrays can be used to specify multiple subscripts or subscript triplets (9.5.3.2). Conditional expressions provide selective evaluation of subexpressions.

2.4.4 Input/output:

The AT edit descriptor provides output of character values with trailing blanks trimmed. The LEADING_ZERO= specifier in the OPEN and WRITE statements, and the LZP, LZS and LZ control edit descriptors,provide control of optional leading zeros during formatted output. A deferred-length allocatable iomsgvariable is allocated by the processor to the length of the explanatory message. A deferred-length allocatable scalar io-unit in a WRITE statement is allocated by the processor to the length of the record to be written.

2.4.5 Execution control:

The REDUCE locality specifier for the DO CONCURRENT construct specifies reduction variables for theloop. The NOTIFY WAIT statement, NOTIFY= specifier on an image selector, and the NOTIFY_TYPE from the intrinsic module ISO_FORTRAN_ENV provide one-sided data-oriented synchronization between images.

2.4.6 Intrinsic procedures:

The intrinsic functions ACOSD, ASIND, ATAND, ATAN2D, COSD, SIND, and TAND are trigonometric functions in which angles are specified in degrees. The intrinsic functions ACOSPI, ASINPI, ATANPI, ATAN2PI, COSPI, SINPI, and TANPI are trigonometric functions in which angles are specified in halfrevolutions (that is, as multiples of p). The intrinsic function SELECTED_LOGICAL_KIND returns kind type parameter values for type logical. The intrinsic subroutine SPLIT parses a string into tokens, one at a time. The intrinsic subroutine SYSTEM_CLOCK supports more than one system clock for an image. The intrinsic subroutine TOKENIZE parses a string into tokens. When a deferred-length allocatable actual argument of an intrinsic procedure is assigned character data, it is allocated by the processor to the length of the data. Execution of a collective subroutine can be successful on an image even when an error condition occurs for the corresponding execution on another image.

2.4.7 Intrinsic modules:

Additional named constants LOGICAL8, LOGICAL16, LOGICAL32, LOGICAL64, and REAL16 have been added to the intrinsic module ISO_FORTRAN_ENV. The subroutines IEEE_GET_ROUNDING_MODE, IEEE_GET_UNDERFLOW_MODE, IEEE_SET_ROUNDING_MODE, and IEEE_SET_UNDERFLOW_MODE, from the intrinsic module IEEE_ARITHMETIC, are now considered to be pure and simple. The subroutines IEEE_GET_MODES, IEEE_GET_STATUS, IEEE_SET_MODES, and IEEE_SET_STATUS, from the intrinsic module IEEE_EXCEPTIONS, are now considered to be pure and simple. The procedures C_F_STRPOINTER and F_C_STRING have been added to the intrinsic module ISO_C_BINDING to assist in the use of null-terminated strings. The subroutine C_F_POINTER in the intrinsic module ISO_C_BINDING has an extra optional dummy argument, LOWER, that specifies the lower bounds for FPTR.

2.4.8 Changes to the intrinsic module IEEE_ARITHMETIC for conformance with ISO/IEC 60559:2020:

The new functions IEEE_MAX, IEEE_MAX_MAG, IEEE_MIN, and IEEE_MIN_MAG perform the operations maximum, maximumMagnitude, minimum, and miminumMagnitude in ISO/IEC 60559:2020. The functions IEEE_MAX_NUM, IEEE_MAX_NUM_MAG, IEEE_MIN_NUM, and IEEE_MIN_NUM_MAG now conform to the operations maximumNumber, maximumMagnitudeNumber, minimumNumber and minimumMagnitudeNumber in ISO/IEC 60559:2020; the changes affect the treatment of zeros and NaNs.

2.4.9 Program units and procedures:

A procedure can be specified to be a simple procedure; a simple procedure references or defines nonlocal variables only via its dummy arguments. Conditional arguments provide actual argument selection in a procedure reference.

2.4.10 John Reid paper on Fortran 2023 changes

The following has been taken from the John Reid paper on Fortran 2023 changes. The document includes the document numbering scheme used by the standards committee to identify where the proposals come from.

- Language elements
    - US 01 & 02. Allow much longer statement lines and overall statement length
    - US 14. Automatic allocation of lengths of character variables
    - US 16. The specifiers typeof and classof
    - US 22. Conditional expressions and arguments
    - US 23. More use of binary, octal, and hexadecimal constants
- Intrinsic procedures and intrinsic modules
    - US 03. Extracting tokens from a string
    - US 04. Trig functions that work in degrees
    - US 05. Trig functions that work with half revolutions
    - US 06. selected logical kind
    - Changes to system clock
    - Changes for conformance with new IEEE standard
    - US 07 & 08. Additional named constants to specify kinds
- Interoperability with C
    - UK 01. Extend the intrinsic procedure c f pointer to allow its pointer result to have specified lower bounds
    - US 09. Procedures for converting between Fortran and C strings
- Input-output
    - US 10. The at edit descriptor
    - US 11. Control over leading zeros in output of real values
    - Namelist
- Coarrays

  - US 12. Allow an object of a type with a coarray ultimate component to be an array or allocatable
  - US 13. Put with notify
  - Error conditions in collectives
- Procedures
  - US 15. Simple procedures
- Array features
  - US 17. Using integer arrays to specify subscripts and section subscripts
  - US 18. Using integer arrays to specify the rank and bounds of an array
  - Using an integer constant to specify rank
  - US 20. Reduction specifier for do concurrent
- US 21. Enumerations
  - Enumeration types
  - Enum types

3 Arithmetic

3.1 Chapter 5 example 18: using un-initialised variables

The Fortran standard has the following definitions

- data object - object, constant, variable, or subobject of a constant
- defined - data object has a valid value
- undefined - data object does not have a valid value

If a program does not provide an initial value (in a type statement) for a variable then its status is said to be undefined.

Consider the following example, which is a variation on example 2 from chapter 4.

```
program ch0518
!
! Updated version of
! ch0402
!
  implicit none
!
! defined    - data object - has a valid value
!
! undefined - data object - does not have a valid value
!

  real     :: n1
  real     :: n2
  real     :: n3
  real     :: average
  real     :: total
  integer :: n = 3

  print *,' Variables have not been assigned values'

  print *,n1
  print *,n2
  print *,n3
  print *,average
  print *,total

  n1       = 1
  n2       = 2
  n3       = 3

  total    = n1 + n2 + n3
  average = total / n

  print *, 'Total of numbers is ', total
  print *, 'Average of the numbers is ', average
```



```
end program
```

Variables n1, n2, n3, total and average all have undefined status. The use of variables with undefined status is processor dependent. Care must be taken when writing programs to ensure that your variables have a defined status wherever possible. We look at this topic in several subsequent sections.

3.2 Computer hardware and real and integer arithmetic in the 1970's and 1980's

We started working in computer services in the University of London in the 1970's. Here are some of the computer systems that were in use in the 70's and 80's.

3.2.1 CDC

These systems were available at Imperial College and the University of London Computer Centre.

The information is taken from

- Assembly Language Programming, Ralph Grishman, Algorithmics Press.

and

`https://en.wikipedia.org/wiki/CDC_6600`

Word size	60 bit
Integer	48 bit, one's complement
Real	60 bit, sign bit, 11 bit exponent, 48 bit mantissa
Double precision	120 bit, 96 bit mantissa

3.2.2 Cray

These systems were available at the University of London Computer Centre.
Information is taken from

`https://en.wikipedia.org/wiki/Cray-1`

Word size	64 bit
Integer	
Real	64 bit
Double precision	128 bit

3.2.3 ICL - 1900 series

Information is taken from

`https://en.wikipedia.org/wiki/ICT_1900_series`

Word size	24
Integer	Single length, 24 bit two's complement Multi-length, 24 bit first word, second and subsequent 23 bit
Real	two words holding a 24 bit mantissa and 9 bit exponent
Double precision	two words holding a 38 bit mantissa and 9 bit exponent
Additional precision	4 words holding a 75 bit mantissa and 9 bit exponent

3.2.4 IBM

Information is taken from

`https://en.wikipedia.org/wiki/IBM_System/360_architecture#Data_formats`

Word size	32
Integer	two's complement binary halfword or fullword values.
Real	32 bit
Double precision	64 bit
Additional precision	The 360/85 and 360/195 also support 128 bit extended precision floating point numbers

For all three formats, bit 0 is a sign and bits 0-7 are a characteristic (exponent, biased by 64). Bits 8-31 (8-63) are a hexadecimal fraction. For extended precision, the low order doubleword has its own sign and characteristic

3.2.5 DEC VAX

The information is taken from

`https://nssdc.gsfc.nasa.gov/nssdc/formats/VAXFloatingPoint.htm`

There are 4 floating point formats.

- F_floating point numbers have the range of approximately plus or minus 2.9E-39 to plus or minus 1.7E+38, with a precision of approximately seven decimal digits.
- D_floating point numbers have the range of approximately plus or minus 2.9E-39 to plus or minus 1.7E+38, with a precision of approximately 16 decimal digits.
- G_floating point numbers have the range of approximately plus or minus 5.6E-309 to plus or minus 0.9E+308, with a precision of approximately 15 decimal digits. The exponent has a bias of 1024 (not 128).
- H_floating point numbers have the range of approximately plus or minus 8.4E-4933 to plus or minus 5.9E+4931, with a precision of approximately 33 decimal digits. The exponent has a bias of 16384 (not 1024).

Word size	32 bits
Integer	32 bits
Real	See above
Double precision	See above

3.3 Chapter 5 example 19: Using the kind query functions and testing for 16 bit real support

The Fortran 90 standard introduced a variety of kind query functions. Here is a module that illustrates the use of the integer kind query functions.

```
module integer_kind_module
  implicit none
  integer, parameter :: i8    = selected_int_kind(2)
  integer, parameter :: i16  = selected_int_kind(4)
  integer, parameter :: i32  = selected_int_kind(9)
  integer, parameter :: i64  = selected_int_kind(15)
end module
```


Here is our current equivalent for real types.

```
module precision_module
  implicit none
!
! Updated with the release of NAG 7 which
! supports 16 bit reals.
!
! single, double, quad naming used by lapack.
! hence sp, dp, qp
!
! we have used hp as half precision
!
  integer, parameter :: hp = selected_real_kind( 3,   4)
  integer, parameter :: sp = selected_real_kind( 6,  37)
  integer, parameter :: dp = selected_real_kind(15, 307)
  integer, parameter :: qp = selected_real_kind(30, 291)
end module
```

3.4 Problems

Compile and run this example with the compilers you have access to.

4 Whole array and additional array features

The idea for this example came from a course given to the Met Office in May 2023.

Consider a 3 d cube.

The data in the front face of the cube is

1	2	3
4	5	6
7	8	9

and the data in the middle of the cube is

10	11	12
13	14	15
16	17	18

and the data in the back plane of the cube is

19	20	21
22	23	24
25	26	27

Here is a table illustrating some of the features of a 3 by 3 cube., where we have added the indices for each cube element.

Front	z=1	y	x	1	2	3
	data	1		1	2	3
	indices			1,1,1	2,1,1	3,1,1
	data	2		4	5	6
	indices			1,2,1	2,2,1	3,2,1
	data	3		7	8	9
	indices			1,3,1	2,3,1	3,3,1
Middle	z=2	y	x	1	2	3
	data	1		10	11	12
	indices			1,1,2	2,1,2	3,1,2
	data	2		13	14	15
	indices			1,2,2	2,2,2	3,2,2
	data	3		16	17	18

	indices			1,3,2	2,3,2	3,3,2
Back	z=3	y	x	1	2	3
	data	1		19	20	21
	indices			1,1,3	2,1,3	3,1,3
	data	2		22	23	24
	indices			1,2,3	2,2,3	3,2,3
	data	3		25	26	27
	indices			1,3,3	2,3,3	3,3,3

Given a 1 d array we can use reshape to populate the 3 d array.

4.1 Chapter 8 example 14: reshape and a 3 d array

Here is the source code

```
program ch0814
  implicit none
  integer , parameter :: nx=3
  integer , parameter :: ny=3
  integer , parameter :: nz=3

  integer               :: x,y,z
  integer               :: I

  integer , dimension(1:nx*ny*nz)      :: one_d = [
(i,i=1,nx*ny*nz) ]

  integer , dimension(1:nx,1:ny,1:nz) :: three_d=0
  character (20) , dimension(3)        :: cube_plane = (/ '
Front   ' ,&
                   ' Middle ' ,&
                   ' Back    ' /)

  print *,' '
  print *,' One dimension array order'
  print *,' '
  print *,one_d
  print *,'default'
  print *,' '
  three_d = reshape(one_d,(/nx,ny,nz/))
  do z=1,3
```

```
    print *,cube_plane(z)
    print *,' '
    do x=1,3
      print 10,three_d(x,1:ny,z)
      10 format(10x,3(1x,i2))
    end do
    print *,' '
  end do
  three_d = reshape(one_d,(/nx,ny,nz/),order=(/1,2,3/))
  print *,'1 * 2 * 3'
  print *,' '
  do z=1,3
    print *,cube_plane(z)
    print *,' '
    do x=1,3
      print 10,three_d(x,1:ny,z)
    end do
    print *,' '
  end do
  three_d = reshape(one_d,(/nx,ny,nz/),order=(/1,3,2/))
  print *,'1 * 3 * 2'
  print *,' '
  do z=1,3
    print *,cube_plane(z)
    print *,' '
    do x=1,3
      print 10,three_d(x,1:ny,z)
    end do
    print *,' '
  end do
  three_d = reshape(one_d,(/nx,ny,nz/),order=(/2,1,3/))
  print *,'2 * 1 * 3'
  print *,' '
  do z=1,3
    print *,cube_plane(z)
    print *,' '
    do x=1,3
      print 10,three_d(x,1:ny,z)
    end do
    print *,' '
  end do
  three_d = reshape(one_d,(/nx,ny,nz/),order=(/2,3,1/))
  print *,'2 * 3 * 1'
  print *,' '
  do z=1,3
    print *,cube_plane(z)
    print *,' '
    do x=1,3
```

```
      print 10,three_d(x,1:ny,z)
    end do
    print *,' '
  end do
  three_d = reshape(one_d,(/nx,ny,nz/),order=(/3,1,2/))
  print *,'3 * 1 * 2'
  print *,' '
  do z=1,3
    print *,cube_plane(z)
    print *,' '
    do x=1,3
      print 10,three_d(x,1:ny,z)
    end do
    print *,' '
  end do
  three_d = reshape(one_d,(/nx,ny,nz/),order=(/3,2,1/))
  print *,'3 * 2 * 1'
  print *,' '
  do z=1,3
    print *,cube_plane(z)
    print *,' '
    do x=1,3
      print 10,three_d(x,1:ny,z)
    end do
    print *,' '
  end do
end program ch0814
```

Here is the output.

```
  One dimension array order

 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26
27
 default

  Front

              1   4   7
              2   5   8
              3   6   9

  Middle

             10  13  16
             11  14  17
             12  15  18

  Back

             19  22  25
             20  23  26
```

```
           21 24 27

1 * 2 * 3

 Front

            1  4  7
            2  5  8
            3  6  9

 Middle

           10 13 16
           11 14 17
           12 15 18

 Back

           19 22 25
           20 23 26
           21 24 27

1 * 3 * 2

 Front

            1 10 19
            2 11 20
            3 12 21

 Middle

            4 13 22
            5 14 23
            6 15 24

 Back

            7 16 25
            8 17 26
            9 18 27

2 * 1 * 3

 Front

            1  2  3
            4  5  6
            7  8  9

 Middle

           10 11 12
           13 14 15
           16 17 18

 Back
```

```
              19 20 21
              22 23 24
              25 26 27

 2 * 3 * 1

  Front

               1  2  3
              10 11 12
              19 20 21

  Middle

               4  5  6
              13 14 15
              22 23 24

  Back

               7  8  9
              16 17 18
              25 26 27

 3 * 1 * 2

  Front

               1 10 19
               4 13 22
               7 16 25

  Middle

               2 11 20
               5 14 23
               8 17 26

  Back

               3 12 21
               6 15 24
               9 18 27

 3 * 2 * 1

  Front

               1  4  7
              10 13 16
              19 22 25

  Middle

               2  5  8
              11 14 17
              20 23 26
```

```
Back

              3   6   9
             12  15  18
             21  24  27
```

Here is a table summarising the output.

default	1 * 2 * 3	1 * 3 * 2	2 * 1 * 3	2 * 3 * 1	3 * 1 * 2	3 * 2 * 1
1 4 7	1 4 7	1 10 19	1 2 3	1 2 3	1 10 19	1 4 7
2 5 8	2 5 8	2 11 20	4 5 6	10 11 12	4 13 22	10 13 16
3 6 9	3 6 9	3 12 21	7 8 9	19 20 21	7 16 25	19 22 25
10 13 16	10 13 16	4 13 22	10 11 12	4 5 6	2 11 20	2 5 8
11 14 17	11 14 17	5 14 23	13 14 15	13 14 15	5 14 23	11 14 17
12 15 18	12 15 18	6 15 24	16 17 18	22 23 24	8 17 26	20 23 26
19 22 25	19 22 25	7 16 25	19 20 21	7 8 9	3 12 21	3 6 9
20 23 26	20 23 26	8 17 26	22 23 24	16 17 18	6 15 24	12 15 18
21 24 27	21 24 27	9 18 27	25 26 27	25 26 27	9 18 27	21 24 27

So there are 6 ways or permutations in filling the 3 d array from the 1 d array. The reshape intrinsic is a very powerful way of transferring data between arrays.

4.2 Problems

Compile and run this example.

5 Introduction to derived types

Initialisation using constructors was missing from earlier editions.

5.1 Chapter 17 example 5: Derived type constructor usage

Here is the source code

```
module date_module

   type date

      integer :: day = 1
      integer :: month = 1
      integer :: year = 2000

   end type

end module

program ch1705

   use date_module

   implicit none

! Initialisation via derived type definition

   type (date) :: d1

! Intialisation via default compiler
! provided constructor at
! declaration time

   type (date) :: d2=date(11,2,1952)

   print *, d1%day, d1%month, d1%year
   print *, d2%day, d2%month, d2%year

! Intialisation via default compiler
! provided constructor at
! run time

   d1=date(1,3,1956)

   print *, d1%day, d1%month, d1%year

end program
```


6 Introduction to pointers

6.1 Additional technical background

A pointer is a variable that has the pointer attribute. A pointer is associated with a target by allocation or pointer assignment. A pointer becomes associated as follows:

- The pointer is allocated as the result of the successful execution of an allocate statement referencing the pointer

or

- The pointer is pointer-assigned to a target that is associated or is specified with the target attribute and, if allocatable, is currently allocated.

A pointer may have a pointer association status of

- associated
- disassociated
- undefined

Its association status may change during execution of a program. Unless a pointer is initialised (explicitly or by default), it has an initial association status of undefined. A pointer may be initialised to have an association status of disassociated.

A pointer shall neither be referenced nor defined until it is associated. A pointer is disassociated following execution of a deallocate or nullify statement, following pointer association with a disassociated pointer, or initially through pointer initialisation.

Examples 1 through 6 highlights some of these issues.

6.2 New examples

There are six new examples in this chapter that use the c_loc function from the C interop facilities of Fortran. You can now see what is happening behind the scenes with examples 1 to 6 with your compiler.

6.3 Chapter 18 example 8: duplicate of example 1 using c_loc to show memory usage

Here is the source code.

```
include 'integer_kind_module.f90'

program ch1807

   use iso_c_binding
   use integer_kind_module

   implicit none
   type     (c_ptr)   :: x
   integer (i64)      :: x_address
   integer, pointer :: a => null(), b => null()
   integer, target   :: c
   integer, target   :: d

   c = 1
```

```
  a => c
  c = 2
  b => c
  d = a + b
  print *, a, b, c, d

  x = c_loc(a)
  x_address = transfer(x,x_address)
  print *,x_address
  x = c_loc(b)
  x_address = transfer(x,x_address)
  print *,x_address
  x = c_loc(c)
  x_address = transfer(x,x_address)
  print *,x_address
  x = clack(d)
  x_address = transfer(x,x_address)
  print *,x_address

end program
```

Here is some sample output from the NAG compiler under Windows.

```
 2 2 2 4
 4219008
 4219008
 4219008
 4219032
```

6.4 Chapter 18 example 9: duplicate of example 2 using c_loc to show memory usage

Here is the source code.

```
include 'integer_kind_module.f90'

program ch1808


  use iso_c_binding
  use integer_kind_module

  implicit none
  type     (c_ptr)   :: x
  integer (i64)      :: x_address
  integer, pointer :: a => null(), b => null()
  integer, target  :: c
  integer, target  :: d

  x = c_loc(a)
  x_address = transfer(x,x_address)
  print *,x_address
```

```
  x = c_loc(b)
  x_address = transfer(x,x_address)
  print *,x_address
  x = c_loc(c)
  x_address = transfer(x,x_address)
  print *,x_address
  x = c_loc(d)
  x_address = transfer(x,x_address)
  print *,x_address

  print *, associated(a)
  print *, associated(b)
  c = 1
  a => c
  c = 2
  b => c
  d = a + b
  print *, a, b, c, d
  print *, associated(a)
  print *, associated(b)

  x = c_loc(a)
  x_address = transfer(x,x_address)
  print *,x_address
  x = c_loc(b)
  x_address = transfer(x,x_address)
  print *,x_address
  x = c_loc(c)
  x_address = transfer(x,x_address)
  print *,x_address
  x = c_loc(d)
  x_address = transfer(x,x_address)
  print *,x_address
end program
```

Here is some sample output from the NAG compiler under Windows.

```
 0
 0
 4219856
 4219860
 F
 F
 2 2 2 4
 T
 T
 4219856
 4219856
 4219856
 4219860
```

6.5 Chapter 18 example 10: duplicate of example 3 using c_loc to show memory usage

Here is the source code.

```
include 'integer_kind_module.f90'

program ch1809

  use iso_c_binding
  use integer_kind_module

  implicit none
  type     (c_ptr)   :: x
  integer (i64)      :: x_address
  integer, pointer :: a => null(), b => null()
  integer, target  :: c
  integer, target  :: d

  x = c_loc(a)
  x_address = transfer(x,x_address)
  print *,x_address
  x = c_loc(b)
  x_address = transfer(x,x_address)
  print *,x_address
  x = c_loc(c)
  x_address = transfer(x,x_address)
  print *,x_address
  x = c_loc(d)
  x_address = transfer(x,x_address)
  print *,x_address

  print *, a
  print *, b

  c = 1
  a => c
  c = 2
  b => c
  d = a + b
  print *, a, b, c, d

end program
```

Here is some sample output from the NAG compiler under Windows with the -C=all flag.

```
 0
 0
 4219856
 4219860
Runtime Error: ch1809.f90, line 28: Reference to disassociated POINTER A
Program terminated by fatal error
```


6.6 Chapter 18 example 11: duplicate of example 4 using c_loc to show memory usage

Here is the source code.

```
include 'integer_kind_module.f90'

program ch1810

  use iso_c_binding
  use integer_kind_module

  implicit none
  type     (c_ptr)   :: x
  integer (i64)      :: x_address
  integer, pointer :: a => null(), b => null()
  integer, target   :: c
  integer, target   :: d

  x = c_loc(a)
  x_address = transfer(x,x_address)
  print *,x_address

  allocate (a)

  x = c_loc(a)
  x_address = transfer(x,x_address)
  print *,x_address

  a = 1
  c = 2
  b => c
  d = a + b
  print *, a, b, c, d
  deallocate (a)

end program
```

Here is some sample output from the NAG compiler under Windows.

```
 0
 141819904
 1  2  2  3
```

6.7 Chapter 18 example 12: duplicate of example 5 using c_loc to show memory usage

Here is the source code.

```
include 'integer_kind_module.f90'

program ch1811
```

```
  use iso_c_binding
  use integer_kind_module

  implicit none
  type     (c_ptr)   :: x
  integer (i64)      :: x_address
  integer, pointer :: a => null(), b => null()
  integer, target   :: c
  integer, target   :: d

  allocate (a)
  allocate (b)

  x = c_loc(a)
  x_address = transfer(x,x_address)
  print *,x_address
  x = c_loc(b)
  x_address = transfer(x,x_address)
  print *,x_address

  a = 100
  b = 200

  print *, a, b
  c = 1
  a => c
  c = 2
  b => c
  d = a + b
  print *, a, b, c, d

  x = c_loc(a)
  x_address = transfer(x,x_address)
  print *,x_address
  x = c_loc(b)
  x_address = transfer(x,x_address)
  print *,x_address
  x = c_loc(c)
  x_address = transfer(x,x_address)
  print *,x_address
  x = c_loc(d)
  x_address = transfer(x,x_address)
  print *,x_address

end program
```

Here is some sample output from the NAG compiler under Windows.

```
 141819904
 141819920
```



```
100 200
2 2 2 4
4219848
4219848
4219848
4219852
```

6.8 Chapter 18 example 13: duplicate of example 6 using c_loc to show memory usage

Not available at this time.

6.9 Chapter 18 example 14: examples of where in memory compilers allocate arrays

This example looks at where in memory compilers allocate arrays.

Here is the program source.

```
!
! Example to show array memory allocation
! using a range of compilers.
! We have several types of array
!
! 1.0 main program array
!
! 1.1 dynamic allocation in the main program
!
! 2.0 automatic allocation in a subroutine
!
! 3.0 dynamic allocation in a subroutime
!

include 'integer_kind_module.f90'

program ch1811

  use iso_c_binding
  use integer_kind_module

  implicit none

  integer , parameter                                :: n =
1024 * 1024
  integer                                               :: i
  integer , dimension(n) , target                    :: y
  integer , dimension(:) , allocatable , target :: z

  type     (c_ptr)   :: x
  integer (i64)      :: address_as_integer

  do i=1,n
    y(i)=i
```

```
   end do

   x = c_loc(y)
   address_as_integer = transfer(x,address_as_integer)
   print 10,address_as_integer
   10 format(' Main program normal array       ',i20)

   allocate(z(n))

   z=y

   x = c_loc(z)
   address_as_integer = transfer(x,address_as_integer)
   print 20,address_as_integer
   20 format(' Main program allocatable array ',i20)

   call automatic_array(n)

   call allocatable_array(n)

end program

subroutine automatic_array(n)

   use iso_c_binding
   use integer_kind_module

   implicit none

   integer , intent(in) :: n
   integer , dimension(n) , target :: z
   integer :: i

   type     (c_ptr)   :: x
   integer (i64)      :: address_as_integer

   do i=1,n
     z(i)=i
   end do

   x = c_loc(z)
   address_as_integer = transfer(x,address_as_integer)
   print 10,address_as_integer
   10 format(' Subroutine automatic array      ',i20)

end subroutine

subroutine allocatable_array(n)
```



```
   use iso_c_binding
   use integer_kind_module

   implicit none

   integer , intent(in) :: n
   integer , dimension(:) , allocatable , target :: z
   integer :: i

   type     (c_ptr)   :: x
   integer (i64)      :: address_as_integer

   allocate(z(n))

   do i=1,n
     z(i)=i
   end do

   x = c_loc(z)
   address_as_integer = transfer(x,address_as_integer)
   print 10,address_as_integer
   10 format(' Subroutine allocatable array    ',i20)

end subroutine
```

Here are some of the results

Nag compiler Windows

```
Main program normal array                          4227392
Main program allocatable array                   150405120
Subroutine automatic array                       154599424
Subroutine allocatable array                     158793728
```

Nag compiler linux

```
Main program normal array                   94570468184928
Main program allocatable array             140053911683072
Subroutine automatic array                 140053907488768
Subroutine allocatable array               140053903294464
```

Intel compiler Windows

```
Main program normal array                  140699484659136
Main program allocatable array               1988156821584
Subroutine automatic array                   1988161052752
Subroutine allocatable array                 1988161073232
```

On this platform you need the /heap-arrays compiler option.

Intel compiler linux

```
Main program normal array                          4794208
Main program allocatable array             139866881916960
Subroutine automatic array                 140736758817280
Subroutine allocatable array               139866877657120
```

gfortran compiler Windows

NA at this time.

gfortran compiler linux

```
Main program normal array                          4210848
Main program allocatable array             140269146931216
Subroutine automatic array                 140269142671376
Subroutine allocatable array                      37194496
```

nvidia compiler linux

```
Main program normal array                          4211232
Main program allocatable array             139971260514336
Subroutine automatic array                 139971256254496
Subroutine allocatable array                      23164480
```


6.10 Problems

Compile and run these examples and examine the output with your compiler.

7 Data structuring in Fortran

Under certain circumstances it is possible to replace the use of pointers with allocatable components. Garbage collection is now automatic.

7.1 Chapter 22 example 8: Rewrite of example 1 to use allocatable components rather than pointers

Here is the source code.

```
module character_list_module
  type character_list
    character (len=1) :: x
    type (character_list), allocatable :: next
  end type
end module

program ch2208

  use character_list_module
  implicit none

  character (len=80) :: fname
  integer :: io_stat_number = 0

  character                                                   :: x
  type (character_list) , allocatable , target :: list
  type (character_list) , pointer                    :: current
=>null()
  type (character_list) , pointer                    :: root
=>null()

  integer :: I = 0, n
  character (len=:), allocatable :: string

  fname='ch2208.f90'
  open ( unit=1 , file=fname , status='old' )

  do

    read (unit=1 , fmt='(a)' , advance='no' ,
iostat=io_stat_number) x

    if ( io_stat_number /= -1 ) then

      if (associated(current)) then
        allocate ( current%next , source = charac-
ter_list(x) )
        current => current%next
        i=i+1
```



```
        else if ( .not.associated(current) ) then
          ! First data item, need to anchor the root
          allocate ( list          , source = charac-
ter_list(x) )
          current => list
          root     => list
          I = I + 1
        end if

      else

        exit

      endif

    end do

    print *, I, ' characters read'

    n = I
    allocate (character(len=n) :: string)
    current => root
    do i=1,n
      string(i:i) =  current%x
      current      => current%next
    end do
    print *, 'data read was:'
    print 100, string
    100 format(a)

end program
```

7.2 Chapter 22 example 9: Rewrite of example 2 to use allocatable components rather than pointers

Here is the source code.

```
module real_list_module
  type real_list
    real :: x
    type (real_list), allocatable :: next
  end type
end module

program ch2209

  use real_list_module
  implicit none
```

```
  character (len=80) :: fname
  integer :: io_stat_number = 0

  real                                                   :: x
  type (real_list) , allocatable , target       :: list
  type (real_list) , pointer                      :: current
=>null()
  type (real_list) , pointer                      :: root
=>null()

  integer :: I = 0, n
  real , allocatable ,  dimension(:) :: y

  fname='ch2209.txt'
  open ( unit=1 , file=fname , status='old' )

  do

    read (unit=1 , fmt=*      ,
iostat=io_stat_number) x

    if ( io_stat_number /= -1 ) then

      if (associated(current)) then
        allocate ( current%next , source = real_list(x) )
        current => current%next
        i=i+1
      else if ( .not.associated(current) ) then
        ! First data item, need to anchor the root
        allocate ( list          , source = real_list(x) )
        current => list
        root     => list
        I = I + 1
      end if

    else

      exit

    endif

  end do

  print *, I, ' numbers read'

  n = I
  allocate (y(n))
```

```
  current => root
  do i=1,n
    y(I)     = current%x
    current => current%next
  end do
  print *, 'data read was:'
  print *,y

end program
```

7.3 Chapter 22 example 10: Linked lists using move_alloc rather than pointers

Here is the source code.

```
module character_linked_list_module

  type character_linked_list
    character (len=1) :: c
    type (character_linked_list), allocatable :: next
  end type character_linked_list

contains

  subroutine add_item_to_list(list,new_character)

    type (character_linked_list) , allocatable :: list
    character , intent(in)                          ::
new_character

    type (character_linked_list) , allocatable :: t

    call move_alloc(list,t)
    allocate(list,source=character_linked_list(new_charac-
ter))
    call move_alloc(t,list%next)

  end subroutine add_item_to_list

  function return_string(list,n)

    type (character_linked_list) , allocatable :: list
    integer , intent(in)                            :: n

    character (len=n)                               :: re-
turn_string

    type (character_linked_list) , allocatable :: t
```

```
    integer                                          :: I

    do i=1,n

      return_string(n-i+1:n-I+1)  = list%c
      call move_alloc(list%next,t)
      call move_alloc(t,list)

    end do

  end function return_string

end module character_linked_list_module

program ch2210

  integer :: z
  character (len=:), allocatable              :: string

  print *,' Calling subroutine to read the data'
  print *,' '

  call read_data()

  print *,' '
  print *,' Returned from subroutine'
  print *,' Automatic deallocation of data structures'
  print *,' '

  print *, 'data read was:'
  print 100, string
  100 format(a)

contains

subroutine read_data()

  use character_linked_list_module
  implicit none

  character (len=80) :: fname
  integer :: io_stat_number = 0

  character                                    :: x
  type (character_linked_list) , allocatable :: list

  integer                                      :: I = 0,
n
```

```
   fname='ch2210.f90'
   open (unit=1, file=fname, status='old')

   do

     read (unit=1, fmt='(a)', advance='no',
iostat=io_stat_number) x

     if ( io_stat_number /= -1 ) then

       call add_item_to_list(list,x)
       i=I+1

     else

       exit

     endif

   end do

   print *, I, ' characters read'
   n = I
   allocate (character(len=n) :: string)

   string = return_string(list,n)

end subroutine

end program
```

8 Generic programming

There are a small number of additional examples. The idea for the <vector> example came from some people from the UK Met Office attending a Fortran course in June 2022. I added the <array> example for completeness.

8.1 Chapter 25 example 3: Generic statistics module with 16 bit real support - Nag

This is a variation on example 2. Here is the main program source.

```
include 'precision_module_16_bit_support.f90'
include 'integer_kind_module.f90'
include 'statistics_module_16_bit_support.f90'
include 'timing_module_16_bit_support.f90'

program ch2503

   use iso_fortran_env
   use precision_module_16_bit_support
   use statistics_module_16_bit_support
   use timing_module_16_bit_support

   implicit none
   integer :: n
   integer :: i
   integer :: repeat_count = 4
   real (hp), allocatable, dimension (:) :: w
   real (hp) :: w_m, w_sd, w_median
   real (sp), allocatable, dimension (:) :: x
   real (sp) :: x_m, x_sd, x_median
   real (dp), allocatable, dimension (:) :: y
   real (dp) :: y_m, y_sd, y_median
   real (qp), allocatable, dimension (:) :: z
   real (qp) :: z_m, z_sd, z_median
   character *20, dimension (3) :: heading = [ '  Allocate
', '  Random       ', '  Statistics   ' ]

   print *,''
   print*,compiler_version()
   print *,''
   call start_timing()
   n =  10

   do i=1,repeat_count

   print *, '
n = ', n
   print *,''
   print *, ' Half precision'
```

```
  print *, ''

  allocate (w(1:n))
  print 100, heading(1), time_difference()
  call random_number(w)
  print 100, heading(2), time_difference()
  call calculate_statistics(w, n, w_m, w_sd, w_median)
  print 100, heading(3), time_difference()
  write (unit=*, fmt=110) w_m
  write (unit=*, fmt=120) w_sd
  write (unit=*, fmt=130) w_median
  deallocate (w)

  print *,''
  print *, ' Single precision'
  print *,''

  allocate (x(1:n))
  print 100, heading(1), time_difference()
100 format (a20, 6x, f18.6)
  call random_number(x)
  print 100, heading(2), time_difference()
  call calculate_statistics(x, n, x_m, x_sd, x_median)
  print 100, heading(3), time_difference()
  write (unit=*, fmt=110) x_m
110 format (' Mean                  = ', f10.6)
  write (unit=*, fmt=120) x_sd
120 format (' Standard deviation = ', f10.6)
  write (unit=*, fmt=130) x_median
130 format (' Median                = ', f10.6)
  deallocate (x)

  print *,''
  print *, ' Double precision'
  print *,''

  allocate (y(1:n))
  print 100, heading(1), time_difference()
  call random_number(y)
  print 100, heading(2), time_difference()
  call calculate_statistics(y, n, y_m, y_sd, y_median)
  print 100, heading(3), time_difference()
  write (unit=*, fmt=110) y_m
  write (unit=*, fmt=120) y_sd
  write (unit=*, fmt=130) y_median
  deallocate (y)

  print *,''
```



```
   print *, ' Quad precision'
   print *,''

   allocate (z(1:n))
   print 100, heading(1), time_difference()
   call random_number(z)
   print 100, heading(2), time_difference()
   call calculate_statistics(z, n, z_m, z_sd, z_median)
   print 100, heading(3), time_difference()
   write (unit=*, fmt=110) z_m
   write (unit=*, fmt=120) z_sd
   write (unit=*, fmt=130) z_median
   deallocate (z)

   n = n * 10

   end do

   call end_timing()

end program
```

Note that we have new versions of the following modules due to the support for 16 bit reals.

- precision_module_16_bit_support.f90
- statistics_module_16_bit_support.f90
- timing_module_16_bit_support.f90

8.2 Chapter 25 example 4: **Generic statistics module with 80 bit real support - gfortran**

This is a variation on example 2. Here is the main program source.

```
include 'precision_module_80_bit_support.f90'
include 'integer_kind_module.f90'
include 'statistics_module_80_bit_support.f90'
include 'timing_module_80_bit_support.f90'

program ch2504

   use precision_module_80_bit_support
   use statistics_module_80_bit_support
   use timing_module_80_bit_support

   implicit none
   integer :: n

   real (sp), allocatable, dimension (:) :: x
```

```
   real (sp) :: x_m, x_sd, x_median

   real (dp), allocatable, dimension (:) :: y
   real (dp) :: y_m, y_sd, y_median

   real (r80), allocatable, dimension (:) :: w
   real (r80) :: w_m, w_sd, w_median

   real (qp), allocatable, dimension (:) :: z
   real (qp) :: z_m, z_sd, z_median

   character *20, dimension (3) :: heading = [ '  Allocate
', '  Random       ', '  Statistics  ' ]

   call start_timing()
   n =  50000000
   print *, ' n = ', n

   print *, ' Single precision'

   allocate (x(1:n))
   print 100, heading(1), time_difference()
100 format (a20, 6x, f18.6)
   call random_number(x)
   print 100, heading(2), time_difference()
   call calculate_statistics(x, n, x_m, x_sd, x_median)
   print 100, heading(3), time_difference()
   write (unit=*, fmt=110) x_m
110 format (' Mean                 = ', f10.6)
   write (unit=*, fmt=120) x_sd
120 format (' Standard deviation = ', f10.6)
   write (unit=*, fmt=130) x_median
130 format (' Median               = ', f10.6)
   deallocate (x)

   print *, ' Double precision'

   allocate (y(1:n))
   print 100, heading(1), time_difference()
   call random_number(y)
   print 100, heading(2), time_difference()
   call calculate_statistics(y, n, y_m, y_sd, y_median)
   print 100, heading(3), time_difference()
   write (unit=*, fmt=110) y_m
   write (unit=*, fmt=120) y_sd
   write (unit=*, fmt=130) y_median
   deallocate (y)
```

```
  print *, ' gfortran 80 bit'

  allocate (w(1:n))
  print 100, heading(1), time_difference()
  call random_number(w)
  print 100, heading(2), time_difference()
  call calculate_statistics(w, n, w_m, w_sd, w_median)
  print 100, heading(3), time_difference()
  write (unit=*, fmt=110) w_m
  write (unit=*, fmt=120) w_sd
  write (unit=*, fmt=130) w_median
  deallocate (w)

  print *, ' Quad precision'

  allocate (z(1:n))
  print 100, heading(1), time_difference()
  call random_number(z)
  print 100, heading(2), time_difference()
  call calculate_statistics(z, n, z_m, z_sd, z_median)
  print 100, heading(3), time_difference()
  write (unit=*, fmt=110) z_m
  write (unit=*, fmt=120) z_sd
  write (unit=*, fmt=130) z_median
  deallocate (z)

  call end_timing()

end program
```

Note that we have new versions of the following modules due to the support for 80 bit reals.

- precision_module_80_bit_support.f90
- statistics_module_80_bit_support.f90
- timing_module_80_bit_support.f90

Here is some sample output.

```
ch2504_gfortran.exe
 2023/12/11  9:33:45 607
  n =      50000000
  Single precision
  Allocate                                 0.001462
  Random                                    0.104088
  Statistics                               0.704748
 Mean                  =    0.335544
 Standard deviation =    0.442733
 Median                =    0.500040
  Double precision
```

```
  Allocate                                  0.008252
  Random                                     0.220561
  Statistics                                0.719511
 Mean                 =    0.499951
 Standard deviation =    0.288647
 Median               =    0.499928
  gfortran 80 bit
  Allocate                                  0.013232
  Random                                     0.282966
  Statistics                                1.096770
 Mean                 =    0.499954
 Standard deviation =    0.288672
 Median               =    0.499878
  Quad precision
  Allocate                                  0.023946
  Random                                     3.121136
  Statistics                                4.065059
 Mean                 =    0.500024
 Standard deviation =    0.288704
 Median               =    0.499997
 2023/12/11  9:33:55 992
 Total time =                             10.383900
```

8.3 Problems

Compile and run these examples if you have access to the Nag and gfortran compilers.

9 C Interop

There are a small number of additional examples. The idea for the <vector> example came from some people from the UK Met Office attending a Fortran course in June 2022. I added the <array> example for completeness.

9.1 Chapter 35 example 15: passing a one d <vector> from C++ to Fortran

This is a variation on example 7.

Here is the Fortran source. It is the same as the original example 7.

```
function summation(x, n) bind (c, name='summation')
  use iso_c_binding
  implicit none
  integer (c_int), value :: n
  real (c_float), dimension (1:n), intent (in) :: x
  real (c_float) :: summation
  integer :: I

  summation = sum(x(1:n))
end function
```

Here is the new C++ source.

```
#include <iostream>
#include <vector>
using namespace std;
extern "C" float summation(float *,int );
int main()
{
  const int n=10;
  vector<float> x(n);
  int i;
  for (i=0;i<n;i++)
    x[i]=1.0f;
  cout << " C++ calling Fortran" << endl;
  cout << " 1 d vector as parameter" << endl;
  cout << " Sum is " << summation(&x[0],n) << endl;
  return(0);
}
```

Please see the batch files and shell scripts on how to compile these programs.

9.2 Chapter 35 example 16: passing a 1 d <array> between C++ and Fortran

The Fortran source is the same as in the previous 2 one d examples.

Here is the C++ source.

```
#include <iostream>
#include <array>
using namespace std;
```

```
extern "C" float summation(float *,int );
int main()
{
  const int n=10;
  array<float,n> x;
  int u;
  for (i=0;i<n;i++)
    x[i]=1.0f;
  cout << " C++ calling Fortran" << endl;
  cout << " 1 d array as parameter" << endl;
  cout << " Sum is " << summation(&x[0],n) << endl;
  return(0);
}
```

Compilation notes

10 IEEE arithmetic

There are a small number of additional examples.

10.1 Chapter 36 example 7: inexact summation with compiler version

This is a variation on example 5. We added details about which the compiler version.

Here is the new source code.

```
program ch3607

  use ieee_arithmetic
  use iso_fortran_env
  implicit none

  integer :: I
  real :: computed_sum
  real :: real_sum
  integer :: array_size

  logical :: inexact_happened = .false.
  integer :: allocate_status

  character *13, dimension (3) :: heading = (/ '
10,000,000', '  100,000,000', '1,000,000,000' /)

  real, allocatable, dimension (:) :: x

  print *,compiler_version()

  if (ieee_support_datatype(x)) then
    print *, ' IEEE support for default precision'
  end if

! 10,000,000

  array_size = 10000000

  do I = 1, 3
    write (unit=*, fmt=100) array_size, heading(I)
100 format (' Array size = ', i15, 2x, a13)
    allocate (x(1:array_size), stat=allocate_status)
    if (allocate_status/=0) then
      print *, ' Allocate fails, program ends'
      stop
    end if
    x = 1.0
    computed_sum = sum(x)
    call ieee_get_flag(ieee_inexact, inexact_happened)
    real_sum = array_size*1.0
```

```
    write (unit=*, fmt=110) computed_sum
110 format (' Computed sum = ', e12.4)
    write (unit=*, fmt=120) real_sum
120 format (' Real sum     = ', e12.4)
    if (inexact_happened) then
      print *, ' inexact arithmetic'
      print *, ' in the summation'
      print *, ' program terminates'
      stop 20
    end if
    deallocate (x)
    array_size = array_size*10
  end do

end program
```

Example ch0510.f90 has been updated to include information on the IEEE 16 bit real support offered by the Nag compiler.

Here is the source code.

```
program ch0510
  implicit none
!
! real arithmetic
!
! 16 bit reals are in the latest IEEE standard.
! we have added tests for that type in this
! program.
!
! 32 and 64 bit reals are normally available.
! The IEEE format is as described below.
!
! 32 bit reals  8 bit exponent, 24 bit mantissa
! 64 bit reals 11 bit exponent, 53 bit mantissa
!
! 128 bit reals and decimal are also in the
! latest IEEE standard.
! We have chosen a portable specification
! for 128 bit reals as Nag use their own.
!

!  integer, parameter :: hp = 16
  integer, parameter :: hp = selected_real_kind( 3,   4)
  integer, parameter :: sp = selected_real_kind( 6,  37)
  integer, parameter :: dp = selected_real_kind(15, 307)
  integer, parameter :: qp = selected_real_kind(30, 291)

  real (hp) :: rhp
  real (sp) :: rsp
```

```
   real (dp) :: rdp
   real (qp) :: rqp

   print *, '            ====================='
   print *, '            Real kind information'
   print *, '            ====================='
   print *, ' kind number'
   print *, '     ', kind(rhp), ' ', kind(rsp), ' ',
kind(rdp), ' ', kind(rqp)
   print *, ' digits details'
   print *, '     ', digits(rhp), ' ', digits(rsp), ' ', dig-
its(rdp), ' ', digits(rqp)
   print *, ' epsilon details'
   print *, '     ', epsilon(rhp)
   print *, '     ', epsilon(rsp)
   print *, '     ', epsilon(rdp)
   print *, '     ', epsilon(rqp)
   print *, ' huge value'
   print *, '     ', huge(rhp)
   print *, '     ', huge(rsp)
   print *, '     ', huge(rdp)
   print *, '     ', huge(rqp)
   print *, ' maxexponent value'
   print *, '     ', maxexponent(rhp)
   print *, '     ', maxexponent(rsp)
   print *, '     ', maxexponent(rdp)
   print *, '     ', maxexponent(rqp)
   print *, ' minexponent value'
   print *, '     ', minexponent(rhp)
   print *, '     ', minexponent(rsp)
   print *, '     ', minexponent(rdp)
   print *, '     ', minexponent(rqp)
   print *, ' precision details'
   print *, '     ', precision(rhp), ' ', precision(rsp), '
', precision(rdp), ' ', precision(rqp)
   print *, ' radix details'
   print *, '     ', radix(rhp), ' ', radix(rsp), ' ', ra-
dix(rdp), ' ', radix(rqp)
   print *, ' range details'
   print *, '     ', range(rhp), ' ', range(rsp), ' ',
range(rdp), ' ', range(rqp)
   print *, ' tiny details'
   print *, '     ', tiny(rhp)
   print *, '     ', tiny(rsp)
   print *, '     ', tiny(rdp)
   print *, '     ', tiny(rqp)
end program
```

Here is the output.

```
            ====================
            Real kind information
            ====================
 kind number
      16    1    2    3
 digits details
      11    24    53    106
 epsilon details
         9.7656E-04
         1.1920929E-07
         2.220446049250313IE-16
      2.46519032881566189191165177E-32
 huge value
        65504.
         3.4028235E+38
        1.797693134862315 7E+308
      8.98846567431157953864652595E+307
 maxexponent value
      16
      128
      1024
      1023
 minexponent value
      -13
      -125
      -1021
      -968
 precision details
      3    6    15    31
 radix details
      2    2    2    2
 range details
      4    37    307    291
 tiny details
         6.1035E-05
         1.1754944E-38
        2.2250738585072014E-308
      2.00416836000897277799610805E-292
```

This means between 6 and 9 digits of precision for default reals in Fortran, which correspond to the IEEE 32 bit real data type.

Here is the output from the NAG compiler from running ch3607.f90.

```
NAG Fortran Compiler Release 7.0(Yurakucho) Build 7017
 IEEE support for default precision
Array size =          10000000      10,000,000
Computed sum =    0.1000E+08
Real sum      =    0.1000E+08
Array size =         100000000     100,000,000
Computed sum =    0.1678E+08
Real sum      =    0.1000E+09
 inexact arithmetic
 in the summation
 program terminates
```

Here is the output from the Intel compiler from running example ch3607.


```
 Intel(R) Fortran Intel(R) 64 Compiler Classic for applications running
on Intel
 (R) 64, Version 2021.5.0 Build 20211109_000000
  IEEE support for default precision
 Array size =          10000000      10,000,000
 Computed sum =    0.1000E+08
 Real sum      =    0.1000E+08
 Array size =         100000000     100,000,000
 Computed sum =    0.1000E+09
 Real sum      =    0.1000E+09
  inexact arithmetic
  in the summation
  program terminates
```

In the Intel example the computed sum matches the exact sum!

10.2 Equivalent versions of ch3607 in C, C++, C# and Java

Note that this behaviour for 32 bit arithmetic is the same with other programming languages. Examples are available below in C, C++, C# and Java.

10.2.1 C version

```
include <stdio.h>
#include <stdlib.h>
#include <string.h>

float calculate_sum(int n)
{
  float *x;

  int i;
  float t;
  t=0.0;

  x = (float*) calloc ( n , sizeof(float) );

  for(i=0;i<n;i++)
  {
    x[i]=1;
    t = t + x[i];
  }

  free(x);
  return(t);
}

int main()
{

  int     I;
  int     j;
  float   computed_sum;
```

```
  float   actual_sum;
  int     array_size;

  char heading[3][15] = { "    10,000,000", "   100,000,000",
"1,000,000,000" };

/*

  Initial array size
  10,000,000

*/

  array_size = 10000000;

  for ( I=0 ; i<3 ; I++ )
  {
    printf(" Array size " );
    printf(" %s ",heading[i]);
    printf("\n");

    computed_sum = calculate_sum(array_size);
    actual_sum    = array_size*1.0;

    printf("    Computed sum %12.1f \n" , computed_sum);
    printf("    Actual sum    %12.1f \n" , actual_sum);

    if (actual_sum != computed_sum)
    {
      printf(" C \n");
      printf(" Accuracy limit of IEEE 32 bit floating point
arithmetic \n");
      printf(" program terminates \n") ;
      return(1);
    }
    array_size = array_size * 10;
  }
  return(0);
}
```

10.2.2 C++ version

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

float calculate_sum(int n)
{
```



```
  vector<float> x(n);
  int i;
  float t;

  t=0.0;

  for(i=0;i<n;i++)
  {
    x[i]=1;
    t = t + x[i];
  }

  return(t);
}

int main()
{

  int     I;
  int     j;
  float   computed_sum;
  float   actual_sum;
  int     array_size;

  string heading[3] = { "    10,000,000", "   100,000,000",
"1,000,000,000" };

/*

  Initial array size
  10,000,000

*/

  array_size = 10000000;

  for ( I=0 ; i<3 ; I++ )
  {

    cout << " Array size " << heading[i] << endl;

    computed_sum = calculate_sum(array_size);
    actual_sum    = array_size*1.0;

    cout << "    Computed sum " ;

    cout.width(12);
    cout.precision(1);
```

```
    cout.setf(ios::right);
    cout.setf(ios::showpoint);
    cout.setf(ios::fixed);

    cout << computed_sum << endl;

    cout << "   Actual sum   " ;

    cout.width(12);
    cout.precision(1);
    cout.setf(ios::right);
    cout.setf(ios::showpoint);
    cout.setf(ios::fixed);

    cout << actual_sum    << endl;

    if (actual_sum != computed_sum)
    {
      cout << " C++ " << endl;
      cout << " Accuracy limit of IEEE 32 bit floating
point arithmetic " << endl;
      cout << " program terminates " << endl;
      return(1);
    }
    array_size = array_size * 10;
  }

  return(0);

}
```

10.2.3 C# version

```
using System;

class ch3607
{

  static float calculate_sum(int n)
  {
    float [] x = new float [n];
    int i;
    float t;

    t=0;

    for(i=0;i<n;i++)
    {
      x[i]=1;
```

```
        t = t + x[i];
      }

      return(t);
    }

    static int Main()
    {

      int     I;
      float   computed_sum;
      float   actual_sum;
      int     array_size;

      string [] heading = { "    10,000,000", "   100,000,000",
"1,000,000,000" };

      /*

      Initial array size
      10,000,000

      */

      array_size = 10000000;

      for ( I=0 ; i<3 ; I++ )
      {

        Console.WriteLine( " Array size {0} " , heading[i] );

        computed_sum = calculate_sum(array_size);
        actual_sum    = array_size*1;

        Console.Write( "    Computed sum " );

        Console.WriteLine( computed_sum );

        Console.Write( "    Actual sum     " );

        Console.WriteLine( actual_sum );

        if (actual_sum != computed_sum)
        {
          Console.WriteLine(" C# ");
          Console.WriteLine(" Accuracy limit of IEEE 32 bit
floating point arithmetic " );
          Console.WriteLine(" program terminates " );
```

```
            return(1);
         }
         array_size = array_size * 10;
      }

      return(0);

}

}
```

10.2.4 Java version

```
class ch3607
{

   public static void main(String[] args)
   {

      int     I;
      float   computed_sum;
      float   actual_sum;
      int     array_size;

      String [] heading = { "    10,000,000", "   100,000,000",
"1,000,000,000" };

      /*

      Initial array size
      10,000,000

      */

      array_size = 10000000;

      for ( I=0 ; i<3 ; I++ )
      {

         System.out.print( " Array size " );
         System.out.println( heading[i] );

         computed_sum = calculate_sum(array_size);
         actual_sum    = array_size*1;

         System.out.print( "    Computed sum " );

         System.out.printf(" %12.1f \n" , computed_sum );
```



```
      System.out.print( "    Actual sum    " );

      System.out.printf(" %12.1f \n" , actual_sum );

      if (actual_sum != computed_sum)
      {
        System.out.println(" Java ");
        System.out.println(" Accuracy limit of IEEE 32 bit
floating point arithmetic " );
        System.out.println(" Program terminates");
        return;
      }
      array_size = array_size * 10;
    }

  }

  static float calculate_sum(int n)
  {
    float [] x = new float [n];
    int i;
    float t;

    t=0;

    for(i=0;i<n;i++)
    {
      x[i]=1;
      t = t + x[i];
    }

    return(t);
  }


}
```

11 Sorting and Searching

In this chapter we look at calling the C++ STL parallel sorting routines from Fortran. Our starting point is the sorting example from chapter 38 in the fourth edition.

11.1 Chapter 38 example 6: calling the C++ STL parallel sort routines

Example 1 in chapter 38 provides a generic sorting module that works with

- 32 bit integers
- 32 bit reals
- 64 bit integers
- 64 bit reals
- 128 bit reals

In this example we provide updated versions that call the C++ STL parallel sorting routines. We provide implementations for

- 32 bit integers
- 32 bit reals
- 64 bit integers
- 64 bit reals

Most C++ compilers don't provide a 128 bit real data type.

11.1.1 C++ code - stl_sort.cxx

Here is the C++ code.

```
#include <execution>
#include <algorithm>
#include <vector>
using namespace std;
extern "C"
{
  void stl_sort_i32(int * x , const int nx)
  {
    vector<int> y(nx);
    int i;
    for(i=0;i<nx;i++)
      y[i]= x[i];
    sort( std::execution::par_unseq, y.begin(), y.end() );
    for(i=0;i<nx;i++)
      x[i]= y[i];
    return;
  }
}
extern "C"
{
  void stl_sort_i64(long long int * x , const int nx)
  {
    vector<long long int> y(nx);
```

```
    int i;
    for(i=0;i<nx;i++)
      y[i]= x[i];
    sort( std::execution::par_unseq, y.begin(), y.end() );
    for(i=0;i<nx;i++)
      x[i]= y[i];
    return;
  }
}
extern "C"
{
  void stl_sort_r32(float * x , const int nx)
  {
    vector<float> y(nx);
    int i;
    for(i=0;i<nx;i++)
      y[i]= x[i];
    sort( std::execution::par_unseq, y.begin(), y.end() );
    for(i=0;i<nx;i++)
      x[i]= y[i];
    return;
  }
}
extern "C"
{
  void stl_sort_r64(double * x , const int nx)
  {
    vector<double> y(nx);
    int i;
    for(i=0;i<nx;i++)
      y[i]= x[i];
    sort( std::execution::par_unseq, y.begin(), y.end() );
    for(i=0;i<nx;i++)
      x[i]= y[i];
    return;
  }
}
```

Note that we have to copy the arrays on both input and output. We use pointers as the parameter passing mechanism from Fortran to C++, and we use the <vector> container class to access the parallel sorting routines in the C++ STL. This functionality came in with the C++ 17 standard.

11.1.2 Fortran wrapper to the C++ STL routines - stl_sort_data_module.f90

Here is the modified generic sort data module. We have replaced the calls to the internal quicksort routine with calls to the C++ sorting routines.

```
module stl_sort_data_module
```



```
  use precision_module
  use integer_kind_module

  interface sort_data

    module procedure sort_real_sp
    module procedure sort_real_dp

    module procedure sort_real_qp
    module procedure sort_integer_8
    module procedure sort_integer_16

    module procedure sort_integer_32
    module procedure sort_integer_64

  end interface

contains

  subroutine sort_real_sp(raw_data, how_many)
    use precision_module
    implicit none
    integer, intent (in) :: how_many
    real (sp), intent (inout), dimension (:) :: raw_data

  interface

    subroutine stl_sort_r32(x,n) bind (c,
name='stl_sort_r32')

      use iso_c_binding

      integer  (c_int)   , value           :: n
      real      (c_float) , dimension(1:n) :: x

      intent (in)                           :: n
      intent (inout)                        :: x

    end subroutine

  end interface

    call stl_sort_r32(raw_data, how_many)

  contains

    recursive subroutine quicksort(l, r)
      implicit none
```

```
      integer, intent (in) :: l, r
      integer :: i, j
      real (sp) :: v, t

      include 'quicksort_include_code.f90'

    end subroutine

  end subroutine

  subroutine sort_real_dp(raw_data, how_many)
    use precision_module
    implicit none
    integer, intent (in) :: how_many
    real (dp), intent (inout), dimension (:) :: raw_data

  interface

    subroutine stl_sort_r64(x,n) bind (c,
name='stl_sort_r64')

      use iso_c_binding

      integer  (c_int)    , value           :: n
      real      (c_double), dimension(1:n)  :: x

      intent (in)                            :: n
      intent (inout)                         :: x

    end subroutine

  end interface

    call stl_sort_r64(raw_data, how_many)

  contains
    recursive subroutine quicksort(l, r)
      implicit none
      integer, intent (in) :: l, r
      integer :: i, j
      real (dp) :: v, t

      include 'quicksort_include_code.f90'

    end subroutine
  end subroutine

  subroutine sort_real_qp(raw_data, how_many)
```



```
  use precision_module
  implicit none
  integer, intent (in) :: how_many
  real (qp), intent (inout), dimension (:) :: raw_data

  call quicksort(1, how_many)

contains
  recursive subroutine quicksort(l, r)
    implicit none
    integer, intent (in) :: l, r
    integer :: i, j
    real (qp) :: v, t

    include 'quicksort_include_code.f90'

  end subroutine
end subroutine

subroutine sort_integer_8(raw_data, how_many)
  use integer_kind_module
  implicit none
  integer, intent (in) :: how_many
  integer (i8), intent (inout), dimension (:) :: raw_data

  call quicksort(1, how_many)

contains
  recursive subroutine quicksort(l, r)
    implicit none
    integer, intent (in) :: l, r
    integer :: i, j
    integer (i8) :: v, t

    include 'quicksort_include_code.f90'

  end subroutine
end subroutine

subroutine sort_integer_16(raw_data, how_many)
  use integer_kind_module
  implicit none
  integer, intent (in) :: how_many
  integer (i16), intent (inout), dimension (:) :: raw_data

  call quicksort(1, how_many)

contains
```

```
    recursive subroutine quicksort(l, r)
      implicit none
      integer, intent (in) :: l, r
      integer :: i, j
      integer (i16) :: v, t

      include 'quicksort_include_code.f90'

    end subroutine
  end subroutine

  subroutine sort_integer_32(raw_data, how_many)
    use integer_kind_module
    implicit none
    integer, intent (in) :: how_many
    integer (i32), intent (inout), dimension (:) :: raw_data

  interface

    subroutine stl_sort_i32(x,n) bind (c,
name='stl_sort_i32')

      use iso_c_binding

      integer  (c_int)   , value           :: n
      integer  (c_int)   , dimension(1:n) :: x

      intent (in)                            :: n
      intent (inout)                         :: x

    end subroutine

  end interface

    call stl_sort_i32(raw_data, how_many)

  contains
    recursive subroutine quicksort(l, r)
      implicit none
      integer, intent (in) :: l, r
      integer :: i, j
      integer (i32) :: v, t

      include 'quicksort_include_code.f90'

    end subroutine
  end subroutine
```



```
  subroutine sort_integer_64(raw_data, how_many)
    use integer_kind_module
    implicit none
    integer, intent (in) :: how_many
    integer (i64), intent (inout), dimension (:) :: raw_data

  interface

    subroutine stl_sort_i64(x,n) bind (c,
name='stl_sort_i64')

      use iso_c_binding

      integer  (c_int)       , value            :: n
      integer  (c_long_long), dimension(1:n)  :: x

      intent (in)                              :: n
      intent (inout)                           :: x

    end subroutine

  end interface

    call stl_sort_i64(raw_data, how_many)

  contains
    recursive subroutine quicksort(l, r)
      implicit none
      integer, intent (in) :: l, r
      integer :: i, j
      integer (i64) :: v, t

      include 'quicksort_include_code.f90'

    end subroutine

  end subroutine
```

end module

11.1.3 Fortran main program - ch3806.f90

Note that we use include statements to make available the other files used in this example:-

- integer_kind_module.f90
- precision_module.f90
- stl_sort_data_module.f90
- timing_module.f90

Here is the full source for the main program.

```
include 'integer_kind_module.f90'
include 'precision_module.f90'
include 'stl_sort_data_module.f90'
include 'timing_module.f90'

program ch3806

  use stl_sort_data_module
  use timing_module

  implicit none
  integer, parameter :: n  =     100000000
  character *12       :: nn = '100,000,000'
  character *80 :: report_file_name = 'ch3801_report.txt'

  real (sp), allocatable, dimension (:) :: x_sp
  real (sp), allocatable, dimension (:) :: t_x_sp
  real (dp), allocatable, dimension (:) :: x_dp
  real (dp), allocatable, dimension (:) :: t_x_dp
  real (qp), allocatable, dimension (:) :: x_qp

  integer (i32), allocatable, dimension (:) :: y_i32
  integer (i64), allocatable, dimension (:) :: y_i64

  integer :: allocate_status = 0

  character *20, dimension (5) :: heading1 = &
  [ '   32 bit real', &
    '   32 bit int ', &
    '   64 bit real', &
    '   64 bit int ', &
    ' 128 bit real' ]

  character *20, dimension (3) :: &
  heading2 = [ '       Allocate ', &
               '       Random    ', &
               '       Sort      ' ]

  print *, 'Program starts'
  print *, 'N = ', nn
  call start_timing()

  open (unit=100, file=report_file_name)

  print *, heading1(1)

  allocate (x_sp(1:n), stat=allocate_status)
  if (allocate_status/=0) then
```



```
    print *, ' Allocate failed. Program terminates'
    stop 10
  end if

  allocate (t_x_sp(1:n), stat=allocate_status)
  if (allocate_status/=0) then
    print *, ' Allocate failed. Program terminates'
    stop 20
  end if

  print 100, heading2(1), time_difference()
  100 format (a20, 2x, f18.6)

  call random_number(x_sp)
  t_x_sp=x_sp

  print 100, heading2(2), time_difference()
  call sort_data(x_sp, n)
  print 100, heading2(3), time_difference()
  write (unit=100, fmt='(a)') ' First 10 32 bit reals'
  write (unit=100, fmt=110) x_sp(1:10)
110 format (5(2x,e14.6))

  print *, heading1(2)

  allocate (y_i32(1:n), stat=allocate_status)
  if (allocate_status/=0) then
    print *, 'Allocate failed. Program terminates'
    stop 30
  end if

  print 100, heading2(1), time_difference()
  y_i32 = int(t_x_sp*1000000000, i32)

  deallocate (x_sp)
  deallocate (t_x_sp)

  print 100, heading2(2), time_difference()
  call sort_data(y_i32, n)
  print 100, heading2(3), time_difference()
  write (unit=100, fmt='(a)') 'First 10 32 bit integers'
  write (unit=100, fmt=120) y_i32(1:10)
120 format (5(2x,i10))
  deallocate (y_i32)

  print *, heading1(3)

  allocate (x_dp(1:n), stat=allocate_status)
```

```
  if (allocate_status/=0) then
    print *, 'Allocate failed. Program terminates'
    stop 30
  end if

  allocate (t_x_dp(1:n), stat=allocate_status)
  if (allocate_status/=0) then
    print *, 'Allocate failed. Program terminates'
    stop 40
  end if

  print 100, heading2(1), time_difference()
  call random_number(x_dp)
  t_x_dp = x_dp
  print 100, heading2(2), time_difference()
  call sort_data(x_dp, n)
  print 100, heading2(3), time_difference()
  write (unit=100, fmt='(a)') 'First 10 64 bit reals'
  write (unit=100, fmt=110) x_dp(1:10)

  print *, heading1(4)

  allocate (y_i64(1:n), stat=allocate_status)
  if (allocate_status/=0) then
    print *, 'Allocate failed. Program terminates'
    stop 40
  end if

  print 100, heading2(1), time_difference()
  y_i64 = int(t_x_dp*1000000000000000_i64, i64)

  deallocate (x_dp)
  deallocate (t_x_dp)

  print 100, heading2(2), time_difference()
  call sort_data(y_i64, n)
  print 100, heading2(3), time_difference()
  write (unit=100, fmt='(a)') 'First 10 64 bit integers'
  write (unit=100, fmt=120) y_i64(1:10)
  deallocate (y_i64)

  print *, heading1(5)

  allocate (x_qp(1:n), stat=allocate_status)
  if (allocate_status/=0) then
    print *, 'Allocate failed. Program terminates'
    stop 50
  end if
```



```
   print 100, heading2(1), time_difference()
   call random_number(x_qp)
   print 100, heading2(2), time_difference()
   call sort_data(x_qp, n)
   print 100, heading2(3), time_difference()
   write (unit=100, fmt='(a)') 'First 10 128 bitreals'
   write (unit=100, fmt=110) x_qp(1:10)

   close (200)
   print *, 'Program terminates'
   call end_timing()

end program
```

We have left in the old quicksort source code.

11.2 Compilation notes

The C++ 17 standard introduced parallel functionality into the STL. You will need a companion C++ compiler that is therefore C++17 compliant.

You will also need a companion C++ compiler that supports STL parallelism.

The Intel compiler provides parallel support via TBB and is automatically available.

The g++ compiler by default has no parallel STL support. It can be made available by installing the Intel oneapi based toolkit and linking against TBB.

The Nag compiler has by default no parallel STL support. It can be made available by installing the Intel oneapi base toolkit and linking against TBB.

There are compatability issues with some Linux distributions when using the Intel compiler.

Full compilation details are available in the compiler Windows batch files and linux shell scripts in the fourth edition tar file available on our web site.

Here are some compilation examples for various compilers.

- Intel Windows

```
ifort -O2 ch3801.f90 -o ch3801_ifort_icl.exe

icl   /O2  -c -std=c++17 stl_sort.cxx
ifort /O2  ch3806.f90 stl_sort.obj /Fech3806_ifort_icl.exe
```

Note the specification of conformance to the C++ 2017 standard.

- Nag linux

```
nagfor -O4     ch3801.f90 -o ch3801_nag.out
nagfor -O4 -c ch3806.cpp -o ch3806_nag.o  -Wc,-std=c++17
nagfor -O4  ch3806.f90  ch3806_nag.o -o ch3806_nag.out
-Wl,-lstdc++ -Wl,-ltbb
```

Note the specification of conformance to the C++ 2017 standard.

Note the explicit linking to the C++ standard library and the Intel Threading Building Blocks.

- gfortran linux

```
gfortran -O2 ch3801.f90 -o ch3801_gfortran.out
g++ -O2 -c ch3806.cxx -o ch3806_gfortran.o -std=c++17
gfortran -O2 ch3806.f90 ch3806_gfortran.o -o
ch3806_gfortran.out -ltbb -lstdc++
```

Note the explicit linking to the C++ standard library and Intel Threading Building Blocks.

11.2.1 Timing results

The examples have been compiled on a number of machines, using a number of operating systems, including native Windows, native linux, linux under hyper-v and linux under WSL

- Dell 5280 workstation
    - Intel I9 10980XE (18 core * 2 with hyper threading) processor with 128 GB ram.
- Dell Vostro 5515 laptop
    - AMD Ryzen 7 5700U (8 cores * 2 with hyper threading) with 32 GB ram
- Dell Studio XPS 7100
    - AMD Phenom II X6 1055T, 6 cores, 16GB ram

The tables below summarise some of the runs on these systems.

			gfortran		Intel ifort	
	Dell 5280			Real	hyper-v	Redhat 9.2
	Processor	I9	10980XE	36 cores	16 cores	
	Memory			128 GB	32 GB	
N Bits	Type	Method	gfortran		Intel	
					ifort	ifort
			ch3801	ch3806	ch3801	ch3806
				C++ stl		C++ stl
			serial	parallel	serial	parallel
32 bit	int	allocate	0.000050	0.000043	0.000053	0.000032
32 bit	int	random	0.257852	0.253551	0.514170	0.498290
32 bit	int	sort	9.703295	0.859386	9.211571	1.007689
Percentage run time				8.86%		10.94%
32 bit	real	allocate	0.000652	0.000432	0.000632	0.000660
32 bit	real	random	0.091293	0.093512	0.066262	0.070446
32 bit	real	sort	8.820538	0.898621	10.164119	0.928175
Percentage run time				10.19%		9.13%
64 bit	int	allocate	0.000896	0.000909	0.000903	0.000910
64 bit	int	random	0.558384	0.560496	0.943222	0.915312
64 bit	int	sort	10.534979	1.023443	10.983383	1.140597
Percentage run time				9.71%		10.38%
64 bit	real	allocate	0.000079	0.000069	0.000099	0.000083
64 bit	real	random	0.161632	0.154877	0.140927	0.148834
64 bit	real	sort	9.849819	1.037352	9.587968	1.158036
Percentage run time				10.53%		12.08%

No 128 bit timings.

			Intel ifx		Nag	
	Dell 5280			Real	hyper-v	Redhat 9.2
	Processor	I9	10980XE	36	16	
	Memory			128	32	
N Bits	Type	Method	Intel		Nag	
			ifx	ifx		
			ch3801	ch3806	ch3801	ch3806
				C++ stl		C++ stl
			serial	parallel	serial	parallel
32 bit	int	allocate	0.000030	0.000042	0.000226	0.000125
32 bit	int	random	1.197002	1.204383	0.373733	0.371976
32 bit	int	sort	9.617738	1.003027	9.872310	0.954357
Percentage run time				10.43%		9.67%
32 bit	real	allocate	0.000823	0.000583	0.000102	0.000229
32 bit	real	random	0.072183	0.070210	0.134922	0.141273
32 bit	real	sort	8.355149	0.943834	8.569433	0.815349
Percentage run time				11.30%		9.51%
64 bit	int	allocate	0.000981	0.000857	0.000903	0.000978
64 bit	int	random	1.313579	1.329516	0.996867	1.005611
64 bit	int	sort	9.488351	1.165321	10.512199	1.069079
Percentage run time				12.28%		10.17%
64 bit	real	allocate	0.000089	0.000103	0.000125	0.000152
64 bit	real	random	0.159218	0.151523	0.365365	0.306116
64 bit	real	sort	8.827790	1.158346	9.540932	1.036015
Percentage run time				13.12%		10.86%

No 128 bit timings.

Dell	Vostro		5515		laptop	
Memory	32		16		16	
Cores	16		8		8	
OS	Windows		Redhat		Redhat	
	Native		hyper-v		Hype-v	
Compiler	Intel		gfortran		nag	
Source file(s)	ch3801	ch3806	ch3801	ch3806	ch3801	ch3806
Data type	Serial	Parallel	Serial	Parallel	Serial	Parallel
32 bit real	11.211	1.594	11.024	1.778	11.629	1.783
32 bit int	9.971	1.282	9.043	1.473	9.282	1.463
64 bit real	11.768	1.906	12.022	2.076	11.917	2.068
64 bit int	9.382	1.641	9.522	1.822	9.736	1.837
128 bit real	24.433	24.085	33.647	34.411	12.566	12.539

Serial and parallel timings only.

Dell	Studio	XPS	7100	
Memory	16			
Cores	6			
OS	openSuSe		Windows	
	native		native	
Compiler	Nag		Intel	
Source file(s)	ch3801	ch3806	ch3801	ch3806
	serial	parallel	serial	parallel
data type				
32 bit real	14.914	2.932	15.164	3.415
32 bit int	14.529	2.456	16.122	3.12
64 bit real	16.552	3.741	15.659	4.166
64 bit int	14.711	3.495	13.153	3.562
128 bit real	17.714	17.034	59.857	51.928

Serial and parallel timings only.

11.3 Problems

Try these examples out with your compiler.

11.4 Bibliography and companion C++ material

There are a set of companion C++ notes and examples available.They can be found at
`https://www.rhymneyconsulting.co.uk/cpp/`

12 Handling missing data using nans

There are several changes to the examples in

- Chapter 39, handling missing data in statistical calculations

We now have

- new C# example to get the data files
- new sed script to convert --- to NANs
- rewrite of statistical program to detect NANs rather than flag values

12.1 Chapter 39 example 5: Replacement C# program and new Python program to get the Met Office files

Here is the replacement C# source file.

```
using System;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Text;
class ch3901
{
  static int Main()
  {
    const int n_sites=37;
    string base_address =
    @"https://www.metoffice.gov.uk/pub/"
    +"data/weather/uk/climate/stationdata/";
    string [] station_name =
    {
    "aberporth",         "armagh",
    "ballypatrick",     "bradford",
    "braemar",           "camborne",
    "cambridge",         "cardiff",
    "chivenor",          "cwmystwyth",
    "dunstaffnage",     "durham",
    "eastbourne",        "eskdalemuir",
    "heathrow",          "hurn",
    "lerwick",           "leuchars",
    "lowestoft",         "manston",
    "nairn",              "newtonrigg",
    "oxford",            "paisley",
    "ringway",           "rossonwye",
    "shawbury",          "sheffield",
    "southampton",      "stornoway",
    "suttonbonington",  "tiree",
    "valley",             "waddington",
    "whitby",             "wickairport",
    "yeovilton",
```

```
    };
    string [] web_address = new string[n_sites];
    string last_part="data.txt";
    string input_string;
    int i;
    // create the web address of each file
    for (i=0;i<n_sites;i++)
    {
      web_address[i]=
      base_address+station_name[i]+last_part;
      System.Console.WriteLine(web_address[i]);
    }
    string[] local_data_file =
    {
      "aberporthdata.txt",          "armaghdata.txt",
      "ballypatrickdata.txt",       "bradforddata.txt",
      "braemardata.txt",            "cambornedata.txt",
      "cambridgedata.txt",          "cardiffdata.txt",
      "chivenordata.txt",           "cwmystwythdata.txt",
      "dunstaffnagedata.txt",       "durhamdata.txt",
      "eastbournedata.txt",         "eskdalemuirdata.txt",
      "heathrowdata.txt",           "hurndata.txt",
      "lerwickdata.txt",            "leucharsdata.txt",
      "lowestoftdata.txt",          "manstondata.txt",
      "nairndata.txt",              "newtonriggdata.txt",
      "oxforddata.txt",             "paisleydata.txt",
      "ringwaydata.txt",            "rossonwyedata.txt",
      "shawburydata.txt",           "sheffielddata.txt",
      "southamptondata.txt",        "stornowaydata.txt",
      "suttonboningtondata.txt",   "tireedata.txt",
      "valleydata.txt",             "waddingtondata.txt",
      "whitbydata.txt",             "wickairportdata.txt",
      "yeoviltondata.txt"
    };

    StreamWriter output_file;
    for (i=0;i<n_sites;i++)
    {
      ServicePointManager.Expect100Continue = true;
      ServicePointManager.SecurityProtocol =
SecurityProtocolType.Tls
       | SecurityProtocolType.Tls11
       | SecurityProtocolType.Tls12
       | SecurityProtocolType.Ssl3;
      // create the web addresses
      System.Console.WriteLine(" Create the web addresses");
      HttpWebRequest  httpwreq   = (HttpWebRequest)
      WebRequest.Create(web_address[i]);
```



```
        // set up connection
        System.Console.WriteLine(" Set up the connection");
        HttpWebResponse httpwresp  = (HttpWebResponse)
        httpwreq.GetResponse();
        // set up input stream
        System.Console.WriteLine(" Set up the input stream");
        StreamReader input_stream = new
          StreamReader
          (httpwresp.GetResponseStream(),Encoding.ASCII);
        // read the whole file
        System.Console.WriteLine(" Read the whole file");
        input_string=input_stream.ReadToEnd();
        // create the output file
        System.Console.WriteLine(" Create the output file");
        output_file =
        File.CreateText("before_"+local_data_file[i]);
        output_file.WriteLine(input_string);
        input_stream.Close();
        output_file.Close();
        System.Console.WriteLine(" Close the files");
      }
      return(0);
    }
}
```

Here is the new Python equivalent.

```
import time
import requests

def main():

  start_time=time.time()
  print(" ** Start time              **",end=" ")
  print(start_time)

  n_stations = 37

  base_address =
"http://www.metoffice.gov.uk/pub/data/weather/uk/cli-
mate/stationdata/"

  station_names = ["aberporthdata.txt"        ,
"armaghdata.txt"       , "ballypatrickdata.txt",
    "bradforddata.txt"          , "braemardata.txt"      ,
"cambornedata.txt",
    "cambridgedata.txt"         , "cardiffdata.txt"      ,
"chivenordata.txt",
```

```
    "cwmystwythdata.txt"       , "dunstaffnagedata.txt",
"durhamdata.txt",
    "eastbournedata.txt"       , "eskdalemuirdata.txt" ,
"heathrowdata.txt",
    "hurndata.txt"              , "lerwickdata.txt"      ,
"leucharsdata.txt",
    "lowestoftdata.txt"        , "manstondata.txt"      ,
"nairndata.txt",
    "newtonriggdata.txt"       , "oxforddata.txt"       ,
"paisleydata.txt",
    "ringwaydata.txt"           , "rossonwyedata.txt"    ,
"shawburydata.txt",
    "sheffielddata.txt"        , "southamptondata.txt" ,
"stornowaydata.txt",
    "suttonboningtondata.txt" , "tireedata.txt"         ,
"valleydata.txt",
    "waddingtondata.txt"       , "whitbydata.txt"       ,
"wickairportdata.txt",
    "yeoviltondata.txt"]

  for I in range(0,n_stations):

     print(       station_names[i]   )
     complete_address = base_address + station_names[i]
     f=open(station_names[i],"w")
     station_data = requests.get(url=complete_ad-
dress).text.replace('\r','')
     f.write(station_data)
     f.close()

  t1=time.time()
  file_read=t1-start_time
  print(" ** Internet file read took  **",end=" ")
  print(" {0:12.6f}".format(file_read))

if ( __name__ == "__main__" ):
  main()
```

This will work on both Windows and Linux.

12.2 Chapter 39 example 6: sed script

Here is the sed script.

```
s/ ---/ nan/g
```

12.3 Chapter 39 example 7: Statistical calculations using NANs

Here is the source file.

```
module statistics_module
```



```
  use ieee_arithmetic

  implicit none

contains

  subroutine calculate_month_averages(x, n, n_months, sum_x,
average_x, index_by_month, month_names)

    implicit none

    real, dimension (:), intent (in) :: x
    integer, intent (in) :: n
    integer, intent (in) :: n_months

    real, dimension (1:n_months), intent (inout) :: sum_x
    real, dimension (1:n_months), intent (inout) :: aver-
age_x

    integer, dimension (1:n), intent (in) :: index_by_month
    character *9, dimension (1:n_months), intent (in) ::
month_names

    integer, dimension (1:n_months) :: n_missing
    integer, dimension (1:n_months) :: n_actual

    integer :: m

    sum_x = 0.0
    average_x = 0.0
    n_missing = 0
    n_actual = 0

    do m = 1, n
      if ( ieee_is_nan(x(m)) ) then
        n_missing(index_by_month(m)) = n_missing(in-
dex_by_month(m)) + 1
      else
        sum_x(index_by_month(m)) = sum_x(index_by_month(m))
+ x(m)
        n_actual(index_by_month(m)) = n_actual(in-
dex_by_month(m)) + 1
      end if
    end do

    do m = 1, n_months
      average_x(m) = sum_x(m)/(n_actual(m))
    end do
```

```
    print *, ' Summary of actual     missing'
    print *, '                values     values'
    do m = 1, n_months
      print 100, month_names(m), n_actual(m), n_missing(m)
100   format (2x, a9, 2x, i6, 2x, i6)
    end do

  end subroutine

end module
```

Here is the replacement main driving program.

```
include 'ch3906_statistics_module.f90'
include 'ch3903_met_office_station_module.f90'
include 'timing_module.f90'

program ch3907

  use met_office_station_module
  use statistics_module
  use timing_module

  implicit none

! met office data user defined type

  type (station_type), dimension (:), allocatable :: sta-
tion_data

! Temporary variables used on the read

  integer :: year
  integer :: month
  real     :: tmax
  real     :: tmin
  integer :: af_days
  real     :: r_af_days
  real     :: rainfall
  real     :: sunshine

! Currently we only calculate the
! rainfall sum and averages.

! real, dimension (1:n_months) :: sum_tmax
! real, dimension (1:n_months) :: sum_tmin
! real, dimension (1:n_months) :: sum_af_days
  real, dimension (1:n_months) :: sum_rainfall
```

```
! real, dimension (1:n_months) :: sum_sunshine

! real, dimension (1:n_months) :: average_tmax
! real, dimension (1:n_months) :: average_tmin
! real, dimension (1:n_months) ::
! average_af_days
  real, dimension (1:n_months) :: average_rainfall
! real, dimension (1:n_months) ::
! average_sunshine

! Table to hold the monthly rainfall averages
! for all stations.

  real, dimension (1:n_months, 1:n_stations) :: rainfall_ta-
ble = 0

  integer :: n_years

  integer :: I, j

  call start_timing()

  call initialise_station_data()

! Process each station

  do j = 1, n_stations

    print *, ' '
    print *, ' Processing ', station_data_file_name(j)
    print *, ' '

    open (unit=100, file=station_data_file_name(j), sta-
tus='old')

!   skip the header lines before starting to
!   read the data

    call skip_header_lines(j)

!   the number of observations at each station
!   is stored in the nl array.

    allocate (station_data(1:nl(j)))

!   Read in the data for each station

    do I = 1, nl(j)
```

```
       read (unit=100, fmt=100) year, month, tmax, tmin,
r_af_days, rainfall, sunshine
100    format (2x, i5, 2x, i2, 2x, f5.1, 3x, f5.1, 3x, f5.0,
2x, f6.1, 2x, f6.1)
       if ( ieee_is_nan(r_af_days) ) then
         af_days = -99
       else
         af_days = int(r_af_days)
       end if
       station_data(I) = station_type(year, month, tmax,
tmin, af_days, rainfall, sunshine)
     end do

     close (100)

!    Do the monthly average calculations
!    for each station

     call calculate_month_averages(station_data%rainfall,
nl(j), n_months, sum_rainfall, average_rainfall, sta-
tion_data%month, &
       month_names)

     n_years = station_data(nl(j))%year - station_data(1)%year
+ 1

     print *, ' '
     print *, ' Start date ', station_data(1)%year, ' ', sta-
tion_data(1)%month
     print *, ' '
     print *, ' Rainfall monthly averages over'
     print 110, n_years
110 format ('   ~ ', i5, ' years            mm     ins')
     do I = 1, n_months
       print 120, month_names(I), average_rainfall(I), (aver-
age_rainfall(I)/25.4)
120    format (2x, a9, 8x, f7.2, 2x, f5.2)
     end do
     print 130, sum(average_rainfall), (sum(average_rain-
fall)/25.4)
130 format ('   Annual rainfall', /, '   average           ',
f8.2, 2x, f5.2)
     print *, ' '
     print *, ' End date   ', station_data(nl(j))%year, ' ',
station_data(nl(j))%month

     rainfall_table(1:n_months, j) = average_rainfall
```

```
!    Deallocate the arrays

     deallocate (station_data)

!    move on to next station

   end do

   print *, ' '
   print 140, site_name(1:n_stations)
140 format (37(2x,a7))
   print *, ' '

   do I = 1, n_months
     print 150, rainfall_table(I, 1:n_stations)/25.4
150 format (37(2x,f7.2))
   end do

   call end_timing()

end program
```

13 Miscellaneous new examples

One or more files are required for these examples. All files are available on our web site. Here is a link

`https://www.rhymneyconsulting.co.uk/fortran/`

The tar and zip files contain both all of the fourth edition examples plus all new examples.

13.1 Chapter 43 example 1: Adding commas to integer output

The following three include files are required:

- include 'integer_kind_module.f90'
- include 'ch4301_display_with_commas_module.f90'
- include 'ch4301_display_with_commas_test_program.f90'

ch4301.f90 has the above three include statements.

Here is sample output.

```
Positive
32 bit
            2147483647                2,147,483,647
               8388607                    8,388,607
                 32767                       32,767
                   127                          127
Positive
64 bit
   9223372036854775807   9,223,372,036,854,775,807
     36028797018963967      36,028,797,018,963,967
       140737488355327         140,737,488,355,327
          549755813887             549,755,813,887
            2147483647               2,147,483,647
               8388607                   8,388,607
                 32767                      32,767
                   127                         127
Negative
32 bit
           -2147483647               -2,147,483,647
              -8388607                   -8,388,607
                -32767                      -32,767
                  -127                         -127
Negative
64 bit
  -9223372036854775807  -9,223,372,036,854,775,807
    -36028797018963967     -36,028,797,018,963,967
      -140737488355327        -140,737,488,355,327
         -549755813887            -549,755,813,887
           -2147483647              -2,147,483,647
              -8388607                  -8,388,607
                -32767                     -32,767
                  -127                        -127
```

The original program only supported positive 64 integers, as we were only interested in producing more readable output in the later memory examples. This version has 32 bit integer support and negative integer support.

Here is the test program.

```
program test

   use integer_kind_module
   use display_with_commas_module

   integer (i32)        :: x=2147483647
   integer (i64)        :: y=9223372036854775807_i64
   integer (i32)        :: x1=-2147483647
   integer (i64)        :: y1=-9223372036854775807_i64

   integer :: i

   print *,' Positive'
   print *,' 32 bit'

   do I=1,4

      print 10,x,display_with_commas(x)
      10 format(2x,i22,2x,a)
      x=x/256

   end do

   print *,' Positive'
   print *,' 64 bit'

   do I=1,8

      print 10,y,display_with_commas(y)
      y=y/256

   end do

   print *,' Negative'
   print *,' 32 bit'

   do I=1,4

      print 10,x1,display_with_commas(x1)
      x1=x1/256

   end do

   print *,' Negative'
   print *,' 64 bit'

   do I=1,8
```

```
    print 10,y1,display_with_commas(y1)
    y1=y1/256

  end do
end program test
```

The files are on our web site.

13.2 Chapter 43 example 2: Kahan summation with timing

The following source files are required.

- include 'integer_kind_module.f90'
- include 'precision_module.f90'
- include 'timing_module.f90'
- include 'kahan_summation_module.f90'

ch4302.f90 is a test program that contains the above include files.

13.2.1 Sample output

Here is some sample output.

```
 2022/ 5/ 5 13:51:32  76
 N =      10000000
 Allocate                  0.000000000000000000
 Initialise                0.162999868392944336
 Intrinsic summation     0.000000000000000000
                                                         5000444.2793215252
 Kahan summation          0.062999963760375977
                                                         5000444.2793215429
 N =     100000000
 Allocate                  0.000000000000000000
 Initialise                1.616000175476074219
 Intrinsic summation     0.108999967575073242
                                                        49998117.4713004455
 Kahan summation          0.524999856948852539
                                                        49998117.4712983146
 N =    1000000000
 Allocate                  0.047000169754028320
 Initialise               16.238999843597412109
 Intrinsic summation     1.116000175476074219
                                                       499995574.2241585851
 Kahan summation          5.306999921798706055
                                                       499995574.2241371870
 2022/ 5/ 5 13:51:57 720
 Total time =                          25.629000
```

13.2.2 Test program

Here is the test program.

```
include 'integer_kind_module.f90'
include 'precision_module.f90'

include 'timing_module.f90'

include 'kahan_summation_module.f90'
```

```
program ch4302

  use timing_module
  use precision_module

  use kahan_summation_module

  implicit none

  integer (i64) :: n = 10000000_i64
  integer :: I
  integer :: j = 3

  real (dp), allocatable, dimension (:)&
                                 :: x
  real (dp) &
                                 :: x_sum = 0.0_dp

  call start_timing()

  do i=1,j

    print 10,n
    10 format(' N = ',i12)

    allocate(x(n))

    print 20,time_difference()
    20 format(' Allocate               ',f22.18)

    call random_number(x)

    print 30,time_difference()
    30 format(' Initialise             ',f22.18)

    x_sum=sum(x)

    print 40, time_difference()
    40 format(' Intrinsic summation ',f22.18)

    print 100, x_sum
    100 format(45x,f20.10)

    x_sum=kahan_sum(x,n)

    print 50, time_difference()
    50 format(' Kahan summation      ',f22.18)
```

```
    print 100, x_sum

    deallocate(x)

    n=n*10_i64

  end do

  call end_timing()

end program ch4302
```

13.2.3 Kahan summation module

Here is the Kahan Summation module.

```
module kahan_summation_module

  use integer_kind_module
  use precision_module

contains

    function kahan_sum(x,n)

    implicit none

    real     (dp)   , intent(in) , dimension (:) :: x
    integer  (i64)  , intent(in)                     :: n

    real  (dp)                                     :: kahan_sum


    real  (dp)                                     :: sum
    real  (dp)                                     :: c
    real  (dp)                                     :: y
    real  (dp)                                     :: t

    integer  (i64)  :: I

    kahan_sum=0.0_dp
    sum        =0.0_dp
    c           =0.0_dp

    do i=1,n

      y = x(I) - c
      t = sum + y
      c = (t - sum) - y
```

```
      sum = t

    end do

    kahan_sum=sum

  end function kahan_sum

end module kahan_summation_module
```

13.3 Chapter 43 example 3: duplicate of ch1814, using the display_with_commas module

Here is the main program.

```
!
! Example to show array memory allocation
! using a range of compilers.
!
! We have several types of array
!
! 1.0 main program array
!
! 1.1 dynamic allocation in the main program
!
! 2.0 automatic allocation in a subroutine
!
! 3.0 dynamic allocation in a subroutime
!

include 'integer_kind_module.f90'
include 'ch4301_display_with_commas_module.f90'

program ch4303

  use iso_c_binding
  use integer_kind_module
  use display_with_commas_module
  use iso_fortran_env

  implicit none

  integer , parameter                                     :: n =
1024 * 1024
  integer                                                  :: i
  integer , dimension(n) , target                        :: y
  integer , dimension(:) , allocatable , target :: z

  type    (c_ptr)   :: x
  integer (i64)     :: address_as_integer
```

```
   print *,''
   print *,compiler_version()
   print *,''

   do i=1,n
     y(i)=i
   end do

   x = c_loc(y)
   address_as_integer = transfer(x,address_as_integer)
   print 10,address_as_integer,display_with_commas(ad-
dress_as_integer)
   10 format(' Main program normal array       ',i20,2x,a)

   allocate(z(n))

   z=y

   x = c_loc(z)
   address_as_integer = transfer(x,address_as_integer)
   print 20,address_as_integer,display_with_commas(ad-
dress_as_integer)
   20 format(' Main program allocatable array ',i20,2x,a)

   call automatic_array(n)

   call allocatable_array(n)

end program

subroutine automatic_array(n)

   use iso_c_binding
   use integer_kind_module
   use display_with_commas_module

   implicit none

   integer , intent(in) :: n
   integer , dimension(n) , target :: z
   integer :: i

   type     (c_ptr)   :: x
   integer (i64)      :: address_as_integer

   do i=1,n
     z(i)=i
```

```
  end do

  x = c_loc(z)
  address_as_integer = transfer(x,address_as_integer)
  print 10,address_as_integer,display_with_commas(ad-
dress_as_integer)
  10 format(' Subroutine automatic array      ',i20,2x,a)

end subroutine

subroutine allocatable_array(n)

  use iso_c_binding
  use integer_kind_module
  use display_with_commas_module

  implicit none

  integer , intent(in) :: n
  integer , dimension(:) , allocatable , target :: z
  integer :: i

  type    (c_ptr)   :: x
  integer (i64)     :: address_as_integer

  allocate(z(n))

  do i=1,n
    z(i)=i
  end do

  x = c_loc(z)
  address_as_integer = transfer(x,address_as_integer)
  print 10,address_as_integer,display_with_commas(ad-
dress_as_integer)
  10 format(' Subroutine allocatable array    ',i20,2x,a)

end subroutine
```

13.3.1 Sample output for the Nag, Intel and gfortran compilers under Windows and Linux

- gfortran Windows

```
 GCC version 13.2.0

 Main program normal array             140700387307584
140,700,387,307,584
```

```
 Main program allocatable array         2239357063232
2,239,357,063,232
 Subroutine automatic array              2239361343552
2,239,361,343,552
 Subroutine allocatable array           2239361347648
2,239,361,347,648
```

- Nag Windows

```
 NAG Fortran Compiler Release 7.1(Hanzomon) Build 7110

 Main program normal array                          4236224
4,236,224
 Main program allocatable array                 150339584
150,339,584
 Subroutine automatic array                      154533888
154,533,888
 Subroutine allocatable array                   158728192
158,728,192
```

- Intel Windows

The default compile causes a stack overflow error. The program must be compiled with the -heap-arrays compiler flag.

```
 Intel(R) Fortran Intel(R) 64 Compiler Classic for applica-
tions running on Intel
 (R) 64, Version 2021.10.0 Build 20230609_000000

 Main program normal array             140698413403840
140,698,413,403,840
 Main program allocatable array         2490087874640
2,490,087,874,640
 Subroutine automatic array              2490092109904
2,490,092,109,904
 Subroutine allocatable array           2490092122192
2,490,092,122,192
```

13.4 Chapter 43 example 4: rewrite of generic statistics module (ch2502) to support large array sizes using 64 bit integers

Here is the main program

```
include 'precision_module.f90'
include 'integer_kind_module.f90'
include 'statistics_module_64.f90'
include 'timing_module.f90'
include 'ch4301_display_with_commas_module.f90'

program ch4304
```

```
  use precision_module
  use statistics_module
  use timing_module
  use iso_fortran_env
  use display_with_commas_module

  implicit none
  integer (i64) :: n
  integer :: i
  integer :: repeat_count
  real (sp), allocatable, dimension (:) :: x
  real (sp) :: x_m, x_sd, x_median
  real (dp), allocatable, dimension (:) :: y
  real (dp) :: y_m, y_sd, y_median
  real (qp), allocatable, dimension (:) :: z
  real (qp) :: z_m, z_sd, z_median
  character *20, dimension (3) :: heading = [ '  Allocate
', '  Random       ', '  Statistics  ' ]

  print *,''
  print *,compiler_version()
  print *,''

  call start_timing()
  n              = 1024 * 1024 * 1024
  repeat_count = 4

  do i=1,repeat_count

  print 10,n,display_with_commas(n)
  10 format(2x,i22,2x,a)

  print *, ' Single precision'

  allocate (x(1:n))
  print 100, heading(1), time_difference()
100 format (a20, 6x, f18.6)
  call random_number(x)
  print 100, heading(2), time_difference()
  call calculate_statistics(x, n, x_m, x_sd, x_median)
  print 100, heading(3), time_difference()
  write (unit=*, fmt=110) x_m
110 format (' Mean                 = ', f10.6)
  write (unit=*, fmt=120) x_sd
120 format (' Standard deviation = ', f10.6)
  write (unit=*, fmt=130) x_median
130 format (' Median               = ', f10.6)
```

```
  deallocate (x)

  print *, ' Double precision'

  allocate (y(1:n))
  print 100, heading(1), time_difference()
  call random_number(y)
  print 100, heading(2), time_difference()
  call calculate_statistics(y, n, y_m, y_sd, y_median)
  print 100, heading(3), time_difference()
  write (unit=*, fmt=110) y_m
  write (unit=*, fmt=120) y_sd
  write (unit=*, fmt=130) y_median
  deallocate (y)

!  print *, ' Quad precision'
!
!  allocate (z(1:n))
!  print 100, heading(1), time_difference()
!  call random_number(z)
!  print 100, heading(2), time_difference()
!  call calculate_statistics(z, n, z_m, z_sd, z_median)
!  print 100, heading(3), time_difference()
!  write (unit=*, fmt=110) z_m
!  write (unit=*, fmt=120) z_sd
!  write (unit=*, fmt=130) z_median
!  deallocate (z)

  n = n * 2

  end do

  call end_timing()

end program ch4304
```

13.4.1 Sample output for the Nag, Intel and gfortran compilers inder Windows and Linux

- gfortran Windows

```
C:\document\fortran\4th_edition_update\exam-
ples>ch4304_gfortran.exe

 GCC version 13.2.0

 2023/11/ 7 12:38:10 485
                1073741824                 1,073,741,824
  Single precision
  Allocate                              0.000987
```

```
 Random                                2.005380
 Statistics                           11.857487
Mean                =    0.015625
Standard deviation =   0.124020
Median              =    0.500002
 Double precision
 Allocate                              0.125542
 Random                                4.547941
 Statistics                           12.649285
Mean                =    0.500000
Standard deviation =   0.288671
Median              =    0.499996
            2147483648                     2,147,483,648
 Single precision
 Allocate                              0.215840
 Random                                4.083466
 Statistics                           23.578701
Mean                =    0.007812
Standard deviation =   0.088042
Median              =    0.499994
 Double precision
 Allocate                              0.214233
 Random                                9.047108
 Statistics                           30.270646
Mean                =    0.500014
Standard deviation =   0.288674
Median              =    0.500018
            4294967296                     4,294,967,296
 Single precision
 Allocate                              0.428649
 Random                                8.095456
 Statistics                           53.014031
Mean                =    0.003906
Standard deviation =   0.062378
Median              =    0.500012
 Double precision
 Allocate                              0.486777
 Random                               18.309089
 Statistics                           60.314001
Mean                =    0.500000
Standard deviation =   0.288676
Median              =    0.499994
            8589934592                     8,589,934,592
 Single precision
 Allocate                              1.221751
 Random                               16.181649
 Statistics                          102.364031
Mean                =    0.001953
```

```
 Standard deviation =   0.044151
 Median              =   0.499992
  Double precision
  Allocate                              1.041581
  Random                               36.459376
  Statistics                          169.129995
 Mean                =   0.499994
 Standard deviation =   0.288674
 Median              =   0.499995
 2023/11/ 7 12:47:38 290
 Total time =                         567.804238

C:\document\fortran\4th_edition_update\examples>
```

- Nag Windows

```
C:\document\fortran\4th_edition_update\examples>ch4304_nag.exe

 NAG Fortran Compiler Release 7.1(Hanzomon) Build 7110

 2023/11/ 7 13:24: 4 531
             1073741824                  1,073,741,824
  Single precision
  Allocate                              0.002557
  Random                                3.270837
  Statistics                           14.537686
 Mean                =   0.015625
 Standard deviation =   0.124020
 Median              =   0.500009
  Double precision
  Allocate                              0.125314
  Random                                2.862425
  Statistics                           15.025153
 Mean                =   0.500005
 Standard deviation =   0.288672
 Median              =   0.499999
             2147483648                  2,147,483,648
  Single precision
  Allocate                              0.219053
  Random                                7.172563
  Statistics                           24.062931
 Mean                =   0.007812
 Standard deviation =   0.088042
 Median              =   0.499982
  Double precision
  Allocate                              0.278872
  Random                                5.987126
  Statistics                           27.168607
```

```
 Mean                  =    0.500003
 Standard deviation =    0.288671
 Median               =    0.499989
               4294967296                4,294,967,296
  Single precision
  Allocate                              0.419991
  Random                                0.000122
  Statistics                           25.099229
 Mean                  =    0.000000
 Standard deviation =    0.000135
 Median               =    0.000000
  Double precision
  Allocate                              0.482229
  Random                                0.000170
  Statistics                           38.768510
 Mean                  =    0.000000
 Standard deviation =    0.000220
 Median               =    0.000000
               8589934592                8,589,934,592
  Single precision
  Allocate                              1.288716
  Random                                0.000162
  Statistics                           57.107477
 Mean                  =    0.000000
 Standard deviation =    0.000155
 Median               =    0.000000
  Double precision
  Allocate                              0.970672
  Random                                0.000145
  Statistics                          110.577813
 Mean                  =    0.000000
 Standard deviation =    0.000158
 Median               =    0.000000
 2023/11/ 7 13:29:41 683
 Total time =                         337.139574
```

- Intel Windows; the default compile generates a stack error. You need to add the -heap-arrays compiler flag.

```
C:\document\fortran\4th_edition_update\exam-
ples>ch4304_intel.exe

 Intel(R) Fortran Intel(R) 64 Compiler Classic for applica-
tions running on Intel
 (R) 64, Version 2021.10.0 Build 20230609_000000

 2023/11/ 7 14: 1:18   53
               1073741824                1,073,741,824
  Single precision
```

```
  Allocate                                 0.000000
  Random                                   4.594000
  Statistics                              12.969000
 Mean                 =    0.125000
 Standard deviation =    0.330719
 Median               =    0.499988
  Double precision
  Allocate                                 0.125000
  Random                                   7.953000
  Statistics                              12.269000
 Mean                 =    0.500011
 Standard deviation =    0.288677
 Median               =    0.500016
             2147483648                  2,147,483,648
  Single precision
  Allocate                                 0.329000
  Random                                   9.206000
  Statistics                              19.750000
 Mean                 =    0.062500
 Standard deviation =    0.242061
 Median               =    0.500007
  Double precision
  Allocate                                 0.235000
  Random                                  15.926000
  Statistics                              21.551000
 Mean                 =    0.500012
 Standard deviation =    0.288677
 Median               =    0.500020
             4294967296                  4,294,967,296
  Single precision
  Allocate                                 0.656000
  Random                                  18.755000
  Statistics                              40.801000
 Mean                 =    0.031250
 Standard deviation =    0.173993
 Median               =    0.500010
  Double precision
  Allocate                                 0.453000
  Random                                  31.863000
  Statistics                              48.201000
 Mean                 =    0.500002
 Standard deviation =    0.288673
 Median               =    0.499996
             8589934592                  8,589,934,592
  Single precision
  Allocate                                 1.062000
  Random                                  39.670000
  Statistics                              98.813000
```



```
 Mean                   =    0.015625
 Standard deviation =    0.124020
 Median                 =    0.500009
  Double precision
  Allocate                               1.031000
  Random                                64.935000
  Statistics                           288.877000
 Mean                   =    0.499998
 Standard deviation =    0.288675
 Median                 =    0.499992
 2023/11/ 7 14:13:42   45
 Total time =                          743.992000

C:\document\fortran\4th_edition_update\examples>
```

13.5 Files and compilation details

Here is a list of the files associated with this chapter.

- ch4301.f90
- ch4301_display_with_commas.f90
- ch4301_display_with_commas_module.f90
- ch4301_display_with_commas_test_program.f90

- ch4302.f90
- ch4302_kahan_sum.c
- ch4302_kahan_summation_module.f90

- ch4303.f90
- integer_kind_module.f90
- ch4301_display_with_commas_module.f90

- ch4304.f90
- precision_module.f90
- integer_kind_module.f90
- statistics_module_64.f90
- timing_module.f90
- ch4301_display_with_commas_module.f90

Where multiple files are involved we have provided batch files and shell scripts to help out.

14 Using the Windows and Linux memory api's

One or more files are required for these examples. All files are available on our web site. Here is a link

https://www.rhymneyconsulting.co.uk/fortran/

The tar and zip files contain both all of the fourth edition examples plus all new examples.

14.1 Chapter 44 example 1: Querying memory availability and usage using the Windows API

Microsoft has an api that provides access to information about memory usage on a Windows system. Here is a link to their documentation.

https://docs.microsoft.com/en-us/windows/win32/api/sysinfoapi/ns-sysinfoapi-memorystatusex

Here is the associated struct.

```
typedef struct _MEMORYSTATUSEX {
  DWORD     dwLength;
  DWORD     dwMemoryLoad;
  DWORDLONG ullTotalPhys;
  DWORDLONG ullAvailPhys;
  DWORDLONG ullTotalPageFile;
  DWORDLONG ullAvailPageFile;
  DWORDLONG ullTotalVirtual;
  DWORDLONG ullAvailVirtual;
  DWORDLONG ullAvailExtendedVirtual;
} MEMORYSTATUSEX, *LPMEMORYSTATUSEX;
```

In this example we provide a Fortran interface to this information, using the C interop facilities available in Fortran.

Here is a link to the example that was a starting point for our programs.

https://docs.microsoft.com/en-us/windows/win32/api/sysinfoapi/nf-sysinfoapi-globalmemorystatusex

14.1.1 Sample output

Here is some sample output.

```
 ch4303_intel.exe
 Intel(R) Fortran Intel(R) 64 Compiler Classic for applications running
on Intel
 (R) 64, Version 2021.5.0 Build 20211109_000000
  Memory usage                        26   %
  Total physical                     17,179,127,808
  Available physical                12,598,317,056
  Total page file                    22,816,272,384
  Available page file               17,433,796,608
  Total virtual                 140,737,488,224,256
  Available virtual            140,733,142,515,712
```

Here is some sample output from the NAG compiler on the same system.

```
 ch4303_nag.exe
 NAG Fortran Compiler Release 7.0(Yurakucho) Build 7017
  Memory usage            27   %
```

```
  Total physical                     17,179,127,808
  Available physical                12,488,970,240
  Total page file                    22,816,272,384
  Available page file               17,327,640,576
  Total virtual                 140,737,488,224,256
  Available virtual           140,733,001,248,768
```

14.1.2 Fortran source file

Here is the Fortran source file.

```
include 'integer_kind_module.f90'
include 'display_with_commas_module.f90'
include 'memory_module_windows.f90'

program ch4303

  use iso_fortran_env
  use memory_module_windows
  use display_with_commas_module

  print *,compiler_version()
  print *,' Memory usage            ',MemoryLoad(),' %'
  print *,' Total physical         ',&
  display_with_commas(TotalPhysical())
  print *,' Available physical    ',&
  display_with_commas(AvailablePhysical())
  print *,' Total page file        ',&
  display_with_commas(TotalPageFile())
  print *,' Available page file   ',&
  display_with_commas(AvailablePageFile())
  print *,' Total virtual          ',&
  display_with_commas(TotalVirtual())
  print *,' Available virtual     ',&
  display_with_commas(AvailableVirtual())

end program ch4303
```

14.1.3 C source file

You also require the following C source file

`ch4303_memory_module_windows.c`

which is shown below.

```
#include <windows.h>

int memory_load()
{
  MEMORYSTATUSEX statex;
  statex.dwLength = sizeof (statex);
  GlobalMemoryStatusEx (&statex);
```

```
  return(statex.dwMemoryLoad);
}

long long int total_physical()
{
  long long int t;

  MEMORYSTATUSEX statex;
  statex.dwLength = sizeof (statex);
  GlobalMemoryStatusEx (&statex);

  t=statex.ullTotalPhys;
  return(t);
}


long long int available_physical()
{
  long long int t;
  MEMORYSTATUSEX statex;
  statex.dwLength = sizeof (statex);
  GlobalMemoryStatusEx (&statex);
  t=statex.ullAvailPhys;
  return(t);
}

long long int total_page_file()
{
  long long int t;
  MEMORYSTATUSEX statex;
  statex.dwLength = sizeof (statex);
  GlobalMemoryStatusEx (&statex);
  t=statex.ullTotalPageFile;
  return(t);
}

long long int available_page_file()
{
  long long int t;
  MEMORYSTATUSEX statex;
  statex.dwLength = sizeof (statex);
  GlobalMemoryStatusEx (&statex);
  t=statex.ullAvailPageFile;
  return(t);
}

long long int total_virtual()
{
```

```
    long long int t;
    MEMORYSTATUSEX statex;
    statex.dwLength = sizeof (statex);
    GlobalMemoryStatusEx (&statex);

    t=statex.ullTotalVirtual;
    return(t);
}

long long int available_virtual()
{
    long long int t;
    MEMORYSTATUSEX statex;
    statex.dwLength = sizeof (statex);
    GlobalMemoryStatusEx (&statex);

    t=statex.ullAvailVirtual;
    return(t);
}
```

14.2 Chapter 44 example 2: Querying memory availability and usage using the Linux API

Here is a link to the Linux api.

https://man7.org/linux/man-pages/man2/sysinfo.2.html

Here is the struct.

```
struct sysinfo
{
    long uptime;              /* Seconds since boot */
    unsigned long loads[3];   /* 1, 5, and 15 minute load aver-
ages */
    unsigned long totalram;   /* Total usable main memory size
*/
    unsigned long freeram;    /* Available memory size */
    unsigned long sharedram; /* Amount of shared memory */
    unsigned long bufferram; /* Memory used by buffers */
    unsigned long totalswap; /* Total swap space size */
    unsigned long freeswap;   /* Swap space still available */
    unsigned short procs;     /* Number of current processes */
    unsigned long totalhigh; /* Total high memory size */
    unsigned long freehigh;   /* Available high memory size */
    unsigned int mem_unit;    /* Memory unit size in bytes */
    char _f[20-2*sizeof(long)-sizeof(int)];
    /* Padding to 64 bytes */
};
```

14.2.1 C source code

Here is out C code.

```
#include <stdio.h>
#include <sys/sysinfo.h>

unsigned long total_ram()
{
  struct sysinfo si;
  sysinfo (&si);
    return( si.totalram ) ;
}

unsigned long free_ram()
{
  struct sysinfo si;
  sysinfo (&si);
    return( si.freeram ) ;
}

unsigned long shared_ram()
{
  struct sysinfo si;
  sysinfo (&si);
    return( si.sharedram ) ;
}

unsigned long buffer_ram()
{
  struct sysinfo si;
  sysinfo (&si);
    return( si.bufferram ) ;
}

unsigned long total_swap()
{
  struct sysinfo si;
  sysinfo (&si);
    return( si.totalswap ) ;
}

unsigned long free_swap()
{
  struct sysinfo si;
  sysinfo (&si);
    return( si.freeswap ) ;
}
unsigned long total_high()
{
  struct sysinfo si;
  sysinfo (&si);
```

```
    return( si.totalhigh ) ;
}

unsigned long free_high()
{
  struct sysinfo si;
  sysinfo (&si);
    return( si.freehigh ) ;
}
```

14.2.2 Fortran C interop code

Here is our Fortran C interop code.

```
module memory_module_linux

  use :: iso_c_binding
  use :: integer_kind_module

contains

! 1

  function totalram()

    use :: iso_c_binding

    interface
      function total_ram() bind (c, name='total_ram')
        use :: integer_kind_module
        integer (i64) :: total_ram
      end function total_ram
    end interface

    integer (c_long_long) :: totalram

    totalram = total_ram()

  end function totalram

! 2

  function freeram()

    use :: iso_c_binding

    interface
      function free_ram() bind (c, name='free_ram')
        use :: integer_kind_module
        integer (i64) :: free_ram
```

```
      end function free_ram
    end interface

    integer (c_long_long) :: freeram

    freeram = free_ram()

  end function freeram

! 3

  function sharedram()

    use :: iso_c_binding

    interface
      function shared_ram() bind (c, name='shared_ram')
        use :: integer_kind_module
        integer (i64) :: shared_ram
      end function shared_ram
    end interface

    integer (c_long_long) :: sharedram

    sharedram = shared_ram()

  end function sharedram

! 4

  function bufferram()

    use :: iso_c_binding

    interface
      function buffer_ram() bind (c, name='buffer_ram')
        use :: integer_kind_module
        integer (i64) :: buffer_ram
      end function buffer_ram
    end interface

    integer (c_long_long) :: bufferram

    bufferram = buffer_ram()

  end function bufferram

! 5
```

```
  function totalswap()

    use :: iso_c_binding

    interface
      function total_swap() bind (c, name='total_swap')
        use :: integer_kind_module
        integer (i64) :: total_swap
      end function total_swap
    end interface

    integer (c_long_long) :: totalswap

    totalswap = total_swap()

  end function totalswap

! 6

  function freeswap()

    use :: iso_c_binding

    interface
      function free_swap() bind (c, name='free_swap')
        use :: integer_kind_module
        integer (i64) :: free_swap
      end function free_swap
    end interface

    integer (c_long_long) :: freeswap

    freeswap = free_swap()

  end function freeswap

! 7

  function totalhigh()

    use :: iso_c_binding

    interface
      function total_high() bind (c, name='total_high')
        use :: integer_kind_module
        integer (i64) :: total_high
      end function total_high
```

```
    end interface

    integer (c_long_long) :: totalhigh

    totalhigh = total_high()

  end function totalhigh

! 8

  function freehigh()

    use :: iso_c_binding

    interface
      function free_high() bind (c, name='free_high')
        use :: integer_kind_module
        integer (i64) :: free_high
      end function free_high
    end interface

    integer (c_long_long) :: freehigh

    freehigh = free_high()

  end function freehigh

end module
```

14.2.3 Fortran test program

Here is the driving program.

```
include 'integer_kind_module.f90'
include 'ch4304_memory_module_linux.f90'
include 'display_with_commas_module.f90'

program ch4304

  use iso_fortran_env
  use memory_module_linux
  use display_with_commas_module

  print *,compiler_version()
  print *,' Total      ram   ',totalram()   ,' ',dis-
play_with_commas(totalram())
  print *,' Free       ram   ',freeram()    ,' ',dis-
play_with_commas(freeram())
  print *,' Share      ram   ',sharedram()  ,' ',dis-
play_with_commas(sharedram())
```

```
  print *,' Buffer     ram   ',bufferram() ,' ',dis-
play_with_commas(bufferram())
  print *,' Total      swap ',totalswap() ,' ',dis-
play_with_commas(totalswap())
  print *,' Free       swap ',freeswap()  ,' ',dis-
play_with_commas(freeswap())
  print *,' Total      high ',totalhigh() ,' ',dis-
play_with_commas(totalhigh())
  print *,' Free       high ',freehigh()  ,' ',dis-
play_with_commas(freehigh())
end program ch4304
```

14.2.4 Sample compile script

Here is the gnu Fortran compile script.

```
gcc -c ch4304_memory_module_linux.c
    -o ch4304_memory_module_linux.o
gfortran ch4304.f90 ch4304_memory_module_linux.o
    -o ch4304.out
```

14.2.5 Sample output

Here are some sample outputs. The first 4 are on the same native Ubuntu installation.

```
 GCC version 9.4.0
  Total      ram          67130130432            67,130,130,432
  Free       ram          62834323456            62,834,323,456
  Share      ram             26738688                26,738,688
  Buffer     ram             73711616                73,711,616
  Total      swap          2147479552             2,147,479,552
  Free       swap          2147479552             2,147,479,552
  Total      high                   0                         0
  Free       high                   0                         0
 Intel(R) Fortran Intel(R) 64 Compiler Classic for applica-
tions running on Intel
 (R) 64, Version 2021.8.0 Build 20221119_000000
  Total      ram          67130130432            67,130,130,432
  Free       ram          62828130304            62,828,130,304
  Share      ram             26738688                26,738,688
  Buffer     ram             73719808                73,719,808
  Total      swap          2147479552             2,147,479,552
  Free       swap          2147479552             2,147,479,552
  Total      high                   0                         0
  Free       high                   0                         0
 NAG Fortran Compiler Release 7.1(Hanzomon) Build 7114
  Total      ram   67130130432           67,130,130,432
  Free       ram   62823251968           62,823,251,968
  Share      ram   26738688                  26,738,688
  Buffer     ram   73732096                  73,732,096
  Total      swap  2147479552             2,147,479,552
  Free       swap  2147479552             2,147,479,552
  Total      high  0                                  0
  Free       high  0                                  0
 nvfortran 22.5-0
  Total      ram               67130130432           67,130,130,432
  Free       ram               62817316864           62,817,316,864
  Share      ram                  26435584               26,435,584
  Buffer     ram                  73748480               73,748,480
  Total      swap               2147479552            2,147,479,552
  Free       swap               2147479552            2,147,479,552
  Total      high                        0                        0
  Free       high                        0                        0
```

The next one is the same system as the previous, but using openSuSe Timbleweed under WSL.

```
 Intel(R) Fortran Intel(R) 64 Compiler Classic for applica-
tions running on Intel
 (R) 64, Version 2021.8.0 Build 20221119_000000
  Total      ram             68412305408              68,412,305,408
  Free       ram             61039616000              61,039,616,000
  Share      ram                       0                           0
  Buffer     ram                       0                           0
  Total      swap           127830847488             127,830,847,488
  Free       swap           127830847488             127,830,847,488
  Total      high              142548992                 142,548,992
  Free       high                 278528                     278,528
```

The next one is on the same system, using Redhat 9 under hyper-v.

```
Intel(R) Fortran Intel(R) 64 Compiler Classic for applica-
tions running on Intel
 (R) 64, Version 2021.8.0 Build 20221119_000000
  Total      ram             26157993984              26,157,993,984
  Free       ram             24257781760              24,257,781,760
  Share      ram                34058240                  34,058,240
  Buffer     ram                 2965504                   2,965,504
  Total      swap             8451518464               8,451,518,464
  Free       swap             8451518464               8,451,518,464
  Total      high                      0                           0
  Free       high                      0                           0
```

This system has been hard coded to have a subset of the total physical ram.

14.3 Chapter 44 example 3: Kahan summation with memory usage - Windows

Here is the Fortran source for ch4305.

```
include 'integer_kind_module.f90'
include 'precision_module.f90'
include 'timing_module.f90'

include 'display_with_commas_module.f90'

include 'kahan_summation_module.f90'
include 'memory_module_windows.f90'

program ch4305

  use timing_module
  use precision_module

  use kahan_summation_module

  use memory_module_windows
  use display_with_commas_module

  implicit none

  integer (i64) :: n = 10000000_i64
  integer :: I
  integer :: j = 4
```

```
  integer (i64) , parameter :: sixty_four_bit=8_i64
  integer (i64) :: nbytes

  real (dp), allocatable, dimension (:) :: x
  real (dp)                                  :: x_sum =
0.0_dp

  character (len=20) :: heading = 'call memory usage    '
  integer :: lu=6
  call start_timing()

  do i=1,j

    nbytes=n*sixty_four_bit
    print *,' Problem size'
    print *, display_with_commas(n)
    print *, display_with_commas(nbytes),' bytes'

    if ( AvailablePhysical() < nbytes ) then
      print *,' Insufficient memory '
      print *,' Memory usage          ',&
                MemoryLoad(),' %'
      print *,' Total physical        ',&
     display_with_commas(TotalPhysical())
      print *,' Available physical    ',&
     display_with_commas(AvailablePhysical())
      print *,' Program terminates'
      stop 20
    end if

    allocate(x(n))

    print 20,time_difference()
    20 format(' Allocate              ',f22.18)

    call random_number(x)

    print 30,time_difference()
    30 format(' Initialise            ',f22.18)

    x_sum=sum(x)

    print 40, time_difference()
    40 format(' Intrinsic summation ',f22.18)

    print 100, x_sum
    100 format(45x,f20.10)
```

```
      x_sum=kahan_sum(x,n)

      print 50, time_difference()
      50 format(' Kahan summation      ',f22.18)

      print 100, x_sum

      print *,' Memory usage           ',&
                 MemoryLoad(),' %'
      print *,' Total physical        ',
      display_with_commas(TotalPhysical())
      print *,' Available physical    ',&
      display_with_commas(AvailablePhysical())

      deallocate(x)

      n=n*10_i64

    end do

    call end_timing()

end program ch4305
```

14.3.1 Sample output

Here is some sample output.

```
14:43:14 D:\fortran_web_site >  ch4305_intel
 2022/ 5/ 5 14:43:19 474
  Problem size
               10,000,000
               80,000,000 bytes
 Allocate                0.015000104904174805
 Initialise              0.167999982833862305
 Intrinsic summation   0.008999824523925781

5000444.2793215252
 Kahan summation        0.047000169754028320

5000444.2793215429
  Memory usage                     20  %
  Total physical                  17,179,127,808
  Available physical              13,604,413,440
  Problem size
              100,000,000
              800,000,000 bytes
 Allocate                0.014999866485595703
 Initialise              1.644000053405761719
 Intrinsic summation   0.108999967575073242
```

```
49998117.4713004455
 Kahan summation        0.535000085830688477

49998117.4712983146
   Memory usage                      24   %
   Total physical                   17,179,127,808
   Available physical               12,886,278,144
   Problem size
              1,000,000,000
              8,000,000,000 bytes
 Allocate                0.0479999858856201172
 Initialise             16.6390001773834222852
 Intrinsic summation    1.21899986267089843 8

499995574.2241585851
 Kahan summation        5.355000019073486328

499995574.2241371870
   Memory usage                      66   %
   Total physical                   17,179,127,808
   Available physical                5,768,028,160
   Problem size
             10,000,000,000
             80,000,000,000 bytes
   Insufficient memory
   Memory usage                      19   %
   Total physical                   17,179,127,808
   Available physical               13,841,661,952
   Program terminates
20
```

We can use the memory functions to detect that there is insufficient memory to make the allocation and terminate the program, providing memory usage figures.

14.4 Chapter 44 example 4: Kahan summation with memory usage: Linux

Here is the source code.

```
include 'integer_kind_module.f90'
include 'precision_module.f90'
include 'timing_module.f90'

include 'display_with_commas_module.f90'

include 'kahan_summation_module.f90'
include 'memory_module_linux.f90'

program ch4306
```

```
  use timing_module
  use precision_module

  use kahan_summation_module

  use memory_module_linux
  use display_with_commas_module

  implicit none

  integer (i64) :: n = 10000000_i64
  integer :: I
  integer :: j = 4
  integer (i64) , parameter :: sixty_four_bit=8_i64
  integer (i64) :: nbytes

  real (dp), allocatable, dimension (:) :: x
  real (dp)                                  :: x_sum =
0.0_dp

  character (len=20) :: heading = 'call memory usage    '
  integer :: lu=6
  call start_timing()

  do i=1,j

    nbytes=n*sixty_four_bit
    print *,' Problem size'
    print *, display_with_commas(n)
    print *, display_with_commas(nbytes),' bytes'

    if ( freeram() < nbytes ) then
      print *,' Insufficient memory '
      print *,' Number of bytes =     ' , display_with_com-
mas(nbytes)
      print *,' Total physical         ' , display_with_com-
mas(freeram())
      print *,' Available physical    ' , display_with_com-
mas(totalram())
      stop 20
    end if

    allocate(x(n))

    print 20,time_difference()
    20 format(' Allocate                ',f22.18)
```

```
      call random_number(x)

      print 30,time_difference()
      30 format(' Initialise             ',f22.18)

      x_sum=sum(x)

      print 40, time_difference()
      40 format(' Intrinsic summation ',f22.18)

      print 100, x_sum
      100 format(45x,f20.10)

      x_sum=kahan_sum(x,n)

      print 50, time_difference()
      50 format(' Kahan summation      ',f22.18)

      print 100, x_sum

      print *,' Total physical         ',display_with_com-
mas(freeram())
      print *,' Available physical    ',display_with_com-
mas(totalram())

      deallocate(x)

      n=n*10_i64

   end do

   call end_timing()

end program ch4306
```

14.4.1 Sample output

Here is some sample output.

```
./ch4306_gnu.out
 2022/ 5/ 5 14:49:30 381
  Problem size
                 10,000,000
                 80,000,000 bytes
 Allocate                 0.000101700000001870
 Initialise               0.115071700000001442
 Intrinsic summation    0.015628499999998269
                                                        5000669.2088570781
 Kahan summation        0.055942299999998113
                                                        5000669.2088573920
  Total physical                  13,136,986,112
  Available physical             13,393,960,960
```

```
  Problem size
              100,000,000
              800,000,000 bytes
 Allocate              0.000668300000000954
 Initialise            1.110655600000001186
 Intrinsic summation   0.149585000000001855
                                                   49997221.2586892843
 Kahan summation       0.528356199999997500
                                                   49997221.2587036043
  Total physical                12,415,856,640
  Available physical            13,393,960,960
  Problem size
            1,000,000,000
            8,000,000,000 bytes
 Allocate              0.003667399999997656
 Initialise           12.188676499999999692
 Intrinsic summation   1.512021000000004278
                                                  500006058.1260393262
 Kahan summation       5.293246199999998680
                                                  500006058.1257698536
  Total physical                 5,201,567,744
  Available physical            13,393,960,960
  Problem size
           10,000,000,000
           80,000,000,000 bytes
  Insufficient memory
  Number of bytes =             80,000,000,000
  Total physical                13,216,075,776
  Available physical            13,393,960,960
STOP 20
```

14.5 Chapter 44 example 5: Modified memory leak example with memory checking - Windows

Here is the Fortran source.

```
include 'integer_kind_module.f90'
include 'memory_module_windows.f90'
include 'display_with_commas_module.f90'


! Update of ch1806 to give a diagnostic information
! about the run time memory behaviour
! of the program

program ch4307

  use iso_fortran_env

  use integer_kind_module
  use memory_module_windows
  use display_with_commas_module

!
! This is a variation on
```

```
! the pointer example in chapter
! 18 that has a memory leak.
!
! We use the memory query functions to provide
! warning messages as memory usage goes up.
!

  implicit none

  integer (i64)                :: n = 100000000_i64
  integer (i64)                :: I=0

  integer                       :: allocate_status = 0

  integer (i64) , dimension ( : ) , pointer :: x
  integer (i64) , dimension (1:5) , target  :: y

  real                                             :: avail-
able
  real                                             :: physi-
cal
  real                                             :: per-
centage_free

  Print *,' Program starts'
  print *,compiler_version()
  print *,' Memory usage           ',MemoryLoad(),' %'
  print *,' Total physical        ',display_with_com-
mas(TotalPhysical())
  print *,' Available physical    ',display_with_com-
mas(AvailablePhysical())
  print *,' Total page file       ',display_with_com-
mas(TotalPageFile())
  print *,' Available page file   ',display_with_com-
mas(AvailablePageFile())
  print *,' Total virtual          ',display_with_com-
mas(TotalVirtual())
  print *,' Available virtual     ',display_with_com-
mas(AvailableVirtual())

  physical = real(AvailablePhysical())
  print *,' '
  print *,' Loop starts'
  print *,' '

  do

    allocate (x(1:n), stat=allocate_status)
```

```
      if (allocate_status>0) then
        print *, ' allocate failed. program ends.'
        stop
      end if

      x =  1_i64

      y = 10_i64

      x => y !                               x now points to y

      i=I+1

      available = real(AvailablePhysical())

      percentage_free = (available/physical)*100

      if (percentage_free < 5.0) then
        print *,' Memory usage over 95%'
        print *,' Program terminates'
        print *,' Iteration count was ',I
        stop 20
      end if

   end do

end program
```

14.5.1 Sample output

Here is some sample output.

```
ch4307_nag.exe
   Program starts
 NAG Fortran Compiler Release 7.0(Yurakucho) Build 7048
   Memory usage           13  %
   Total physical                      33,663,741,952
   Available physical                 28,979,843,072
   Total page file                     38,764,015,616
   Available page file                32,489,934,848
   Total virtual                  140,737,488,224,256
   Available virtual             140,733,000,802,304

   Loop starts

   Memory usage over 95%
   Program terminates
   Iteration count was  35
STOP: 20
```

14.6 Chapter 44 example 6: Modified memory leak example with memory checking - Linux

Here is the source file.

```
include 'integer_kind_module.f90'
include 'memory_module_linux.f90'
include 'display_with_commas_module.f90'


! Update of ch1806 to give a diagnostic information
! about the run time memory behaviour
! of the program

program ch4308

   use iso_fortran_env

   use integer_kind_module
   use memory_module_linux
   use display_with_commas_module

!
! This is a variation on
! the pointer example in chapter
! 18 that has a memory leak.
!
! We use the memory query functions to provide
! warning messages as memory usage goes up.
!

   implicit none

   integer (i64)                :: n = 100000000_i64
   integer (i64)                :: I=0

   integer                       :: allocate_status = 0

   integer (i64) , dimension ( : ) , pointer :: x
   integer (i64) , dimension (1:5) , target  :: y

   real                                                 :: avail-
able
   real                                                 :: physi-
cal
   real                                                 :: per-
centage_free

   Print *,' Program starts'
```

```
  print *,compiler_version()
  print *,' Total ram  ',totalram(),' ',display_with_com-
mas(totalram())
  print *,' Free  ram  ',freeram() ,' ',display_with_com-
mas(freeram())

  physical = real(totalram())
  print *,' '
  print *,' Loop starts'
  print *,' '

  do

    allocate (x(1:n), stat=allocate_status)

    if (allocate_status>0) then
      print *, ' allocate failed. program ends.'
      stop
    end if

    x =  1_i64

    y = 10_i64

    x => y !                            x now points to y

    i=I+1

    available = real(freeram())

    percentage_free = (available/physical)*100

    if (percentage_free < 5.0) then
      print *,' Memory usage over 95%'
      print *,' Program terminates'
      print *,' Iteration count was ',I
      stop 20
    end if

  end do

end program
```

14.6.1 Sample output

Here is some sample output.

```
./ch4308_gnu.out
   Program starts
  GCC version 11.2.1 20220420 [revision
691af15031e00227ba6d5935c1d737026cda4129]
```

```
  Total ram             33663741952                   33,663,741,952
  Free  ram             29230915584                   29,230,993,408

  Loop starts

  Memory usage over 95%
  Program terminates
  Iteration count was                         35
STOP 20
```

14.7 Files and compilation details

Here is a list of the files associated with this chapter.

- ch4401.f90
- ch4401_memory_module_windows.c
- ch4401_memory_module_windows.f90
- ch4402.f90
- ch4402_memory_module_linux.c
- ch4402_memory_module_linux.f90
- ch4402_test.f90
- ch4402_wsl_suse.sh
- ch4403.f90
- ch4403_linux.f90
- ch4403_windows.f90
- ch4404.f90
- ch4405.f90
- ch4406.f90

Where multiple files are involved we have provided batch files and shell scripts to help out.

15 Nvidia HPC toolkit and gpu programming

Both Intel and Nvidia toolkits offer the possibility of developing code that can run on both CPUs and GPUs, i.e. with a system with a cpu and graphics card it is possible to do processing on both the CPU and GPU. In this chapter we look at Nvidia offerings.

15.1 Nvidia Toolkit overview

Nvidia make the following toolkits available.

- Nvidia HPC toolkit
- Nvidia Cuda toolkit

The HPC toolkit can be found at

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

and the Cuda toolkit can be found at

`https://developer.nvidia.com/cuda-downloads`

More detailed coverage is given below.

15.2 Nvidia HPC toolkit

We have used it on a variety of Linux platforms. This is not available currently on a Windows platform.

15.3 Nvidia Cuda toolkit

This is available for both Windows and Linux. We have used it on both platforms.

15.4 Nvidia and GPU programming

In addition to conventional Fortran and C++ programming we are also trying out usage of the GPU, and have started running their examples and writing our own parallel examples.

15.4.1 Nvidia Fortran

For Fortran (using nvfortran) we have tried the following

- native linux
- hyper-v and a linux distro
- wsl and a linux distro
- There is no Windows HPC toolkit at this time.

They all require access via an Nvidia driver to the GPU for the CUDA Fortran examples. The only one we have got to work is a native ubuntu 20.04.4 install on the Dell T5820.

The compiler can also be used as a plain Fortran compiler.

We have had a number of problems with different versions of the sdk for general purpose Fortran programming with our fourth edition examples. We have used the following versions

- 21.3
- 21.9
- 21.11

- 22.3
- 22.5
- 22.7
- 22.11

Version 22.5 compiles most of the examples from the book.

The following are general compilation messages, which apply to all versions:

- a warning about detection of integer overflow;
- kind type errors with 128 bit reals - Nvidia support 32 and 64 bit reals only;
- no support for allocatable components;
- C compilation errors due to lack of conformance to the latest C standards;

These diagnostic messages are not a real issue.

We get problems with malloc and loader warnings with the 21.3 version.

We get illegal instruction generation with the 21.9 and 21.11 versions on an AMD hardware platform.

We am currently using the following system setups:

- Dell T5280, Nvidia graphics card, Quadro RTX 4000 with 8GB of RAM
    - ubuntu 20.04.4, native install, 22.5 - works;
    - openSuSe 15.3, hyper-v install, 22.5;
    - ubuntu 20.04.4, wsl install, 22.5;
- Dell 5515, Intel graphics card
    - openSuSe 15.3, hyper-v, 22.3;
    - ubuntu 20.04.4, wsl, 22.5;
- Dell 7100, Nvidia graphics card, Geforce GTX 750 GTi
    - openSuSe 15.3, native install, 22.7, illegal instruction error messages when running the executables;
    - ubuntu 20.04.4, wsl, 22.5;

Obviously if you don't have an Nvidia graphics card you can't run and try out some of the GPU examples.

15.5 Parallel programming and Cuda Fortran

The following information has been taken from the Cuda Fortran Programming guide. The online version is available at:

```
https://docs.nvidia.com/hpc-sdk/compilers/
cuda-fortran-prog-guide/index.html#abstract
```

- Graphic processing units or GPUs have evolved into programmable, highly parallel computational units with very high memory bandwidth, and tremendous potential for many applications. GPU designs are optimized for the computations found in graphics rendering, but are general enough to be useful in many data-parallel, compute-intensive programs.

- NVIDIA introduced CUDA®, a general purpose parallel programming architecture, with compilers and libraries to support the programming of NVIDIA GPUs. CUDA comes with an extended C compiler, here called CUDA C, allowing direct programming of the GPU from a high level language. The programming model supports four key abstractions: cooperating threads organized into thread groups, shared memory and barrier synchronization within thread groups, and coordinated independent thread groups organized into a grid. A CUDA programmer must partition the program into coarse grain blocks that can be executed in parallel. Each block is partitioned into fine grain threads, which can cooperate using shared memory and barrier synchronization. A properly designed CUDA program will run on any CUDA-enabled GPU, regardless of the number of available processor cores.
- CUDA Fortran includes a Fortran 2003 compiler and tool chain for programming NVIDIA GPUs using Fortran. NVIDIA 2022 includes support for CUDA Fortran on Linux. CUDA Fortran is an analog to NVIDIA's CUDA C compiler. Compared to the NVIDIA OpenACC directives-based model and compilers, CUDA Fortran is a lower-level explicit programming model with substantial runtime library components that give expert programmers direct control of all aspects of GPGPU programming.

This is from the 22.11 version.

The following has been taken from CUDA Fortran for Scientists and Engineers, and here is a link to the book.

```
https://www.elsevier.com/books/
cuda-fortran-for-scientists-and-engineers/ruetsch/978-0-12-416970-8
```

- .... A few characteristics of the CUDA programming model are very different from cpu based parallel programming models. One difference is that there is very little overhead associated with creating gpu based threads. In addition to fast thread creation, context switches, where threads change state from active to inactive and vice versa, are very fast for gpu threads based to cpu threads. .... In the CUDA programming model, we essentially write a serial code that is executed by many gpu based threads in parallel. Each thread executing this code has a means of identifying itself in order to operate on different data, but the code that CUDA threads execute is very similar to what we would write for serial CPU code......

The book is available in both printed and electronic versions. Essential complement to the PGI and Nvidia provided documentation.

15.6 Basic steps involved in CUDA Fortran programming

The following 6 steps are involved in Cuda Fortran programming.

- Initialize and select the GPU to run on. Often this is implicit in the program and defaults to NVIDIA device 0.
- Allocate space for data on the GPU.
- Move data from the host to the GPU, or in some cases, initialize the data on the GPU.
- Launch kernels from the host to run on the GPU.

- Gather results back from the GPU for further analysis our output from the host program.
- Deallocate the data on the GPU allocated in step 2. This might be implicitly performed when the host program exits.

CUDA Fortran allows the definition of Fortran subroutines that execute in parallel on the GPU when called from the Fortran program which has been invoked and is running on the host or, starting in CUDA 5.0, on the device. Such a subroutine is called a device kernel or kernel.

A call to a kernel specifies how many parallel instances of the kernel must be executed; each instance will be executed by a different CUDA thread. The CUDA threads are organized into thread blocks, and each thread has a global thread block index, and a local thread index within its thread block.

Device sub programs have access to block and grid indices and dimensions through several built-in read-only variables. These variables are of type dim3; the module cudafor defines the derived type dim3 as follows:

```
type(dim3)
     integer(kind=4) :: x,y,z
end type
```

These predefined variables are not accessible in host subprograms.

- The variable `threadidx` contains the thread index within its thread block; for one- or two-dimensional thread blocks, the `threadidx%y` and/or `threadidx%z` components have the value one.
- The variable `blockdim` contains the dimensions of the thread block; `blockdim` has the same value for all thread blocks in the same grid.
- The variable `blockidx` contains the block index within the grid; as with threadidx, for one-dimensional grids, `blockidx%y` and/or `blockidx%z` has the value one.
- The variable `griddim` contains the dimensions of the grid.
- The constant warpsize is declared to be type integer. Threads are executed in groups of 32, called warps; warpsize contains the number of threads in a warp, and is currently 32.

The examples that follow typically use

- threadidx
- blockdim
- blockidx

in the code.

15.6.1 Execution Configuration

A call to a kernel subroutine must specify an execution configuration. The execution configuration defines the dimensionality and extent of the grid and thread blocks that execute the subroutine. It may also specify a dynamic shared memory extent, in bytes, and a stream identifier, to support concurrent stream execution on the device.

A kernel subroutine call looks like this:


```
call kernel<<<grid,block[,bytes][,streamid]>>>(arg1,arg2,...)
```

where

- grid and block are either integer expressions (for one-dimensional grids and thread blocks), or are type(dim3), for one- or two-dimensional grids and thread blocks.
- If grid is type(dim3), the value of each component must be equal to or greater than one, and the product is usually limited by the compute capability of the device.
- If block is type(dim3), the value of each component must be equal to or greater than one, and the product of the component values must be less than or equal to 1024.
- The value of bytes must be an integer; it specifies the number of bytes of shared memory to be allocated for each thread block, in addition to the statically allocated shared memory. This memory is used for the assumed-size shared variables in the thread block; refer to Shared data for more information. If the value of bytes is not specified, its value is treated as zero.
- The value of streamid must be an integer greater than or equal to zero; it specifies the stream to which this call is associated. Nonzero stream values can be created with a call to cudaStreamCreate. Starting in CUDA 7.0, the constant cudaStreamPerThread can be specified to use a unique default stream for each CPU thread.

We will illustrate the above in the examples.

15.6.2 Thread Blocks

Each thread is assigned a thread block index accessed through the built-in blockidx variable, and a thread index accessed through threadidx. The thread index may be a one-, two-, or three-dimensional index. In CUDA Fortran, the thread index for each dimension starts at one.

Threads in the same thread block may cooperate by using shared memory, and by synchronizing at a barrier using the SYNCTHREADS() intrinsic. Each thread in the block waits at the call to SYNCTHREADS() until all threads have reached that call. The shared memory acts like a low-latency, high bandwidth software managed cache memory. Currently, the maximum number of threads in a thread block is 1024.

A kernel may be invoked with many thread blocks, each with the same thread block size. The thread blocks are organized into a one-, two-, or three-dimensional grid of blocks, so each thread has a thread index within the block, and a block index within the grid. When invoking a kernel, the first argument in the chevron <<<>>> syntax is the grid size, and the second argument is the thread block size. Thread blocks must be able to execute independently; two thread blocks may be executed in parallel or one after the other, by the same core or by different cores.

The dim3 derived type, defined in the cudafor module, can be used to declare variables in host code which can conveniently hold the launch configuration values if they are not scalars; for example:

```
type(dim3) :: blocks, threads
...
```

```
blocks = dim3(n/256, n/16, 1)
threads = dim3(16, 16, 1)
call devkernel<<<blocks, threads>>>( ... )
```

15.6.3 Mapping data onto threads

In this section we show how data in an array is assigned to a thread and thread block.

If we assume an array of size 12, and 3 thread blocks and 4 threads per block using the following equation

```
       I = (blockidx%x-1) * blockDim%x + threadidx%x
```

we have the following mapping

```
Array index | 1 2 3 4 | 5 6 7 8 | 9 10 11 12 |
blockidx%x  | 1 1 1 1 | 2 2 2 2 | 3  3  3 3 |
threadidx%x | 1 2 3 4 | 1 2 3 4 | 1  2  3 4 |
blockDim%x  | 4 4 4 4 | 4 4 4 4 | 4  4  4 4 |
I value     | 1 2 3 4 | 5 6 7 8 | 9 10 11 12 |
```

and the examples will use this equation and variables to organise the mapping between data and threads.

15.7 Chapter 45 example 1: basic device driver test program

This program is provided by Nvidia and tests out access to the GPU. This should be the first program you try out. We've made the following changes

- added implicit none
- added a device test to see if the device query worked - it is possible to run this on a system without access to a GPU. The original version generated a 684,306 line file on one system!
- minor layout changes to make it easier to read

Here is the modified source.

```
!
! An example of getting device
! properties in CUDA Fortran
!
! Build with
!
!   nvfortran ch4501.cuf
!
! Rewritten to test for the return
! status of the device query.
!
! Running the original version generated
! a 684,306 line file.
!
! Also added implicit none
!
program ch4501
```

```
   use cudafor
   implicit none

   integer istat, num, numdevices
   type(cudadeviceprop) :: prop
     istat = cudaGetDeviceCount(numdevices)
!
! Test the status to check things have worked
!
     if (istat /=0) then
       print *,' istat = ',istat
       print *,' numdevices = ',numdevices
       print *,' Error in cudaGetDeviceCount'
       print *,' Program terminates'
       stop 10
     end if
!
     do num = 0, numdevices-1
       istat = cudaGetDeviceProperties(prop, num)
       call printDeviceProperties(prop, num)
     end do
end program
!
subroutine printDeviceProperties(prop, num)
   use cudafor
   implicit none
   type(cudadeviceprop) :: prop
   integer :: num
   integer :: ilen
     ilen = verify(prop%name, ' ', .true.)
     write (*,900) "Device Number:                    "
,num
     write (*,901) "Device Name:                      "
,prop%name(1:ilen)
     write (*,903) "Total Global Memory:            "
,real(prop%totalGlobalMem)/1e9," Gbytes"
     write (*,902) "sharedMemPerBlock:              "
,prop%sharedMemPerBlock," bytes"
     write (*,900) "regsPerBlock:                     "
,prop%regsPerBlock
     write (*,900) "warpSize:                          "
,prop%warpSize
     write (*,900) "maxThreadsPerBlock:             "
,prop%maxThreadsPerBlock
     write (*,904) "maxThreadsDim:                    "
,prop%maxThreadsDim
     write (*,904) "maxGridSize:                       "
,prop%maxGridSize
```

```
     write (*,903) "ClockRate:                         "
,real(prop%clockRate)/1e6," GHz"
     write (*,902) "Total Const Memory:              "
,prop%totalConstMem," bytes"
     write (*,905) "Compute Capability Revision: "
,prop%major,prop%minor
     write (*,902) "TextureAlignment:                "
,prop%textureAlignment," bytes"
     write (*,906) "deviceOverlap:                    "
,prop%deviceOverlap
     write (*,900) "multiProcessorCount:            "
,prop%multiProcessorCount
     write (*,906) "integrated:                        "
,prop%integrated
     write (*,906) "canMapHostMemory:                 "
,prop%canMapHostMemory
     write (*,906) "ECCEnabled:                        "
,prop%ECCEnabled
     write (*,906) "UnifiedAddressing:                "
,prop%unifiedAddressing
     write (*,900) "L2 Cache Size:                     "
,prop%l2CacheSize
     write (*,900) "maxThreadsPerSMP:                 "
,prop%maxThreadsPerMultiProcessor
     900 format (a,i0)
     901 format (a,a)
     902 format (a,i0,a)
     903 format (a,f5.3,a)
     904 format (a,2(i0,1x,'x',1x),i0)
     905 format (a,i0,'.',i0)
     906 format (a,l0)
     return
end subroutine
```

Here is the output on a Dell 5820 system with an Nvidia Quadro RTX GPU. This is using a native Ubuntu installation.

```
Device Number:                 0
Device Name:                   Quadro RTX 4000
Total Global Memory:          8.347 Gbytes
sharedMemPerBlock:            49152 bytes
regsPerBlock:                  65536
warpSize:                      32
maxThreadsPerBlock:           1024
maxThreadsDim:                 1024 x 1024 x 64
maxGridSize:                   2147483647 x 65535 x 65535
ClockRate:                     1.545 GHz
Total Const Memory:           65536 bytes
Compute Capability Revision: 7.5
TextureAlignment:             512 bytes
deviceOverlap:                 1
multiProcessorCount:          36
```



```
integrated:                    0
canMapHostMemory:            1
ECCEnabled:                    0
UnifiedAddressing:           1
L2 Cache Size:                4194304
maxThreadsPerSMP:            1024
```

Here is the output on the same system from Ubuntu 22.04 under WSL.

```
 istat =             35
 numdevices =          32529
 Error in cudaGetDeviceCount
 Program terminates
  10
```

The program can't access the GPU from the Windows Subsystem for Linux.

15.7.1 Nvidia Quadro RTX GPU properties

Some of the key properties are

- Total Global Memory: 8.347 Gbytes
- maxThreadsPerBlock: 1024
- maxThreadsDim: 1024 x 1024 x 64
- maxGridSize: 2147483647 x 65535 x 65535
- Compute Capability Revision: 7.5
- multiProcessorCount: 36

and we will using the above information in the examples that follow.

15.8 Chapter 45 example 2: gpu and cpu computation, 32 bit integers

This example is based on example 2.12.2 in the Fortran Cuda Programming guide.

We've made some changes:

- added a precision module - we've had to modify our standard precision module to work with the Nvidia compiler. They only support 32 and 64 bit reals on the CPU at this time.
- added a timing module - we can provide timing information about the execution of the program. We also modified the base timing module to check the count characteristics of the nvfortran compiler.

Here is the new source code.

```
include 'integer_kind_module.f90'
include 'nvidia_precision_module.f90'
include 'timing_module.f90'
!
! The basis for the example is 2.12.2 in
! the Cuda Fortran Programming Guide
!
module initialise_array
  use integer_kind_module
  contains
```

```
  attributes (device) subroutine initialise(z)
    implicit none
    integer (i32) , dimension(:) , device :: z
    integer                                      :: I
      I = (blockidx%x-1) * blockdim%x + threadidx%x
      z(i)=I
  end subroutine

end module
module calculate
  use integer_kind_module
  use initialise_array
  implicit none

  contains

    attributes (global) subroutine Kernel(x)

      implicit none
      integer (i32) , dimension(:) , device :: x

        call initialise(x)

    end subroutine

    function device_summation(x)

      implicit none
      integer (i32)             :: device_summation
      integer (i32) , &
        dimension(:) , device :: x
      integer (i32)             :: total
      integer                    :: I
        total = 0
        !$cuf kernel do <<< * , * >>>
        do I = 1 , size(x)
          total = total + x(I)
        end do

        device_summation = total

    end function

end module
program test
  use integer_kind_module
  use precision_module , wp => dp
```



```
  use timing_module
  use calculate

  use cudafor

  implicit none

  integer                                :: n
  integer (i32) , dimension(:) , &
    allocatable , device              :: x
  integer (i32) , dimension(:) , &
    allocatable                         :: y
  integer (i32)                         :: cpu_sum
  integer (i32)                         :: device_sum

  integer                                :: I
  integer                                :: allocation_status

  integer                                :: threads_per_block
  integer                                :: thread_blocks

  integer                                :: loop_count
  integer                                :: ierrSync
  integer                                :: ierrAsync
  integer                                :: istat

  real (wp) , dimension(20,8)        :: timing_figures
  real (wp)                              :: t

! the loop_count value depends on whether
! we are dealing with 32 or 64 bit data items.
! set up 20 to work with both 32 and 64 bit data
    call start_timing()

    print *,'   Thread     Threads               N
Sum           Time'
    print *,'   blocks     per block'
    allocation_status = 0
    threads_per_block = 1024
    n                   =     1 * 1024
    loop_count = 20

    do I = 1 , loop_count

      thread_blocks      =   n/threads_per_block
      cpu_sum=0

      allocate(x(n),stat=allocation_status)
```

```
        if (allocation_status > 0) then
          print *,' Device allocation failed'
          print *,' N = ',n
          print *,' Program terminates'
          stop 10
        end if

        t = time_difference()
        timing_figures(i,1) = t

        allocate(y(n),stat=allocation_status)
        if (allocation_status > 0) then
          print *,' CPU allocation failed'
          print *,' N = ',n
          print *,' Program terminates'
          stop 20
        end if

        t = time_difference()
        timing_figures(i,2) = t
        x=0

        t = time_difference()
        timing_figures(i,3) = t

        y=x
        t = time_difference()
        timing_figures(i,4) = t

        call Kernel<<< thread_blocks  , threads_per_block
>>>(x)

        ierrSync  = cudaGetLastError()
        ierrAsync = cudaDeviceSynchronize()
        if ( ierrSync /= cudaSuccess ) then
          write (* ,*) ' Sync kernel error : ' ,
cudaGetErrorString( ierrSync )
        end if
        if ( ierrAsync /= cudaSuccess ) then
          write (* ,*) ' Async kernel error : ' ,
cudaGetErrorString ( ierrAsync )
        end if
        istat = cudaDeviceSynchronize ()

        device_sum = device_summation(x)
        t = time_difference()
        timing_figures(i,7) = t
```



```
        print 20,thread_blocks,threads_per_block,n,device_sum,t
        20 format(2x,i8,2x,i6,6x,i10,2x,i20,2x,f10.7)

        y=x

        t = time_difference()
        timing_figures(i,4) = t
        cpu_sum=sum(y)
        t = time_difference()
        timing_figures(i,8) = t
        print 30,cpu_sum,t
        30 format(36x,i20,2x,f10.7)

        deallocate(x)
        t = time_difference()
        timing_figures(i,5) = t

        deallocate(y)
        t = time_difference()
        timing_figures(i,6) = t
        n = n * 2

      end do

    call end_timing()
      print *,'  Allocate                     Assign
Deallocate                 Summation'
!            1234567890123456789012345678901234567890
    print *,'        gpu           cpu           gpu
cpu          gpu           cpu           gpu           cpu'
    do I=1,20
      print 40,timing_figures(i,1:8)
      40 format(8(2x,f10.7))
    end do
end program
```

Here is the output.

```
 2022/11/25  7:28:22   0
     Thread    Threads             N                  Sum          Time
     blocks    per block
          1     1024            1024               524800   0.0000730
                                                     524800   0.0000000
          2     1024            2048              2098176   0.0000331
                                                    2098176   0.0000010
          4     1024            4096              8390656   0.0000291
                                                    8390656   0.0000019
          8     1024            8192             33558528   0.0000282
                                                   33558528   0.0000029
         16     1024           16384            134225920   0.0000282
                                                  134225920   0.0000060
         32     1024           32768            536887296   0.0000279
```

```
                                                  536887296    0.0000109
        64     1024             65536        -2147450880    0.0000360
                                                -2147450880    0.0000219
       128     1024            131072              65536    0.0000410
                                                      65536    0.0000441
       256     1024            262144             131072    0.0000520
                                                     131072    0.0000899
       512     1024            524288             262144    0.0000880
                                                     262144    0.0001672
      1024     1024           1048576             524288    0.0000961
                                                     524288    0.0003421
      2048     1024           2097152            1048576    0.0001459
                                                    1048576    0.0006330
      4096     1024           4194304            2097152    0.0002570
                                                    2097152    0.0012550
      8192     1024           8388608            4194304    0.0004442
                                                    4194304    0.0024871
     16384     1024          16777216            8388608    0.0006640
                                                    8388608    0.0050099
     32768     1024          33554432           16777216    0.0011070
                                                   16777216    0.0095861
     65536     1024          67108864           33554432    0.0021300
                                                   33554432    0.0182519
    131072     1024         134217728           67108864    0.0039990
                                                   67108864    0.0353799
    262144     1024         268435456          134217728    0.0076902
                                                  134217728    0.0718269
    524288     1024         536870912          268435456    0.0141171
                                                  268435456    0.1284690
 2022/11/25  7:28:25 709
 Total time =                          3.708907
```

Allocate		Assign		Deallocate		Summation	
gpu	cpu	gpu	cpu	gpu	cpu	gpu	cpu
0.2893159	0.0000069	0.0000341	0.0000250	0.0000069	0.0000031	0.0000730	0.0000000
0.0000041	0.0000009	0.0000038	0.0000100	0.0000040	0.0000000	0.0000331	0.0000010
0.0000039	0.0000000	0.0000031	0.0000109	0.0000031	0.0000000	0.0000291	0.0000019
0.0000031	0.0000000	0.0000039	0.0000140	0.0000031	0.0000000	0.0000282	0.0000029
0.0000028	0.0000000	0.0000031	0.0000200	0.0000028	0.0000000	0.0000282	0.0000060
0.0000031	0.0000000	0.0000031	0.0000301	0.0000031	0.0000000	0.0000279	0.0000109
0.0000031	0.0000000	0.0000029	0.0000520	0.0000031	0.0000010	0.0000360	0.0000219
0.0000021	0.0000010	0.0000029	0.0000968	0.0000060	0.0000000	0.0000410	0.0000441
0.0000050	0.0000010	0.0000040	0.0002000	0.0000091	0.0000009	0.0000520	0.0000899
0.0001562	0.0000010	0.0000119	0.0003228	0.0001258	0.0000012	0.0000880	0.0001672
0.0000990	0.0000009	0.0000120	0.0005989	0.0001180	0.0000010	0.0000961	0.0003421
0.0001299	0.0000010	0.0000100	0.0010021	0.0001108	0.0000012	0.0001459	0.0006330
0.0002668	0.0000100	0.0000101	0.0021260	0.0001250	0.0005891	0.0002570	0.0012550
0.0001340	0.0000069	0.0000119	0.0038729	0.0001359	0.0011931	0.0004442	0.0024871
0.0001819	0.0000081	0.0000119	0.0077722	0.0001600	0.0022781	0.0006640	0.0050099
0.0002649	0.0000081	0.0000129	0.0153229	0.0002170	0.0045268	0.0011070	0.0095861
0.0004170	0.0000081	0.0000131	0.0310362	0.0003259	0.0090120	0.0021300	0.0182519
0.0007350	0.0000091	0.0000140	0.0547940	0.0005310	0.0180411	0.0039990	0.0353799
0.0013349	0.0000081	0.0000148	0.1088310	0.0009751	0.0359791	0.0076902	0.0718269
0.0026638	0.0000100	0.0000191	0.1992931	0.0011399	0.0453350	0.0141171	0.1284690

Look at the following

- cpu and gpu times for the same size problem

- cpu and gpu times as the problem size increases

We get integer overflow as the problem size increases.

The next three examples are variations on this one, for 64 bit integers and 32 and 64 bit reals.

15.9 Chapter 45 example 3: gpu and cpu computation, 64 bit integers

Here is the source code.

```
include 'integer_kind_module.f90'
include 'nvidia_precision_module.f90'
include 'timing_module.f90'
!
! The basis for the example is 2.12.2 in
! the Cuda Fortran Programming Guide
!
module initialise_array
  use integer_kind_module
  contains

  attributes (device) subroutine initialise(z)
    implicit none
    integer (i64) , dimension(:) , device :: z
    integer                                :: I
      I = (blockidx%x-1) * blockdim%x + threadidx%x
      z(i)=I
  end subroutine

end module
module calculate
  use integer_kind_module
  use initialise_array
  implicit none

  contains

    attributes (global) subroutine Kernel(x)

      implicit none
      integer (i64) , dimension(:) , device :: x

        call initialise(x)

    end subroutine

    function device_summation(x)

      implicit none
      integer (i64)             :: device_summation
```

```
      integer (i64) , &
        dimension(:) , device :: x
      integer (i64)            :: total
      integer                  :: I
        total = 0
        !$cuf kernel do <<< * , * >>>
        do I = 1 , size(x)
          total = total + x(I)
        end do

        device_summation = total

    end function

end module
program test
  use integer_kind_module
  use precision_module , wp => dp
  use timing_module
  use calculate

  use cudafor

  implicit none

  integer                                :: n
  integer (i64) , dimension(:) , &
    allocatable , device               :: x
  integer (i64) , dimension(:) , &
    allocatable                        :: y
  integer (i64)                        :: cpu_sum
  integer (i64)                        :: device_sum

  integer                                :: I
  integer                                :: allocation_status

  integer                                :: threads_per_block
  integer                                :: thread_blocks

  integer                                :: loop_count
  integer                                :: ierrSync
  integer                                :: ierrAsync
  integer                                :: istat

  real (wp) , dimension(20,8)        :: timing_figures
  real (wp)                              :: t

! the loop_count value depends on whether
```

```
! we are dealing with 32 or 64 bit data items.
! set up 20 to work with both 32 and 64 bit data
     call start_timing()

     print *,'    Thread     Threads               N
Sum           Time'
     print *,'    blocks     per block'
     allocation_status = 0
     threads_per_block = 1024
     n                    =     1 * 1024
     loop_count = 20

     do I = 1 , loop_count

       thread_blocks      =  n/threads_per_block
       cpu_sum=0

       allocate(x(n),stat=allocation_status)
       if (allocation_status > 0) then
         print *,' Device allocation failed'
         print *,' N = ',n
         print *,' Program terminates'
         stop 10
       end if

       t = time_difference()
       timing_figures(i,1) = t

       allocate(y(n),stat=allocation_status)
       if (allocation_status > 0) then
         print *,' CPU allocation failed'
         print *,' N = ',n
         print *,' Program terminates'
         stop 20
       end if

       t = time_difference()
       timing_figures(i,2) = t
       x=0

       t = time_difference()
       timing_figures(i,3) = t

       y=x
       t = time_difference()
       timing_figures(i,4) = t
```

```
       call Kernel<<< thread_blocks   , threads_per_block
>>>(x)

       ierrSync  = cudaGetLastError()
       ierrAsync = cudaDeviceSynchronize()
       if ( ierrSync /= cudaSuccess ) then
         write (* ,*) ' Sync kernel error : ' ,
cudaGetErrorString( ierrSync )
       end if
       if ( ierrAsync /= cudaSuccess ) then
         write (* ,*) ' Async kernel error : ' ,
cudaGetErrorString ( ierrAsync )
       end if
       istat = cudaDeviceSynchronize ()

       device_sum = device_summation(x)
       t = time_difference()
       timing_figures(i,7) = t

       print 20,thread_blocks,threads_per_block,n,device_sum,t
       20 format(2x,i8,2x,i6,6x,i10,2x,i20,2x,f10.7)

       y=x

       t = time_difference()
       timing_figures(i,4) = t
       cpu_sum=sum(y)
       t = time_difference()
       timing_figures(i,8) = t
       print 30,cpu_sum,t
       30 format(36x,i20,2x,f10.7)

       deallocate(x)
       t = time_difference()
       timing_figures(i,5) = t

       deallocate(y)
       t = time_difference()
       timing_figures(i,6) = t
       n = n * 2

     end do

   call end_timing()
     print *,'  Allocate                   Assign
Deallocate                 Summation'
!            1234567890123456789012345678901234567890
```

```
    print *,'          gpu             cpu            gpu
cpu            gpu             cpu            gpu            cpu'
    do I=1,20
      print 40,timing_figures(i,1:8)
      40 format(8(2x,f10.7))
    end do
end program
```

Here is the output.

```
 2022/11/30 10:14:39 442
     Thread     Threads              N                   Sum            Time
     blocks     per block
          1      1024              1024                    524800
0.0000658
                                                             524800
0.0000009
          2      1024              2048                   2098176
0.0000298
                                                            2098176
0.0000021
          4      1024              4096                   8390656
0.0000279
                                                            8390656
0.0000031
          8      1024              8192                  33558528
0.0000259
                                                           33558528
0.0000050
         16      1024             16384                 134225920
0.0000260
                                                          134225920
0.0000121
         32      1024             32768                 536887296
0.0000251
                                                          536887296
0.0000200
         64      1024             65536                2147516416
0.0000291
                                                         2147516416
0.0000420
        128      1024            131072                8590000128    0.0000430
                                                         8590000128
0.0000861
        256      1024            262144               34359869440    0.0000548
                                                        34359869440
0.0001611
        512      1024            524288              137439215616    0.0000598
                                                       137439215616
0.0003290
       1024      1024           1048576              549756338176    0.0000879
                                                       549756338176
0.0006010
       2048      1024           2097152             2199024304128    0.0001359
                                                      2199024304128
0.0012053
       4096      1024           4194304             8796095119360    0.0002279
```

```
                                                  8796095119360
0.0023810
      8192     1024          8388608         35184376283136    0.0004132
                                                 35184376283136
0.0047920
     16384     1024         16777216        140737496743936    0.0007450
                                                140737496743936
0.0094822
     32768     1024         33554432        562949970198528    0.0014982
                                                562949970198528
0.0188470
     65536     1024         67108864       2251799847239680    0.0027871
                                               2251799847239680
0.0365910
    131072     1024        134217728       9007199321849856    0.0056432
                                               9007199321849856
0.0708540
    262144     1024        268435456      36028797153181696    0.0112171
                                              36028797153181696
0.1284890
    524288     1024        536870912     144115188344291328    0.0223523
                                             144115188344291328
0.2568512
 2022/11/30 10:14:45 628
 Total time =                          6.186443
   Allocate                      Assign                        Deallocate
Summation
            gpu            cpu            gpu            cpu            gpu
cpu            gpu            cpu
   0.2291901    0.0000128    0.0000441    0.0000482    0.0000070
0.0000071    0.0000658    0.0000009
   0.0000038    0.0000012    0.0000038    0.0000101    0.0000041
0.0000000    0.0000298    0.0000021
   0.0000038    0.0000000    0.0000040    0.0000132    0.0000038
0.0000000    0.0000279    0.0000031
   0.0000031    0.0000009    0.0000031    0.0000191    0.0000029
0.0000000    0.0000259    0.0000050
   0.0000031    0.0000000    0.0000028    0.0000360    0.0000029
0.0000010    0.0000260    0.0000121
   0.0000021    0.0000010    0.0000031    0.0000510    0.0000029
0.0000000    0.0000251    0.0000200
   0.0000031    0.0000000    0.0000040    0.0000948    0.0000050
0.0000010    0.0000291    0.0000420
   0.0000040    0.0000000    0.0000050    0.0002009    0.0000081
0.0000010    0.0000430    0.0000861
   0.0001518    0.0000012    0.0000110    0.0003300    0.0001159
0.0000010    0.0000548    0.0001611
   0.0000951    0.0000009    0.0000110    0.0005601    0.0001161
0.0000010    0.0000598    0.0003290
   0.0000961    0.0000009    0.0000100    0.0009881    0.0001071
0.0000009    0.0000879    0.0006010
   0.0001500    0.0000119    0.0000100    0.0020020    0.0001218
0.0005832    0.0001359    0.0012053
   0.0001308    0.0000060    0.0000110    0.0038441    0.0001271
0.0011470    0.0002279    0.0023810
   0.0001760    0.0000069    0.0000119    0.0076429    0.0001528
0.0022800    0.0004132    0.0047920
```

```
    0.0002592    0.0000079    0.0000121    0.0149989    0.0002060
0.0045090    0.0007450    0.0094822
    0.0004170    0.0000081    0.0000128    0.0311039    0.0003199
0.0090182    0.0014982    0.0188470
    0.0007420    0.0000100    0.0000150    0.0582519    0.0005332
0.0183649    0.0027871    0.0365910
    0.0013580    0.0000091    0.0000138    0.1182460    0.0009689
0.0361149    0.0056432    0.0708540
    0.0026291    0.0000090    0.0000160    0.1877139    0.0011620
0.0464781    0.0112171    0.1284890
    0.0033741    0.0000069    0.0000159    0.3794767    0.0022259
0.0906270    0.0223523    0.2568512
```

Look at the following

- cpu and gpu times for the same size problem
- cpu and gpu times as the problem size increases

There is no integer overflow in this case.

15.10 Chapter 45 example 4: gpu and cpu computation, 32 bit reals

Here is the source code

```
include 'precision_module.f90'
include 'timing_module.f90'
!
! The basis for the example is 2.12.2 in
! the Cuda Fortran Programming Guide
!
module initialise_array
  use precision_module , wp => sp
  contains

  attributes (device) subroutine initialise(z)
    implicit none
    real (wp) , dimension(:) , device :: z
    integer                           :: I
      I = (blockidx%x-1) * blockdim%x + threadidx%x
      z(i)=I
  end subroutine

end module
module calculate
  use precision_module , wp => sp
  use initialise_array
  implicit none

  contains

    attributes (global) subroutine Kernel(x)

      implicit none
      real (wp) , dimension(:) , device :: x
```

```
        call initialise(x)

    end subroutine

    function device_summation(x)

      implicit none
      real (wp)                :: device_summation
      real (wp) , &
        dimension(:) , device :: x
      real (wp)                :: total
      integer                  :: I
        total = 0
        !$cuf kernel do <<< * , * >>>
        do I = 1 , size(x)
          total = total + x(I)
        end do

        device_summation = total

    end function

end module
program test
  use precision_module , wp => sp
  use timing_module
  use calculate

  use cudafor

  implicit none

  integer                             :: n
  real (wp) , dimension(:) , &
    allocatable , device            :: x
  real (wp) , dimension(:) , &
    allocatable                     :: y
  real (wp)                          :: cpu_sum
  real (wp)                          :: device_sum

  integer                             :: I
  integer                             :: allocation_status

  integer                             :: threads_per_block
  integer                             :: thread_blocks

  integer                             :: loop_count
```



```
  integer                              :: ierrSync
  integer                              :: ierrAsync
  integer                              :: istat

  real (dp) , dimension(20,8)   :: timing_figures
  real (dp)                         :: t

! the loop_count value depends on whether
! we are dealing with 32 or 64 bit data items.
! set up 20 to work with both 32 and 64 bit data
    call start_timing()

    print *,'   Thread    Threads              N
Sum           Time'
    print *,'   blocks    per block'
    allocation_status = 0
    threads_per_block = 1024
    n                    =    1 * 1024
    loop_count = 20

    do I = 1 , loop_count

      thread_blocks      =  n/threads_per_block
      cpu_sum=0

      allocate(x(n),stat=allocation_status)
      if (allocation_status > 0) then
        print *,' Device allocation failed'
        print *,' N = ',n
        print *,' Program terminates'
        stop 10
      end if

      t = time_difference()
      timing_figures(i,1) = t

      allocate(y(n),stat=allocation_status)
      if (allocation_status > 0) then
        print *,' CPU allocation failed'
        print *,' N = ',n
        print *,' Program terminates'
        stop 20
      end if

      t = time_difference()
      timing_figures(i,2) = t
      x=0
```

```
       t = time_difference()
       timing_figures(i,3) = t

       y=x
       t = time_difference()
       timing_figures(i,4) = t

       call Kernel<<< thread_blocks  , threads_per_block
>>>(x)

       ierrSync  = cudaGetLastError()
       ierrAsync = cudaDeviceSynchronize()
       if ( ierrSync /= cudaSuccess ) then
         write (* ,*) ' Sync kernel error : ' ,
cudaGetErrorString( ierrSync )
       end if
       if ( ierrAsync /= cudaSuccess ) then
         write (* ,*) ' Async kernel error : ' ,
cudaGetErrorString ( ierrAsync )
       end if
       istat = cudaDeviceSynchronize ()

       device_sum = device_summation(x)
       t = time_difference()
       timing_figures(i,7) = t

       print 20,thread_blocks,threads_per_block,n,device_sum,t
       20 format(2x,i8,2x,i6,6x,i10,2x,f21.0,2x,f10.7)

       y=x

       t = time_difference()
       timing_figures(i,4) = t
       cpu_sum=sum(y)
       t = time_difference()
       timing_figures(i,8) = t
       print 30,cpu_sum,t
       30 format(36x,f21.0,2x,f10.7)

       deallocate(x)
       t = time_difference()
       timing_figures(i,5) = t

       deallocate(y)
       t = time_difference()
       timing_figures(i,6) = t
       n = n * 2
```



```
     end do

    call end_timing()
      print *,'  Allocate                    Assign
Deallocate                 Summation'
!            1234567890123456789012345678901234567890
    print *,'         gpu           cpu           gpu
cpu           gpu           cpu           gpu           cpu'
    do I=1,20
      print 40,timing_figures(i,1:8)
      40 format(8(2x,f10.7))
    end do
end program
```

Here is the output.

```
 2022/11/30 10:14:45 696
     Thread     Threads              N                   Sum           Time
     blocks     per block
          1     1024              1024                    524800.
0.0000681
                                                            524800.
0.0000000
          2     1024              2048                   2098176.
0.0000310
                                                           2098176.
0.0000010
          4     1024              4096                   8390656.
0.0000351
                                                           8390656.
0.0000019
          8     1024              8192                  33558528.
0.0000251
                                                          33558528.
0.0000028
         16     1024             16384                 134225920.
0.0000251
                                                         134225920.
0.0000050
         32     1024             32768                 536887296.
0.0000250
                                                         536887296.
0.0000101
         64     1024             65536                2147506688.
0.0000250
                                                        2147516416.
0.0000201
        128     1024            131072                8589958144.
0.0000301
                                                        8590000128.
0.0000438
        256     1024            262144               34359762944.
0.0000461
                                                       34359828480.
0.0000851
```

```
        512     1024           524288           137438986240.
0.0000570
                                                   137439051776.
0.0001629
       1024     1024          1048576          549769707520.
0.0000720
                                                   549755944960.
0.0003360
       2048     1024          2097152         2199037149184.    0.0001090
                                                 2199023255552.
0.0006110
       4096     1024          4194304         8796105605120.    0.0001630
                                                 8796093022208.
0.0012469
       8192     1024          8388608        35184384671744.    0.0002880
                                                35184883793920.
0.0024350
      16384     1024         16777216       140738612428800.    0.0005150
                                               140738192998400.
0.0048981
      32768     1024         33554432       562950691618816.    0.0008500
                                               563017867591680.
0.0093050
      65536     1024         67108864      2251800082120704.    0.0014942
                                              2252438153199616.
0.0176950
     131072     1024        134217728      9007334546210816.    0.0029719
                                              9020689747017728.
0.0343949
     262144     1024        268435456     36028801313931264.    0.0060379
                                             36042283216273408.
0.0693681
     524288     1024        536870912    144115188075855872.    0.0119860
                                            150128966923976704.
0.1356740
 2022/11/30 10:14:49 362
 Total time =                          3.666342
   Allocate                      Assign                      Deallocate
Summation
           gpu           cpu           gpu           cpu           gpu
cpu           gpu           cpu
    0.2630569    0.0000081    0.0000330    0.0000298    0.0000081
0.0000031    0.0000681    0.0000000
    0.0000039    0.0000000    0.0000040    0.0000090    0.0000040
0.0000000    0.0000310    0.0000010
    0.0000039    0.0000000    0.0000031    0.0000100    0.0000031
0.0000000    0.0000351    0.0000019
    0.0000028    0.0000000    0.0000041    0.0000131    0.0000031
0.0000000    0.0000251    0.0000028
    0.0000029    0.0000000    0.0000031    0.0000200    0.0000031
0.0000010    0.0000251    0.0000050
    0.0000019    0.0000012    0.0000028    0.0000298    0.0000031
0.0000000    0.0000250    0.0000101
    0.0000031    0.0000009    0.0000029    0.0000541    0.0000038
0.0000000    0.0000250    0.0000201
    0.0000031    0.0000000    0.0000040    0.0001030    0.0000050
0.0000012    0.0000301    0.0000438
```



```
   0.0000038    0.0000012    0.0000050    0.0001948    0.0000078
0.0000012    0.0000461    0.0000851
   0.0001579    0.0000012    0.0000119    0.0003371    0.0001211
0.0000009    0.0000570    0.0001629
   0.0000949    0.0000022    0.0000100    0.0005781    0.0001120
0.0000010    0.0000720    0.0003360
   0.0000961    0.0000009    0.0000110    0.0010631    0.0001090
0.0000009    0.0001090    0.0006110
   0.0001491    0.0000119    0.0000100    0.0021210    0.0001231
0.0005810    0.0001630    0.0012469
   0.0001318    0.0000072    0.0000110    0.0038271    0.0001359
0.0011601    0.0002880    0.0024350
   0.0001970    0.0000078    0.0000122    0.0078680    0.0001860
0.0022750    0.0005150    0.0048981
   0.0002689    0.0000091    0.0000119    0.0151670    0.0002160
0.0045221    0.0008500    0.0093050
   0.0004249    0.0000090    0.0000131    0.0306599    0.0003140
0.0090329    0.0014942    0.0176950
   0.0007351    0.0000090    0.0000132    0.0540631    0.0005331
0.0180490    0.0029719    0.0343949
   0.0013519    0.0000090    0.0000151    0.1072519    0.0009461
0.0361290    0.0060379    0.0693681
   0.0026328    0.0000091    0.0000159    0.2119811    0.0013599
0.0468009    0.0119860    0.1356740
```

Look at the following

- cpu and gpu times for the same size problem;
- cpu and gpu times as the problem size increases;
- summation values for the gpu and cpu for the same sized problems - we get different computational results after 8 iterations;

15.11 Chapter 45 example 5: gpu and cpu computation, 64 bit reals

Here is the source code

```
include 'precision_module.f90'
include 'timing_module.f90'
!
! The basis for the example is 2.12.2 in
! the Cuda Fortran Programming Guide
!
module initialise_array
  use precision_module , wp => dp
  contains

  attributes (device) subroutine initialise(z)
    implicit none
    real (wp) , dimension(:) , device :: z
    integer                              :: I
      I = (blockidx%x-1) * blockdim%x + threadidx%x
      z(i)=I
  end subroutine
```

```
end module
module calculate
  use precision_module , wp => dp
  use initialise_array
  implicit none

  contains

    attributes (global) subroutine Kernel(x)

      implicit none
      real (wp) , dimension(:) , device :: x

        call initialise(x)

    end subroutine

    function device_summation(x)

      implicit none
      real (wp)                  :: device_summation
      real (wp) , &
        dimension(:) , device :: x
      real (wp)                  :: total
      integer                    :: I
        total = 0
        !$cuf kernel do <<< * , * >>>
        do I = 1 , size(x)
          total = total + x(I)
        end do

        device_summation = total

    end function

end module
program test
  use precision_module , wp => dp
  use timing_module
  use calculate

  use cudafor

  implicit none

  integer                          :: n
  real (wp) , dimension(:) , &
    allocatable , device         :: x
```

```
  real (wp) , dimension(:) , &
    allocatable                   :: y
  real (wp)                        :: cpu_sum
  real (wp)                        :: device_sum

  integer                          :: I
  integer                          :: allocation_status

  integer                          :: threads_per_block
  integer                          :: thread_blocks

  integer                          :: loop_count
  integer                          :: ierrSync
  integer                          :: ierrAsync
  integer                          :: istat

  real (dp) , dimension(20,8)   :: timing_figures
  real (dp)                        :: t

! the loop_count value depends on whether
! we are dealing with 32 or 64 bit data items.
! set up 20 to work with both 32 and 64 bit data
    call start_timing()

    print *,'   Thread     Threads              N
Sum          Time'
    print *,'   blocks     per block'
    allocation_status = 0
    threads_per_block = 1024
    n                   =     1 * 1024
    loop_count = 20

    do I = 1 , loop_count

      thread_blocks      =  n/threads_per_block
      cpu_sum=0

      allocate(x(n),stat=allocation_status)
      if (allocation_status > 0) then
        print *,' Device allocation failed'
        print *,' N = ',n
        print *,' Program terminates'
        stop 10
      end if

      t = time_difference()
      timing_figures(i,1) = t
```

```
        allocate(y(n),stat=allocation_status)
        if (allocation_status > 0) then
          print *,' CPU allocation failed'
          print *,' N = ',n
          print *,' Program terminates'
          stop 20
        end if

        t = time_difference()
        timing_figures(i,2) = t
        x=0

        t = time_difference()
        timing_figures(i,3) = t

        y=x
        t = time_difference()
        timing_figures(i,4) = t

        call Kernel<<< thread_blocks  , threads_per_block
>>>(x)

        ierrSync  = cudaGetLastError()
        ierrAsync = cudaDeviceSynchronize()
        if ( ierrSync /= cudaSuccess ) then
          write (* ,*) ' Sync kernel error : ' ,
cudaGetErrorString( ierrSync )
        end if
        if ( ierrAsync /= cudaSuccess ) then
          write (* ,*) ' Async kernel error : ' ,
cudaGetErrorString ( ierrAsync )
        end if
        istat = cudaDeviceSynchronize ()

        device_sum = device_summation(x)
        t = time_difference()
        timing_figures(i,7) = t

        print 20,thread_blocks,threads_per_block,n,device_sum,t
        20 format(2x,i8,2x,i6,6x,i10,2x,f21.0,2x,f10.7)

        y=x

        t = time_difference()
        timing_figures(i,4) = t
        cpu_sum=sum(y)
        t = time_difference()
        timing_figures(i,8) = t
```

```
        print 30,cpu_sum,t
        30 format(36x,f21.0,2x,f10.7)

        deallocate(x)
        t = time_difference()
        timing_figures(i,5) = t

        deallocate(y)
        t = time_difference()
        timing_figures(i,6) = t
        n = n * 2

      end do

    call end_timing()
      print *,'  Allocate                       Assign
Deallocate                  Summation'
!             1234567890123456789012345678901234567890
    print *,'          gpu            cpu           gpu
cpu           gpu            cpu           gpu            cpu'
    do I=1,20
      print 40,timing_figures(i,1:8)
      40 format(8(2x,f10.7))
    end do
end program
```

Here is the output

```
 2022/11/30 10:14:49 426
    Thread     Threads                N                   Sum           Time
    blocks     per block
         1     1024              1024                      524800.
0.0000508
                                                              524800.
0.0000012
         2     1024              2048                     2098176.
0.0000320
                                                             2098176.
0.0000019
         4     1024              4096                     8390656.
0.0000231
                                                             8390656.
0.0000028
         8     1024              8192                    33558528.
0.0000210
                                                            33558528.
0.0000050
        16     1024             16384                   134225920.
0.0000210
                                                           134225920.
0.0000091
        32     1024             32768                   536887296.
0.0000301
```

```
                                                            536887296.
0.0000188
        64     1024            65536               2147516416.
0.0000332
                                                           2147516416.
0.0000380
       128     1024           131072               8590000128.
0.0000391
                                                           8590000128.
0.0000861
       256     1024           262144              34359869440.
0.0000591
                                                          34359869440.
0.0001579
       512     1024           524288             137439215616.
0.0000780
                                                         137439215616.
0.0002899
      1024     1024          1048576             549756338176.
0.0001240
                                                         549756338176.
0.0005748
      2048     1024          2097152            2199024304128.    0.0002029
                                                        2199024304128.
0.0010359
      4096     1024          4194304            8796095119360.    0.0003779
                                                        8796095119360.
0.0020199
      8192     1024          8388608           35184376283136.    0.0007641
                                                       35184376283136.
0.0040598
     16384     1024         16777216          140737496743936.    0.0010212
                                                      140737496743936.
0.0080361
     32768     1024         33554432          562949970198528.    0.0015650
                                                      562949970198528.
0.0160630
     65536     1024         67108864         2251799847239680.    0.0028451
                                                     2251799847239680.
0.0323019
    131072     1024        134217728         9007199321849856.    0.0055599
                                                     9007199321849856.
0.0645239
    262144     1024        268435456        36028797153181696.    0.0111342
                                                    36028797153181696.
0.1290501
    524288     1024        536870912       144115188344291328.    0.0225160
                                                   144115188344291328.
0.2583959
 2022/11/30 10:14:54 755
 Total time =                           5.328152
   Allocate                    Assign                     Deallocate
Summation
         gpu           cpu           gpu           cpu           gpu
cpu           gpu           cpu
   0.2520980    0.0000110    0.0000339    0.0000200    0.0000048
0.0000041    0.0000508    0.0000012
```

```
    0.0000031    0.0000009    0.0000031    0.0000090    0.0000031
0.0000000    0.0000320    0.0000019
    0.0000019    0.0000010    0.0000019    0.0000110    0.0000022
0.0000009    0.0000231    0.0000028
    0.0000019    0.0000000    0.0000031    0.0000169    0.0000021
0.0000000    0.0000210    0.0000050
    0.0000019    0.0000010    0.0000019    0.0000269    0.0000021
0.0000010    0.0000210    0.0000091
    0.0000019    0.0000009    0.0000022    0.0000460    0.0000022
0.0000009    0.0000301    0.0000188
    0.0000019    0.0000000    0.0000031    0.0000879    0.0000040
0.0000010    0.0000332    0.0000380
    0.0000031    0.0000009    0.0000041    0.0001788    0.0000081
0.0000000    0.0000391    0.0000861
    0.0001390    0.0000010    0.0000090    0.0002971    0.0001001
0.0000009    0.0000591    0.0001579
    0.0000770    0.0000000    0.0000101    0.0004751    0.0000911
0.0000010    0.0000780    0.0002899
    0.0000770    0.0000009    0.0000081    0.0008950    0.0000921
0.0000009    0.0001240    0.0005748
    0.0001512    0.0000069    0.0000090    0.0017102    0.0000951
0.0004380    0.0002029    0.0010359
    0.0001049    0.0000050    0.0000091    0.0030072    0.0001020
0.0008290    0.0003779    0.0020199
    0.0001230    0.0000060    0.0000091    0.0059221    0.0001230
0.0016141    0.0007641    0.0040598
    0.0001879    0.0000062    0.0000100    0.0119059    0.0001619
0.0031340    0.0010212    0.0080361
    0.0003059    0.0000062    0.0000100    0.0244870    0.0002501
0.0059600    0.0015650    0.0160630
    0.0004771    0.0000069    0.0000090    0.0463850    0.0004111
0.0120411    0.0028451    0.0323019
    0.0008919    0.0000060    0.0000121    0.0940702    0.0007401
0.0234630    0.0055599    0.0645239
    0.0016458    0.0000062    0.0000129    0.1882619    0.0013229
0.0454951    0.0111342    0.1290501
    0.0032279    0.0000069    0.0000151    0.3912561    0.0025039
0.0913012    0.0225160    0.2583959
```

Look at the following

- cpu and gpu times for the same size problem;
- cpu and gpu times as the problem size increases;

Summation values for the gpu and cpu are now the same.

15.12 Chapter 45 example 6: calculating pi

In this section we look at a Cuda Fortran program to calulate pi using the same methods as in the chapters on parallel programming with MPI, Openmp and coarray fortran. We also look at comparing the timing with these other 3 methods and with other compilers.

Here is the source code.

```
include 'precision_module.f90'
include 'integer_kind_module.f90'
include 'timing_module.f90'
module fill
  use integer_kind_module
```

```
  use precision_module

  implicit none
  contains
    attributes(global) subroutine fill_pi_array(y, n)
      implicit none

      real (dp) , device :: y(:)
      integer, value      :: n
      integer              :: I
      real                  :: x
      real (dp)            :: width
        width = 1.0_dp/n
        I = (blockidx%x-1)*blockdim%x + threadidx%x
        x = width*(real(i,dp)-0.5_dp)
        if (I <= n) then
          y(I) = 4.0_dp/(1.0_dp+x*x)
        end if
      return
    end subroutine
end module
program parallel_pi
  use cublas
  use fill

  use integer_kind_module
  use precision_module
  use timing_module

  implicit none
  integer                                  :: n
  real (dp) , allocatable , device :: x(:)
  real (dp)                                :: calculated_pi
  real (dp)                                :: intrinsic_pi =
4.0_dp*atan(1.0_dp)
  real (dp)                                :: pi_difference
  integer                                  :: threads_per_block
= 1000
  integer                                  :: thread_blocks
  integer                                  :: I
  character (20)                          :: heading

    call start_timing()

    n=1000000

    do I=1,3
```

```
      print 10,n
      10 format(' N =     ',i12)

      thread_blocks = n/threads_per_block
      allocate(x(n))

      heading = ' Allocation'
      print 100,heading,time_difference()
      100 format(a20,f18.6)
     call fill_pi_array<<<thread_blocks,threads_per_block>>>(x,n)

      heading = ' Fill array'
      print 100,heading,time_difference()
      calculated_pi = dasum(n,x,1)/n

      heading = ' dasum call'
      print 100,heading,time_difference()
      print 20,calculated_pi
      20 format(' Calculated ',f18.15)
      print 30,intrinsic_pi
      30 format(' Intrinsic  ',f18.15)
      pi_difference=abs(calculated_pi-intrinsic_pi)
      print 40,pi_difference
      40 format(' Difference ',f18.15)
      deallocate(x)
      heading = ' Deallocation'
      print 100,heading,time_difference()
      n=n*10
      print *,' '

    end do
  call end_timing()

end program
```

Here is the output.

```
 2022/11/30 10:14:55 326
 N =           1000000
 Allocation                     0.206592
 Fill array                     0.000338
 dasum call                     7.525906
 Calculated  3.141592653590480
 Intrinsic   3.141592653589793
 Difference  0.000000000000687
 Deallocation                   0.000319

 N =          10000000
 Allocation                     0.000492
 Fill array                     0.003186
 dasum call                     0.000415
 Calculated  3.141592653585755
```

```
Intrinsic   3.141592653589793
Difference  0.000000000004038
Deallocation                    0.000344

N =       100000000
Allocation                      0.002609
Fill array                      0.030919
dasum call                      0.003685
Calculated  3.141592653583027
Intrinsic   3.141592653589793
Difference  0.000000000006766
Deallocation                    0.001230

2022/11/30 10:15: 3 102
Total time =                               7.776064
```

Here are some other timing figures.

15.12.1 Timing figures, example ch3204, MPI, Intel Fortran

Here is the output.

```
2022/12/ 1  9:35:20 282
 fortran_internal_pi =     3.1415926535897931
N intervals =        100000 time =    0.000655
pi =    3.1415926535981265
difference =    0.0000000000083333
N intervals =       1000000 time =    0.000236
pi =    3.1415926535898753
difference =    0.0000000000000822
N intervals =      10000000 time =    0.002073
pi =    3.1415926535897842
difference =    0.0000000000000089
N intervals =     100000000 time =    0.020611
pi =    3.1415926535897980
difference =    0.0000000000000049
N intervals =    1000000000 time =    0.206553
pi =    3.1415926535898402
difference =    0.0000000000000471
     0.013
2022/12/ 1  9:35:20 526
Total time =                               0.243566
```

15.12.2 Timing figures, example ch3304, openmp, Intel Fortran

Here is the output.

```
2022/11/30 16:47: 3 953
 Maximum number of threads is            36
 Number of threads =            36
N intervals =        100000 time =                  0.007952
difference =           0.0000000000083324
N intervals =       1000000 time =                  0.000305
difference =           0.0000000000000835
N intervals =      10000000 time =                  0.001730
```

```
 difference =          0.0000000000000004
 N intervals =    100000000 time =             0.015941
 difference =          0.0000000000000098
 N intervals =   1000000000 time =             0.060744
 difference =          0.0000000000000098
 2022/11/30 16:47: 4  40
 Total time =                         0.086695
```

15.12.3 Timing figures, example ch3304, openmp, nvidia Fortran

Here is the output.

```
 2022/11/30 16:45:25  40
  Maximum number of threads is              36
  Number of threads =            36
 N intervals =       100000 time =             0.001457
 difference =          0.0000000000083684
 N intervals =      1000000 time =             0.011744
 difference =          0.0000000000000289
 N intervals =     10000000 time =             0.087735
 difference =          0.0000000000000622
 N intervals =    100000000 time =             0.479924
 difference =          0.0000000000006333
 N intervals =   1000000000 time =             3.842343
 difference =          0.0000000000001776
 2022/11/30 16:45:29 463
 Total time =                         4.423211
```

15.12.4 Timing figures, example ch3304, openmp, Nag Fortran

Here is the output.

```
 2022/11/30 16:42:42 351
  Maximum number of threads is  36
  Number of threads =  36
 N intervals =       100000 time =             0.003242
 difference =          0.0000000000083329
 N intervals =      1000000 time =             0.003348
 difference =          0.0000000000000964
 N intervals =     10000000 time =             0.005580
 difference =          0.0000000000000102
 N intervals =    100000000 time =             0.021254
 difference =          0.0000000000000027
 N intervals =   1000000000 time =             0.105482
 difference =          0.0000000000000453
 2022/11/30 16:42:42 490
 Total time =                         0.138916
```

15.12.5 Timing figures, example ch3304, openmp, gfortran

Here is the output.

```
 2022/11/30 16:42:54 511
  Maximum number of threads is             36
```

```
  Number of threads =            36
 N intervals =       100000 time =             0.001403
 difference =         0.0000000000083338
 N intervals =      1000000 time =             0.000333
 difference =         0.0000000000000839
 N intervals =     10000000 time =             0.002151
 difference =         0.0000000000000093
 N intervals =    100000000 time =             0.028183
 difference =         0.0000000000000280
 N intervals =   1000000000 time =             0.101257
 difference =         0.0000000000000284
 2022/11/30 16:42:54 644
 Total time =                            0.133334
```

15.12.6 Timing figures, example ch3403, coarray Fortran, Intel Fortran

Here is the output.

```
  Number of images =            36
 2022/11/29 17:32: 5 617
 n intervals =       100000 time =          0.000739
 pi =    3.1415926535981269
 difference =    0.0000000000083338
 n intervals =      1000000 time =          0.000642
 pi =    3.1415926535898748
 difference =    0.0000000000000817
 n intervals =     10000000 time =          0.001582
 pi =    3.1415926535897971
 difference =    0.0000000000000040
 n intervals =    100000000 time =          0.011347
 pi =    3.1415926535898029
 difference =    0.0000000000000098
 n intervals =   1000000000 time =          0.105456
 pi =    3.1415926535898109
 difference =    0.0000000000000178
 2022/11/29 17:32: 5 737
 Total time =                            0.120299
```

15.12.7 Timing figures, example ch3403, coarray Fortran, Nag Fortran

Here is the output.

```
  Number of images =  36
 2022/11/30 16:30:38 331
 n intervals =       100000 time =          0.001245
 pi =    3.1415926535981251
 difference =    0.0000000000083320
 n intervals =      1000000 time =          0.000282
 pi =    3.1415926535898757
 difference =    0.0000000000000826
 n intervals =     10000000 time =          0.002181
 pi =    3.1415926535897838
```

```
difference =    0.0000000000000093
n intervals =     100000000 time =          0.028454
pi =    3.1415926535897989
difference =    0.0000000000000058
n intervals =    1000000000 time =          0.099151
pi =    3.1415926535898406
difference =    0.0000000000000475
2022/11/30 16:30:38 462
Total time =                         0.131326
```

15.12.8 Timing figures summary

In this section we compare some of the results. Here is a table with timing figures.

Example	Method	Compiler	Problem	Calculation		Time	Notes
number			size	time			
ch4506	gpu	nvidia	100,000,000	fill array	dasum call		1
				0.030919	0.003685	0.035	
ch3204	mpi	Intel	100,000,000	0.020611		0.021	
ch3304	openmp	Intel	100,000,000	0.015941		0.016	
ch3304	openmp	Nag	100,000,000	0.021254		0.021	
ch3304	openmp	nvidia	100,000,000	0.479924		0.480	
ch3304	openmp	gfortran	100,000,000	0.281183		0.281	
ch3403	coarray	Intel	100,000,000	0.011347		0.011	
ch3403	coarray	Nag	100,000,000	0.028454		0.028	
ch4506	gpu	nvidia	1,000,000,000	fill array	dasum call		1
				NA	NA		2
ch3204	mpi	Intel	1,000,000,000	0.2065553		0.207	

Example	Method	Compiler	Problem	Calculation		Time	Notes
number			size	time			
ch3304	openmp	Intel	1,000,000,000	0.060744		0.061	
ch3304	openmp	Nag	1,000,000,000	0.105482		0.105	
ch3304	openmp	nvidia	1,000,000,000	3.842343		3.842	
ch3304	openmp	gfortran	1,000,000,000	0.101257		0.101	
ch3403	coarray	Intel	1,000,000,000	0.105456		0.105	
ch3403	coarray	Nag	1,000,000,000	0.099151		0.099	

15.12.8.1 Notes

1 Nvidia timing has 2 components

2 Timing not available. Size too large for gpu allocation

15.13 Nvidia Cuda

We have got this to work on Windows at this time using Microsoft VS 2022. This provides C++ based parallel programming.

16 Intel oneapi toolkits

As was stated in the previous chapter both Intel and Nvidia toolkits offer the possibility of developing code that can run on both CPUs and GPUs, i.e. with a system with a cpu and graphics card it is possible to do processing on both the CPU and GPU. In this chapter we look at Intel's offerings.

16.1 Intel toolkit overview

Intel make their compilers available via a variety of toolkits: Here is the Intel link.

```
https://www.intel.com/content/www/us/en/devel-
oper/tools/oneapi/toolkits.html#gs.597yak
```

They make the following toolkits available:

- Intel® oneAPI Base Toolkit
- Intel oneAPI HPC Toolkit
- Intel® AI Analytics Toolkit
- Intel® Distribution of OpenVINO toolkit (Powered by oneAPI)
- Intel® oneAPI Rendering Toolkit
- Intel oneAPI IoT Toolkit
- Intel® System Bring-up Toolkit

The two Intel toolkits we have looked at are:

- Intel Base toolkit
- Intel HPC toolkit

More detailed coverage is given below.

16.2 Intel base toolkit

Intel recommend installing this toolkit first. You can take the default install (which is large) or choose a subset. We normally omit the Python component. We have used this toolkit on Windows, Linux (various distributions) and the Mac. Here are the components as of July 2022.

- Intel® oneAPI Collective Communications Library
- Intel® oneAPI Data Analytics Library
- Intel® oneAPI Deep Neural Networks Library
- Intel® oneAPI DPC++©++ Compiler
- Intel® oneAPI DPC++ Library
- Intel® oneAPI Math Kernel Library
- Intel® oneAPI Threading Building Blocks
- Intel® oneAPI Video Processing Library
- Intel® Advisor
- Intel® Distribution for GDB*
- Intel® Distribution for Python*
- Intel® DPC++ Compatibility Tool

- Intel® FPGA Add-on for oneAPI Base Toolkit
- Intel® Integrated Performance Primitives
- Intel® VTune™ Profile

This is about 40 GB.

16.3 Intel HPC toolkit

We recommend installing all of this toolkit. We have used this toolkit on Windows, Linux (various distributions) and the Mac. It has the following components as of July 2022.

- Intel oneAPI DPC++©++ Compiler
- Intel® C++ Compiler Classic
- Intel® Cluster Checker
- Intel® Fortran Compiler
- Intel® Fortran Compiler Classic
- Intel® Inspector
- Intel® MPI Library
- Intel® Trace Analyzer and Collector

This is about 17 GB.

16.4 Native Intel gpu examples

Currently we do not have access to an Intel gpu and cannot provide any examples.

16.5 Intel support for Nvidia gpus - under development

Here is a link to the Intel oneAPI toolkit November 2023 announcements.

`https://www.intel.com/content/www/us/en/developer/tools/oneapi/toolkits.html`

and here is a link to their Nvidia developments.

`https://developer.codeplay.com/products/oneapi/nvidia/2024.0.0/guides/get-started-guide-nvidia#install-oneapi-for-nvidia-gpus`

Here is an extract from the Intel site.

- Supported Platforms
- This release has been tested on the following platforms:
- GPU Hardware
    - NVIDIA A100-PCIE-40GB
- Architecture
    - Ampere - sm_80
- Operating System
    - Ubuntu 22.04.2 LTS
- CUDA
    - 12.2

- GPU Driver
    - 535.54.03
- This release should work across a wide array of NVIDIA GPUs and CUDA versions, but Codeplay cannot guarantee correct operation on untested platforms.
- The package has been tested on Ubuntu 22.04 only, but can be installed on any Linux systems
- System setup and installation
    - You will need the following C++ development tools installed in order to build and run oneAPI applications:

      cmake

      gcc,

      g++,

      make and

      pkg-config.
    - The following console commands will install the above tools on the most popular Linux distributions:
    - Ubuntu

      sudo apt update

      sudo apt -y install cmake pkg-config build-essential
    - Verify that the tools are installed by running:

      which cmake pkg-config make gcc g++
    - You should see output similar to:

      /usr/bin/cmake

      /usr/bin/pkg-config

      /usr/bin/make

      /usr/bin/gcc

      /usr/bin/g++

As we have installed the Nvidia toolkit on a native Ubuntu system we will concentrate on the Ubuntu version in what follows. Here is a link to some on line information.

`https://developer.codeplay.com/products/oneapi/nvidia/2024.0.0/guides/`

Here is an extract from that site.

- oneAPI for NVIDIA GPUs 2024.0.0
    - oneAPI for NVIDIA GPUs is a plugin for Intel® oneAPI Toolkits that enables developers to build oneAPI applications with DPC++ / SYCL™ and run them on NVIDIA GPUs.
    - The plugin adds a CUDA® backend to DPC++ and you will see the terms oneAPI for NVIDIA GPUs and DPC++

CUDA plugin used interchangeably throughout this documentation.

Details of working with Redhat and SuSe are given at the end of this chapter.

16.6 Documentation

Intel make available a range of documentation. Here are some of their guides and documentation.

- Intel® oneAPI DPC++/C++ Compiler Developer Guide and Reference: 823 pages
  - dpcpp-cpp-compiler_developer-guide-reference_2024.0-767253-792222.pdf
- Intel® oneAPI Programming Guide: 114 pages
  - oneapi_programming-guide_2024.0-771723-785315.pdf
- Intel® oneAPI DPC++ Library Developer Guide and Reference: 60 pages
  - onedpl_developer-guide_2022.3-768913-792229.pdf
- Get Started with the Intel oneAPI DPC++ Library: 6 pages
  - onedpl_get-started-guide_2022.3-768911-792228.pdf

Intel recommend the following free book.

- Data parallel C++ Mastering DPC++ for Programming of Heterogeneous Systems using C++ and SYCL

Here is a link

`https://link.springer.com/book/10.1007/978-1-4842-5574-2`

16.7 Installing the Intel Nvidia toolkit on other Linux operating systems

16.7.1 Red Hat and Fedora

- sudo yum update
- sudo yum -y install cmake pkgconfig
- sudo yum groupinstall "Development Tools"

16.7.2 SUSE

sudo zypper update

- sudo zypper --non-interactive install cmake pkg-config
- sudo zypper --non-interactive install pattern devel_C_C++

Verify that the tools are installed by running:

- which cmake pkg-config make gcc g++

You should see output similar to:

- /usr/bin/cmake
- /usr/bin/pkg-config
- /usr/bin/make
- /usr/bin/gcc

- /usr/bin/g++

17 Templates and generic programming in the next standard

There are currently two proposals before the standards committee concerning templates and generic programming in the next standard - F202Y

One was a simple proposal from Japan, and the second is the J3 subgroup on generics and template programming.

Here are links to some of the latest documents.

`https://j3-fortran.org/doc/year/23/23-104.txt`

- Formal specs for TEMPLATE

`https://j3-fortran.org/doc/year/23/23-148.txt`

- Thoughts on additional generics features

`https://j3-fortran.org/doc/year/23/23-155r2.txt`

- Formal syntax for generics

`https://j3-fortran.org/doc/year/23/23-159.txt`

- simple templates

`https://j3-fortran.org/doc/year/23/23-166r1.txt`

- Pushing the usability of templates

`https://j3-fortran.org/doc/year/23/23-187.txt`

- Shorthands for Simple Templates

`https://j3-fortran.org/doc/year/23/23-188.txt`

- Possible Solutions to Long Templates
- https://j3-fortran.org/doc/year/23/23-202.txt

Packaging long argument lists of templates

`https://j3-fortran.org/doc/year/23/23-204.txt`

- Templates Tutorials

In this chapter we provide two examples, one based on the Japanese syntax and the second based on the J3 syntax.

Both of these examples are drafts and are subject to change. They hopefully highlight some examples of what can be achieved.

17.1 Background information

The fourth edition has a number of examples of generic programming:

- chapter 25: example 1 - generic sorting module;
- chapter 25: example 2 - generic statistics module;
- chapter 38: example 1 - generic sorting example with timing module;
- chapter 38: example 6 - generic sorting module calling the C++ STL parallel sorting routines;

In all of the examples we, the programmer, have to provide subroutines that implement the sorting and statistics calculations ourselves, for each data type we are interested in. We use the interface syntax mechanism of Fortran 90 to do this. So we have:

ch2501

```
interface sort_data
  module procedure sort_real_sp
  module procedure sort_real_dp
  module procedure sort_real_qp
  module procedure sort_integer_8
  module procedure sort_integer_16
  module procedure sort_integer_32
  module procedure sort_integer_64
end interface
```

ch2502

```
interface calculate_statistics
  module procedure calculate_sp
  module procedure calculate_dp
  module procedure calculate_qp
end interface
```

ch3801

```
interface sort_data
  module procedure sort_real_sp
  module procedure sort_real_dp
  module procedure sort_real_qp
  module procedure sort_integer_8
  module procedure sort_integer_16
  module procedure sort_integer_32
  module procedure sort_integer_64
end interface
```

ch3806

```
interface sort_data
  module procedure sort_real_sp
  module procedure sort_real_dp
  module procedure sort_real_qp
  module procedure sort_integer_8
  module procedure sort_integer_16
  module procedure sort_integer_32
  module procedure sort_integer_64
end interface
```

and corresponding C++ code.

```
extern "C"
{
  void stl_sort_i32(int * x , const int nx)
  {
```

```
    vector<int> y(nx);
    int i;
    for(i=0;i<nx;i++)
      y[i]= x[i];
    sort( std::execution::par_unseq, y.begin(), y.end() );
    for(i=0;i<nx;i++)
      x[i]= y[i];
    return;
  }
}
extern "C"
{
  void stl_sort_i64(long long int * x , const int nx)
  {
    vector<long long int> y(nx);
    int i;
    for(i=0;i<nx;i++)
      y[i]= x[i];
    sort( std::execution::par_unseq, y.begin(), y.end() );
    for(i=0;i<nx;i++)
      x[i]= y[i];
    return;
  }
}
extern "C"
{
  void stl_sort_r32(float * x , const int nx)
  {
    vector<float> y(nx);
    int i;
    for(i=0;i<nx;i++)
      y[i]= x[i];
    sort( std::execution::par_unseq, y.begin(), y.end() );
    for(i=0;i<nx;i++)
      x[i]= y[i];
    return;
  }
}
extern "C"
{
  void stl_sort_r64(double * x , const int nx)
  {
    vector<double> y(nx);
    int i;
    for(i=0;i<nx;i++)
      y[i]= x[i];
    sort( std::execution::par_unseq, y.begin(), y.end() );
    for(i=0;i<nx;i++)
```



```
      x[i]= y[i];
    return;
  }
}
```

The other syntax mechanism that has been used in the sorting examples is the include option where we reduce our coding by 'including' a common algorithm, that is independent of the type of data that we are working with.

Here is an example of the quicksort include code.

```
  i = l
  j = r
  v = raw_data(int((l+r)/2))
  do
    do while (raw_data(i)<v)
      i = i + 1
    end do
    do while (v<raw_data(j))
      j = j - 1
    end do
    if (i<=j) then
      t = raw_data(i)
      raw_data(i) = raw_data(j)
      raw_data(j) = t
      i = i + 1
      j = j - 1
    end if
    if (i>j) exit
  end do
  if (l<j) then
    call quicksort(l, j)
  end if
  if (i<r) then
    call quicksort(i, r)
  end if
```

i.e. this code works with any type where the operations of comparison and assignment are defined.

Here is a complete sort subroutine for real type.

```
  subroutine sort_real_sp(raw_data, how_many)
    use precision_module
    implicit none
    integer, intent (in) :: how_many
    real (sp), intent (inout), dimension (:) :: raw_data

    call quicksort(1, how_many)

  contains
```

```
    recursive subroutine quicksort(l, r)
      implicit none
      integer, intent (in) :: l, r
      integer :: i, j
      real (sp) :: v, t

!        include 'quicksort_include_code.f90'
  i = l
  j = r
  v = raw_data(int((l+r)/2))
  do
    do while (raw_data(i)<v)
      i = i + 1
    end do
    do while (v<raw_data(j))
      j = j - 1
    end do
    if (i<=j) then
      t = raw_data(i)
      raw_data(i) = raw_data(j)
      raw_data(j) = t
      i = i + 1
      j = j - 1
    end if
    if (i>j) exit
  end do
  if (l<j) then
    call quicksort(l, j)
  end if
  if (i<r) then
    call quicksort(i, r)
  end if

    end subroutine

  end subroutine
```

We will use this subroutine as a starting point in the examples that follow.

17.2 Chapter 47 - example 1 - generic sort template, Japanese proposal

17.2.1 Template source code

Here is the template source code.

```
module sort_template_module_japan

contains

  generic subroutine sort(x, n)
```



```
    use precision_module
    use integer_kind_module

    type(i8,i16,i32,i64,sp,dp,qp) , intent(inout) :: x(:)
    integer , intent(in)     :: n

    call quicksort(1, n)

  contains

    recursive subroutine quicksort(l, r)

      implicit none
      integer, intent (in) :: l, r
      integer :: i, j
      typeof (x) :: v, t

! used to include the common sorting code
! include 'quicksort_include_code.f90'

        i = l
        j = r
        v = x(int((l+r)/2))
        do
          do while (x(i)<v)
            i = i + 1
          end do
          do while (v<x(j))
            j = j - 1
          end do
          if (i<=j) then
            t = x(i)
            x(i) = x(j)
            x(j) = t
            i = i + 1
            j = j - 1
          end if
          if (i>j) exit
        end do
        if (l<j) then
          call quicksort(l, j)
        end if
        if (i<r) then
          call quicksort(i, r)
        end if

    end subroutine
```

```
   end subroutine

end template

end module
```

The two key statements are

```
   generic subroutine sort(x, n)
```

and

```
      type(i8,i16,i32,i64,sp,dp,qp) , intent(inout) :: x(:)
```

and the last statement says that we want to be able to create or instantiate a generic sort subroutine with arrays of type

- integer - i8, i16, i32, i64, as defined in the integer_kind_module

and

- real - sp (single precision), dp (double precision), qp (quad precision), as defined in the precision_module.

The operations of comparison and assignment are defined and know by the compiler for these integer and real intrinsic kind types.

17.2.2 Complete Japanese program source code

Here is the complete source code. It is an updated version of example 1 in chapter 38.

```
include 'integer_kind_module.f90'
include 'precision_module.f90'
include 'timing_module.f90'

module sort_template_module_japan

contains

   generic subroutine sort(x, n)

      use precision_module
      use integer_kind_module

      type(i8,i16,i32,i64,sp,dp,qp) , intent(inout) :: x(:)
      integer , intent(in)     :: n

      call quicksort(1, n)

   contains

      recursive subroutine quicksort(l, r)

         implicit none
         integer, intent (in) :: l, r
         integer :: i, j
         typeof (x) :: v, t
```

```
! used to include the common sorting code
! include 'quicksort_include_code.f90'

         i = l
         j = r
         v = x(int((l+r)/2))
         do
           do while (x(i)<v)
             i = i + 1
           end do
           do while (v<x(j))
             j = j - 1
           end do
           if (i<=j) then
             t = x(i)
             x(i) = x(j)
             x(j) = t
             i = i + 1
             j = j - 1
           end if
           if (i>j) exit
         end do
         if (l<j) then
           call quicksort(l, j)
         end if
         if (i<r) then
           call quicksort(i, r)
         end if

     end subroutine

   end subroutine

end template

end module

program ch4701

   use precision_module
   use integer_kind_module
   use timing_module
   use sort_template_module_japan

   implicit none

   integer, parameter :: n  =      1000
```

```
   character *12       :: nn =    '1,000'
   character *80       :: report_file_name = 'ch3801_re-
port.txt'

   real (sp), allocatable, dimension (:) :: x_sp
   real (sp), allocatable, dimension (:) :: t_x_sp

   real (dp), allocatable, dimension (:) :: x_dp
   real (dp), allocatable, dimension (:) :: t_x_dp

   real (qp), allocatable, dimension (:) :: x_qp

   integer (i32), allocatable, dimension (:) :: y_i32
   integer (i64), allocatable, dimension (:) :: y_i64

   integer :: allocate_status = 0

   character *20, dimension (5) :: heading1 = &
   [ '   32 bit real', &
     '   32 bit int ', &
     '   64 bit real', &
     '   64 bit int ', &
     ' 128 bit real' ]

   character *20, dimension (3) :: &
   heading2 = [ '       Allocate ', &
                '       Random    ', &
                '       Sort      ' ]

   print *, 'Program starts'
   print *, 'N = ', nn
   call start_timing()

   open (unit=100, file=report_file_name)

   print *, heading1(1)

   allocate (x_sp(1:n), stat=allocate_status)
   if (allocate_status/=0) then
     print *, ' Allocate failed. Program terminates'
     stop 10
   end if

   print 100, heading2(1), time_difference()
   100 format (a20, 2x, f18.6)

   call random_number(x_sp)
   t_x_sp = x_sp
```

```
  print 100, heading2(2), time_difference()
  call sort_data(x_sp, n)
  print 100, heading2(3), time_difference()
  write (unit=100, fmt='(a)') ' First 10 32 bit reals'
  write (unit=100, fmt=110) x_sp(1:10)
110 format (5(2x,e14.6))

  print *, heading1(2)

  allocate (y_i32(1:n), stat=allocate_status)
  if (allocate_status/=0) then
    print *, 'Allocate failed. Program terminates'
    stop 30
  end if

  print 100, heading2(1), time_difference()
  y_i32 = int(t_x_sp*1000000000, i32)

  deallocate (x_sp)
  deallocate (t_x_sp)

  print 100, heading2(2), time_difference()
  call sort_data(y_i32, n)
  print 100, heading2(3), time_difference()
  write (unit=100, fmt='(a)') 'First 10 32 bit integers'
  write (unit=100, fmt=120) y_i32(1:10)
120 format (5(2x,i10))
  deallocate (y_i32)

  print *, heading1(3)

  allocate (x_dp(1:n), stat=allocate_status)
  if (allocate_status/=0) then
    print *, 'Allocate failed. Program terminates'
    stop 30
  end if

  allocate (t_x_dp(1:n), stat=allocate_status)
  if (allocate_status/=0) then
    print *, 'Allocate failed. Program terminates'
    stop 40
  end if

  print 100, heading2(1), time_difference()
  call random_number(x_dp)
  t_x_dp = x_dp
  print 100, heading2(2), time_difference()
```

```
   call sort_data(x_dp, n)
   print 100, heading2(3), time_difference()
   write (unit=100, fmt='(a)') 'First 10 64 bit reals'
   write (unit=100, fmt=110) x_dp(1:10)

   print *, heading1(4)

   allocate (y_i64(1:n), stat=allocate_status)
   if (allocate_status/=0) then
     print *, 'Allocate failed. Program terminates'
     stop 40
   end if

   print 100, heading2(1), time_difference()
   y_i64 = int(t_x_dp*1000000000000000_i64, i64)

   deallocate (x_dp)
   deallocate (t_x_dp)

   print 100, heading2(2), time_difference()
   call sort_data(y_i64, n)
   print 100, heading2(3), time_difference()
   write (unit=100, fmt='(a)') 'First 10 64 bit integers'
   write (unit=100, fmt=120) y_i64(1:10)
   deallocate (y_i64)

   print *, heading1(5)

   allocate (x_qp(1:n), stat=allocate_status)
   if (allocate_status/=0) then
     print *, 'Allocate failed. Program terminates'
     stop 50
   end if

   print 100, heading2(1), time_difference()
   call random_number(x_qp)
   print 100, heading2(2), time_difference()
   call sort_data(x_qp, n)
   print 100, heading2(3), time_difference()
   write (unit=100, fmt='(a)') 'First 10 128 bitreals'
   write (unit=100, fmt=110) x_qp(1:10)

   close (200)
   print *, 'Program terminates'
   call end_timing()

end program
```


17.3 Chapter 47 - example 2 - generic sort template - J3 proposal

17.3.1 J3 proposal template source code

Here is the sort template

```
module sort_template_module_j3

template sort_template(k)

!   use precision_module
!   use integer_kind_module

  private

  public :: sort_data

! integer , constant  :: k
  integer , parameter :: k

  interface sort_data
    module procedure sort
  end interface

contains

  subroutine sort(x, n)

    use precision_module
    use integer_kind_module

    type(k) , intent(inout) :: x(:)
    integer , intent(in)     :: n

    call quicksort(1, n)

  contains

    recursive subroutine quicksort(l, r)

      implicit none
      integer, intent (in) :: l, r
      integer :: i, j
      type (k) :: v, t

! used to include the common sorting code
! include 'quicksort_include_code.f90'

        i = l
        j = r
```

```
        v = x(int((l+r)/2))
        do
          do while (x(i)<v)
            i = i + 1
          end do
          do while (v<x(j))
            j = j - 1
          end do
          if (i<=j) then
            t = x(i)
            x(i) = x(j)
            x(j) = t
            i = i + 1
            j = j - 1
          end if
          if (i>j) exit
        end do
        if (l<j) then
          call quicksort(l, j)
        end if
        if (i<r) then
          call quicksort(i, r)
        end if

    end subroutine

  end subroutine

end template

end module
```

The key statements are

```
template sort_template(k)
```

and

```
    type(k) , intent(inout) :: x(:)
```

where the sort routine is effectively parameterised by the type of the x array.

The next set of statements of interest are in the main program.

```
  instantiate sort_template( sp)
  instantiate sort_template( dp)
  instantiate sort_template( qp)
  instantiate sort_template(i32)
  instantiate sort_template(i64)
```

where we are telling the compiler we want to create or instantiate the sort_template with arrays of integer i32 and i64 type, and real arrays of type sp, dp and qp type. Again the operations of comparison and assignment are know by the compiler for these integer and real internal kind types.

17.3.2 J3 proposal complete program source code

Here is the complete source code

```
include 'integer_kind_module.f90'
include 'precision_module.f90'
include 'timing_module.f90'

module sort_template_module_j3

template sort_template(k)

!   use precision_module
!   use integer_kind_module

  private

  public :: sort_data

! integer , constant  :: k
  integer , parameter :: k

  interface sort_data
    module procedure sort
  end interface

contains

  subroutine sort(x, n)

    use precision_module
    use integer_kind_module

    type(k) , intent(inout) :: x(:)
    integer , intent(in)     :: n

    call quicksort(1, n)

  contains

    recursive subroutine quicksort(l, r)

      implicit none
      integer, intent (in) :: l, r
      integer :: i, j
      type (k) :: v, t
```

```
! used to include the common sorting code
! include 'quicksort_include_code.f90'

        i = l
        j = r
        v = x(int((l+r)/2))
        do
          do while (x(i)<v)
            i = i + 1
          end do
          do while (v<x(j))
            j = j - 1
          end do
          if (i<=j) then
            t = x(i)
            x(i) = x(j)
            x(j) = t
            i = i + 1
            j = j - 1
          end if
          if (i>j) exit
        end do
        if (l<j) then
          call quicksort(l, j)
        end if
        if (i<r) then
          call quicksort(i, r)
        end if

    end subroutine

  end subroutine

end template

end module

program ch4702

  use precision_module
  use integer_kind_module
  use timing_module
  use sort_template_module_usa

  implicit none

  integer, parameter :: n   =      1000
  character *12       :: nn =   '1,000'
```

```
  character *80       :: report_file_name = 'ch3801_re-
port.txt'

  instantiate sort_template( sp)
  instantiate sort_template( dp)
  instantiate sort_template( qp)
  instantiate sort_template(i32)
  instantiate sort_template(i64)

  real (sp), allocatable, dimension (:) :: x_sp
  real (sp), allocatable, dimension (:) :: t_x_sp

  real (dp), allocatable, dimension (:) :: x_dp
  real (dp), allocatable, dimension (:) :: t_x_dp

  real (qp), allocatable, dimension (:) :: x_qp

  integer (i32), allocatable, dimension (:) :: y_i32
  integer (i64), allocatable, dimension (:) :: y_i64

  integer :: allocate_status = 0

  character *20, dimension (5) :: heading1 = &
  [ '   32 bit real', &
    '   32 bit int ', &
    '   64 bit real', &
    '   64 bit int ', &
    ' 128 bit real' ]

  character *20, dimension (3) :: &
  heading2 = [ '       Allocate ', &
               '       Random    ', &
               '       Sort      ' ]

  print *, 'Program starts'
  print *, 'N = ', nn
  call start_timing()

  open (unit=100, file=report_file_name)

  print *, heading1(1)

  allocate (x_sp(1:n), stat=allocate_status)
  if (allocate_status/=0) then
    print *, ' Allocate failed. Program terminates'
    stop 10
  end if
```

```
  print 100, heading2(1), time_difference()
  100 format (a20, 2x, f18.6)

  call random_number(x_sp)
  t_x_sp = x_sp

  print 100, heading2(2), time_difference()
  call sort_data(x_sp, n)
  print 100, heading2(3), time_difference()
  write (unit=100, fmt='(a)') ' First 10 32 bit reals'
  write (unit=100, fmt=110) x_sp(1:10)
110 format (5(2x,e14.6))

  print *, heading1(2)

  allocate (y_i32(1:n), stat=allocate_status)
  if (allocate_status/=0) then
    print *, 'Allocate failed. Program terminates'
    stop 30
  end if

  print 100, heading2(1), time_difference()
  y_i32 = int(t_x_sp*1000000000, i32)

  deallocate (x_sp)
  deallocate (t_x_sp)

  print 100, heading2(2), time_difference()
  call sort_data(y_i32, n)
  print 100, heading2(3), time_difference()
  write (unit=100, fmt='(a)') 'First 10 32 bit integers'
  write (unit=100, fmt=120) y_i32(1:10)
120 format (5(2x,i10))
  deallocate (y_i32)

  print *, heading1(3)

  allocate (x_dp(1:n), stat=allocate_status)
  if (allocate_status/=0) then
    print *, 'Allocate failed. Program terminates'
    stop 30
  end if

  allocate (t_x_dp(1:n), stat=allocate_status)
  if (allocate_status/=0) then
    print *, 'Allocate failed. Program terminates'
    stop 40
  end if
```



```
print 100, heading2(1), time_difference()
call random_number(x_dp)
t_x_dp = x_dp
print 100, heading2(2), time_difference()
call sort_data(x_dp, n)
print 100, heading2(3), time_difference()
write (unit=100, fmt='(a)') 'First 10 64 bit reals'
write (unit=100, fmt=110) x_dp(1:10)

print *, heading1(4)

allocate (y_i64(1:n), stat=allocate_status)
if (allocate_status/=0) then
  print *, 'Allocate failed. Program terminates'
  stop 40
end if

print 100, heading2(1), time_difference()
y_i64 = int(t_x_dp*1000000000000000_i64, i64)

deallocate (x_dp)
deallocate (t_x_dp)

print 100, heading2(2), time_difference()
call sort_data(y_i64, n)
print 100, heading2(3), time_difference()
write (unit=100, fmt='(a)') 'First 10 64 bit integers'
write (unit=100, fmt=120) y_i64(1:10)
deallocate (y_i64)

print *, heading1(5)

allocate (x_qp(1:n), stat=allocate_status)
if (allocate_status/=0) then
  print *, 'Allocate failed. Program terminates'
  stop 50
end if

print 100, heading2(1), time_difference()
call random_number(x_qp)
print 100, heading2(2), time_difference()
call sort_data(x_qp, n)
print 100, heading2(3), time_difference()
write (unit=100, fmt='(a)') 'First 10 128 bitreals'
write (unit=100, fmt=110) x_qp(1:10)

close (200)
```

```
  print *, 'Program terminates'
  call end_timing()

end program
```

17.4 diff output between the two examples

Here is the diff output between the two complete examples

```
5c5,21
< module sort_template_module_japan
---
> module sort_template_module_j3
>
> template sort_template(k)
>
> !  use precision_module
> !  use integer_kind_module
>
>   private
>
>   public :: sort_data
>
> ! integer , constant  :: k
>   integer , parameter :: k
>
>   interface sort_data
>     module procedure sort
>   end interface
9c25
<   generic subroutine sort(x, n)
---
>   subroutine sort(x, n)
14c30
<     type(i8,i16,i32,i64,sp,dp,qp) , intent(inout) :: x(:)
---
>     type(k) , intent(inout) :: x(:)
65c81
< program ch4701
---
> program ch4702
70c86
<   use sort_template_module_japan
---
>   use sort_template_module_j3
76c92,98
<   character *80       :: report_file_name = 'ch4701_re-
port.txt'
---
```

```
>    character *80       :: report_file_name = 'ch4702_re-
port.txt'
>
>    instantiate sort_template( sp)
>    instantiate sort_template( dp)
>    instantiate sort_template( qp)
>    instantiate sort_template(i32)
>    instantiate sort_template(i64)
```

As can be seen the implementation using the two new Fortran F202Y proposals is quite straightforward.

17.5 Line counts for the three sort modules

Here are the line counts for the three sort modules

```
  161     sort_data_module.f90
   27     quicksort_include_code.f90
```

The total line count for the sort_data_module with included quicksort common code is 161 + (7 * 27) -7

```
  333     Total  sort_data_module.f9
   79     sort_template_module_j3.f90
   62     sort_template_module_japan.f90
```

We now also have only one set of code to modify if we want to reimplement the sort algorithm, instead of the previous 7.

17.6 Acknowledgements

Thanks to John Reid who recommended getting in touch with Brad Richardson for more information about the J3 proposals for generics and templates in the next standard.

Thanks to Brad Richardson for providing the zip file which contained the slides from his presentation at the J3 and WG5 meeting at Manchester in June 2023. Thanks to Brad for also providing a link to his youtube presentation on generics and templates.

Thanks to John Reid for correcting the Japanese example.

‘Though this be madness, yet there is method in’t’

Shakespeare.

‘Plenty of practice’ he went on repeating, all the time that Alice was getting him on his feet again. ‘plenty of practice.’

The White Knight, Through the Looking Glass and What Alice Found There, Lewis Carroll.

18 Compilers used with compilation details

In this chapter we will look at the compilers we use on Windows, Linux and the UK university HPC system at Edinburgh.

We currently use the following compilers

- NAG
- Intel
- gfortran
- nvidia
- Cray

The NAG and Intel compilers we use natively on both Windows and Linux.

The gfortran compiler we use on Linux primarily. We use it on a native install (openSuSe linux), under Hyper-V (openSuSe, Redhat, ubuntu), and also under WSL (openSuSe and Ubuntu).

The Nvidia compiler we use under Linux. There is no Windows version at the moment. We use it under a native install (openSuSe), under Hyper-V (openSuSe and Redhat) and under WSL (openSuSe).

The Cray compiler we use on the HPC systems at Edinburgh.

18.1 Windows and Linux compile scripts

A small number of batch files (Windows) and shell scripts (linux) are available:

- Windows
  - gfortran_compile.bat
  - ifort_compile.bat
  - ifx_compile.bat
  - nag_compile.bat
- Linux
  - cray_compile.sh
  - gfortran_compile.sh
  - ifort_compile.sh
  - ifx_compile.sh

- nag_compile.sh
- nvidia_compile.sh

We recommend downloading the fourth edition update tar file and extracting all of the files. This should provide you with all of the files in the fourth edition and fourth edition update.

18.2 Reruns of examples from the fourth edition with current compilers

In this section we have reruns of some of the examples from the fourth edition with current compilers.

18.2.1 Chapter 33 - example 5, comparison of whole array, do loop, do concurrent and openmp

Here are the compiler details.

gfortran - linux

- GCC version 13.2.1 20230803
  [revision cc279d6c64562f05019e1d12d0d825f9391b5553]
- -mtune=generic -march=x86-64 -O2 -fopenmp
  -fpre-include=/usr/include/finclude/math-vector-fortran.h

gfortran - windows

- GCC version 13.2.0
- -mtune=generic -march=x86-64 -mthreads -O2 -fopenmp

Intel - linux

- Intel(R) Fortran Intel(R) 64 Compiler Classic for applications running on Intel
- (R) 64, Version 2021.9.0 Build 20230302_000000
- -O2 -fopenmp -o ch3305_ifort_linux.out

Intel - windows

- Intel(R) Fortran Intel(R) 64 Compiler Classic for applications running on Intel
- (R) 64, Version 2021.10.0 Build 20230609_000000
- /O2 /openmp /o:ch3305_ifort.exe

Nag - windows

- NAG Fortran Compiler Release 7.1(Hanzomon) Build 7110
- -O4 -openmp

Nvidia - linux

- nvfortran 23.9-0
- ch3305.f90 -fast -Mvect=simd -Mflushz -Mcache_align
  -Mno-signed-zeros -fopenmp -mp

Here are the summary timing figures.

ch3305.f90	Comparison of whole array, do loop, do concurrent and openmp					
	Memory		128 GB			

ch3305.f90	Comparison of whole array, do loop, do concurrent and openmp					
	CPU		Intel I9-10980XE			
	Cores		36			
	Nag	Intel	Intel	gfortran	gfortran	nvfortran
	windows	windows	linux	linux	windows	linux
	7.1-7110	2021.10.0	2021.9.0	13.2.1	13.2.0	23.9-0
Whole array	0.378274	0.196800	0.169849	0.191275	0.179287	0.170696
Do loop	0.185623	0.177500	0.180843	0.191207	0.179637	0.170382
Do concurrent	0.174196	0.039400	0.038133	0.178620	0.170870	0.170599
openmp	0.047436	0.042400	0.037865	0.045798	0.045414	0.045564

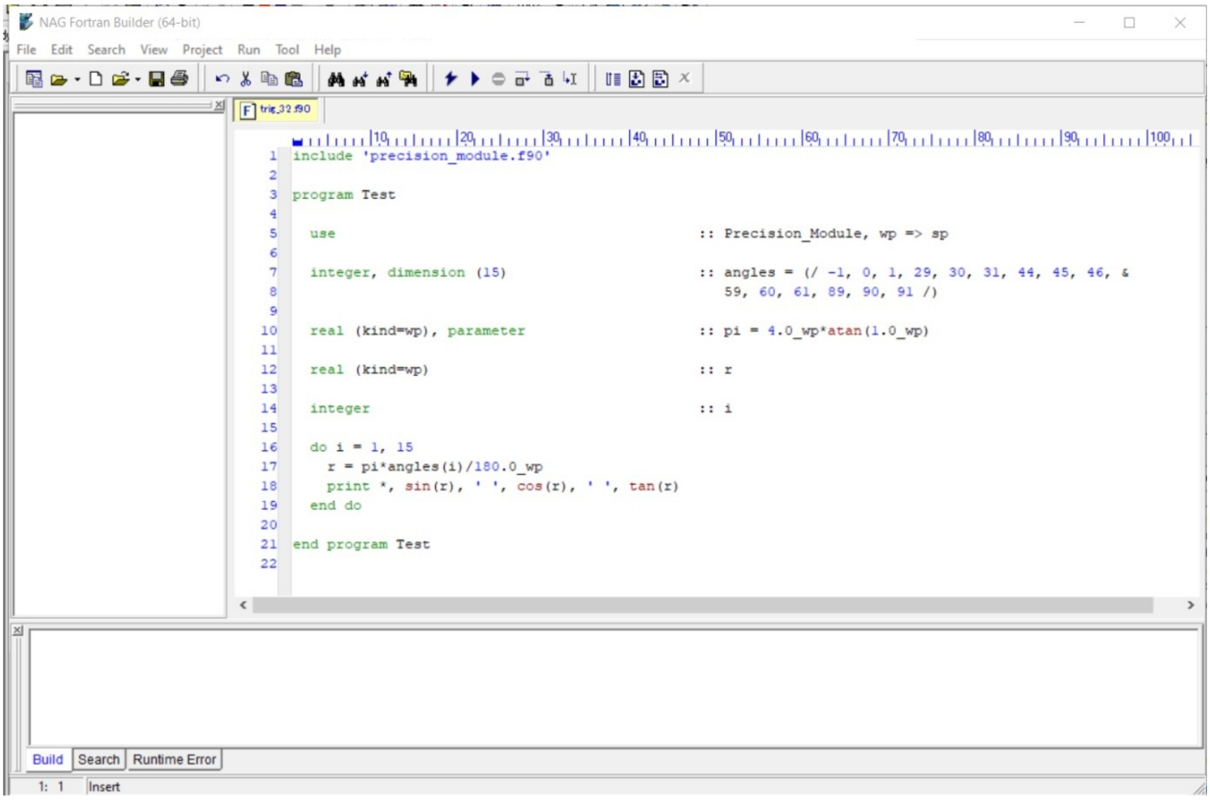
19 Development environments

We cover some of the development environment options in this chapter. Most Fortran compilers don't come with a bundled IDE. In this chapter we look at some options.

19.1 NAG

NAG provide Fortran Builder. We prepared an article for the August 2015 edition of Fortran Forum on Fortran Builder. The document is available on the FortranPlus site.

Here is a screen shot of Fortran Builder.



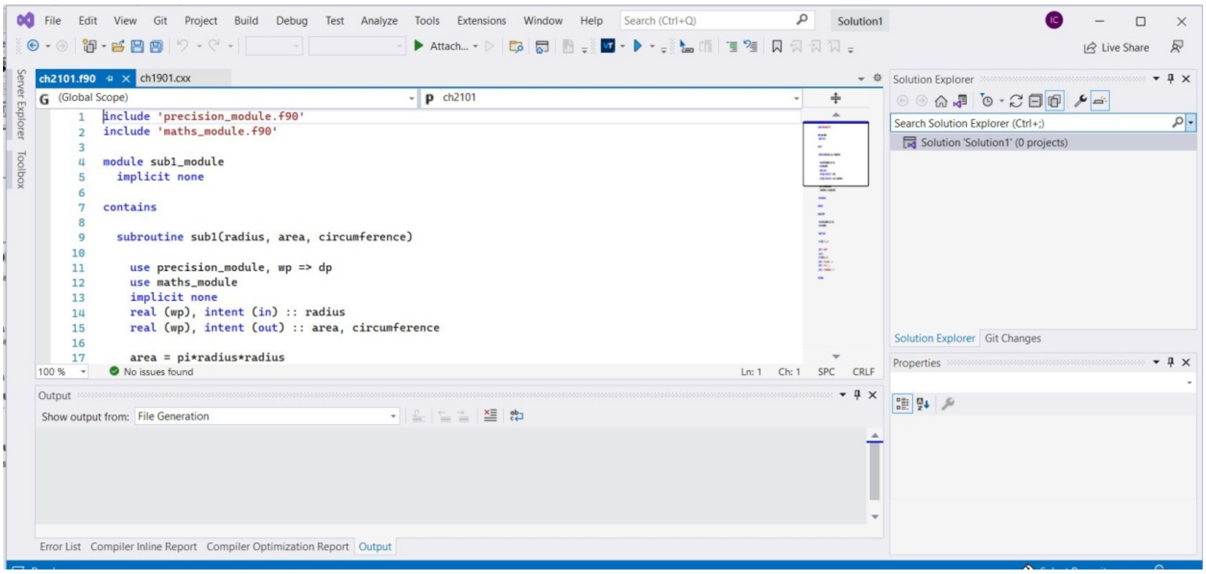
Consult the article for a comprehensive coverage.

19.2 Intel

On a Windows platform Intel integrates into Visual Studio. See the next section.

19.3 Microsoft Visual Studio

Here is a screen shot of a recent version of Visual Studio.



We recommend installing Visual Studio Community Edition before installing the Intel compiler suite. Visit

`https://visualstudio.microsoft.com/vs/community/`

for more details of Visual Studio.

Visit

`https://www.intel.com/content/www/us/en/developer/tools/oneapi/toolkits.html#gs.zbt6x0`

for details of the Intel toolkits. We recommend installing the Intel base toolkit plus the Intel HPC toolkit.

We recommend installing a range of products including the Microsoft C++ compiler and C# compiler.

19.4 Microsoft Visual Code

Microsoft also make Visual Code available.

Here is some blurb taken from their site.

- Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity).

Here is a link

`https://code.visualstudio.com/`

Versions are available for

- Windows

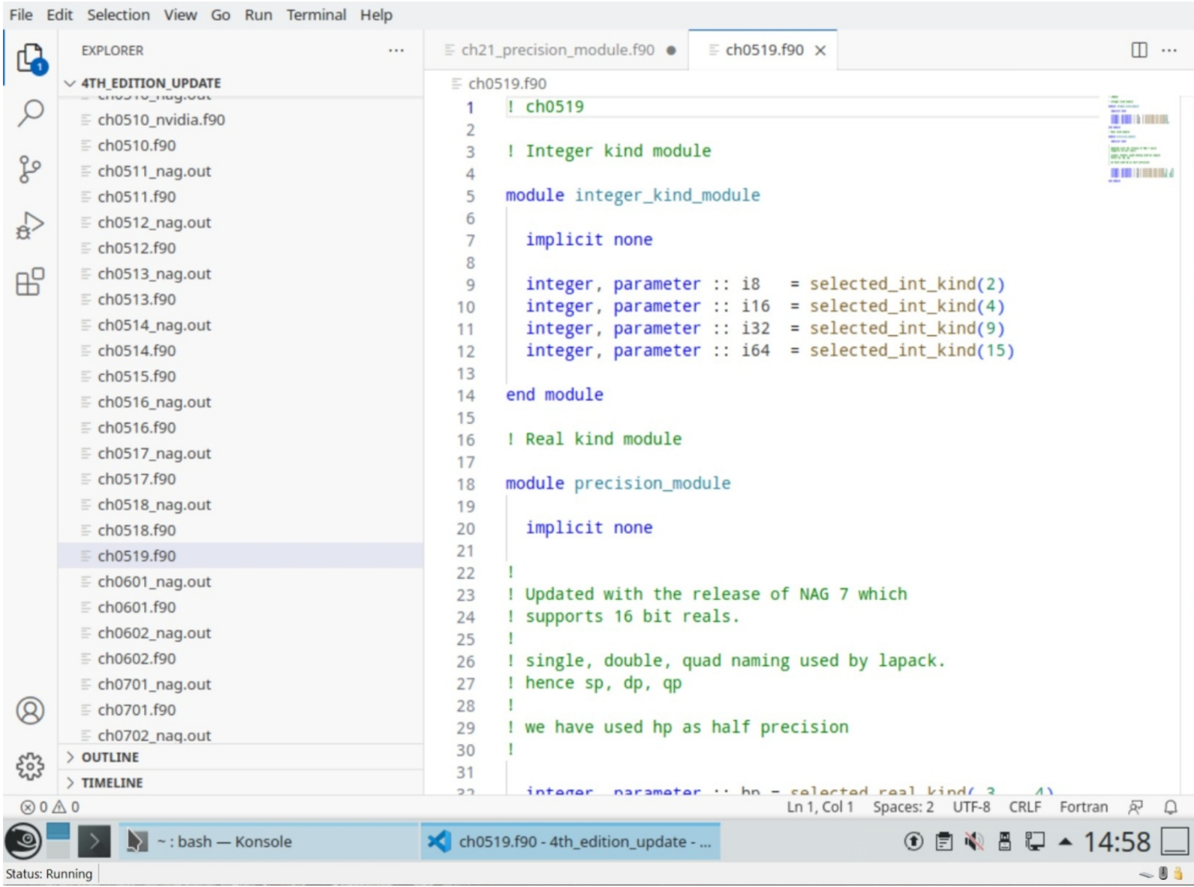
- Linux
- Apple Mac

Here are some of the Fortran extensions for Microsoft VSCode.

- 1 fortran
    - an extension for VS Code which provides support for the Fortran language. Xavier Hahn
- 2 Modern Fortran
    - Fortran language support, syntax high lighting, Language Server Support, debugging etc. The Fortran Programming Language.
- 3 Fortran Intellisense
    - VSCode interface to the Fortran language server.
- 4 Fortran Breakpoint support
    - Add breakpoint support for Fortran. ekibun
- 5 fortran - ekon
    - An extension for VS Code which provides syntax high-light support for the Fortran Language. Ekon Benefits.
- 6 vscode-modern-fortran-formatter
    - Modern Fortran Formatter using fprettify. yukiuuh.

One or more of these may be installed.

Here is a screen shot on a Linux distribution.



Here is a screen shot. taken from a Windows installation.