ISO/IEC JTC 1/SC 22/WG 5 "Fortran"
Convenorship: **ANSI**
Convenor: **Lionel Steve Mr**

DIN Suggestions for F202Y

Document type	Related content	Document date	Expected action
General / Other		2024-02-04	

Description

This is the formal list of F202Y feature requests from Germany. Please read, consider, and discuss them as we will be voting on them at the June Berkeley meeting.

Fortran 202y feature suggestions from DIN

Date: January, 2024

Author: Reinhold Bader

With the expectation that the final feature list for the next Fortran Standard will be voted on by the 2024 meeting of the International Fortran Standards Committee JTC1/SC22/WG5 (https://wg5-fortran.org/), hereby a list of suggestions from Germany/DIN is submitted for that vote.

Contents

DIN-1 – Execution of collective procedures on a specified team

Proposed by: R.Bader

Formal Requirement

The collective subroutines from Fortran 2018 should support execution in a team that is not the current team.

Rationale

If e.g. a reduction operation across the initial team's images is needed within the context of a CHANGE TEAM construct, this is rather cumbersome to do with the current Fortran semantics. Either it is necessary to exit the CHANGE TEAM construct (this permits a single CO_REDUCE invocation but can impose significant overhead on handling of team-specific data), or the reduction must be manually performed in two phases: first, a team-specific call to CO_REDUCE, followed by an assembly across teams (this leads to code that is hard to understand and maintain).

Further, the ability to perform a reduction on a subset of images corresponding to a defined team variable without needing to execute a CHANGE TEAM statement is considered useful.

Specification

Add an optional TEAM argument of type TEAM_TYPE to all collective subroutines and say that execution of the subroutine applies for the specified team if the argument is present.

Further remarks

See https://j3-fortran.org/doc/year/22/22-163.txt for the original (deferred) paper, which also includes a suggested set of edits for the already existing collectives.

DIN-2 – extend C interoperability to support UNION types

Proposed by: Steve Lionel

Formal Requirement

It should be possible to directly define and access components of C union types from within Fortran.

Rationale

Many non-Fortran APIs make use of union types. Because the existing C interoperability in Fortran does not cover such types, writing significant amounts of glue code is necessary to work around this gap in the semantics.

Further remarks

DIN supports the already existing suggestion for this, see https://github.com/j3-fortran/fortran_proposals/issues/188. Many compilers already support the feature as a compiler extension. The feature should not be available for non-interoperable types.

DIN-3 – support execution on APUs

Proposed by: R. Bader

Formal Requirements

It should be possible to execute suitably defined code sections on Accelerated Processing Units if such are available. It should be possible for the programmer to determine the availability of such units, and control specific execution on them. Support for asynchronous execution should be included, to enable concurrent execution of code operating on independent data on multiple APUs, as well as on the host CPU.

Rationale

Technological development of computational hardware throughout the last decade indicates that APUs are here to stay, to a large part due to energy efficiency reasons. Currently, support for the use of such hardware is largely implementation dependent, and is limited to a small set of constructs like DO CONCURRENT, or depends on (sometimes very complex) semantics defined outside Fortran proper, like OpenMP.

Specifications (sketch)

The following types of code section should be enabled for APU processing:

1. DO CONCURRENT loop bodies;
2. BLOCK constructs that call PURE (or SIMPLE?) procedures, or contain statements inside the block that do not produce side effects (this likely needs some additional definition of suitable semantics). Such a construct nested inside a loop would permit concurrent processing of irregular data structures.

The enablement is programmer-directed, and could be achieved by

- adding the keyword ASYNCHRONOUS(*id*) to the loop or block statement, where *id* is an integer variable suitable for later synchronization via a WAIT statement;
- adding a modifier DEVICE(*device_name*, *device_partition*) to the loop or block statement. The variables *device_name* (of a yet-to-be defined intrinsic derived type) and *device_partition* (of type INTEGER) require initialization by intrinsic functions that need to be newly created.

In case no APUs are available, a default execution mode for the CPU should be supported. It would remain implementation-dependent whether or not execution is actually asynchronous (e.g., in case two blocks are scheduled to the same partition of the same device, initiation of the second one would need to wait for completion of the first). Completion of the execution would be signaled by a subsequent WAIT statement with the respective *id* variable as argument, or a blanket WAIT in case all pending executions should complete.

Local variables are expected to be created on device memory. An exception model may be helpful in dealing with failures to supply the needed memory resources. For pre-existing variables from the host environment, the ASYNCHRONOUS attribute might be useful, or (alternatively?) a locality specification.

Atomic operations (separate from those specified for coarray programming) should be supported. Support for memory alignment settings for various architectures should be added.

A facility for identifying which devices are accessible from which coarray images of a program is needed, such that the programmer can perform appropriate device management.

Further comments

An earlier proposal similar in spirit is https://github.com/j3-fortran/fortran_proposals/issues/271

DIN-4 Generic processing of assumed rank objects

Proposed by: R. Bader

Rationale

The concept of assumed rank permits definition of interfaces that are rank-agnostic. The addition of the SELECT RANK block construct permits definitions and references to the object by resolving at run time to whatever rank the actual argument has. This is fine in case the array rank e.g. reflects different problem dimensions that require different algorithms for its solution. However, there are also cases in which uniform treatment of the argument's data irrespective of its rank is needed. This is currently cumbersome to do.

Formal Requirement

Allow generic processing of assumed-rank arguments, possibly under suitable restrictions.

Specifications

None yet. https://github.com/j3-fortran/fortran_proposals/issues/144 gives an initial idea for this, but this is likely too limited, and outdated anyway, since Fortran 2023 defines new array processing concepts that could be used or extended for this purpose. DIN would also consider it acceptable if the concept was realized within the generics facility planned for F202y, see https://github.com/j3-fortran/generics/blob/main/J3-Papers/use_case_rank_agnostic_loops.txt, or alternatively with the generic function concept referenced in DIN-5 below.

DIN-5 Generic procedures

Proposed by: Hidetoshi IwashitaDIN hereby indicates its support for introducing generic procedures as outlined in J3 papers https://j3-fortran.org/doc/year/23/23-223r2.txt and https://j3-fortran.org/doc/year/23/23-244r1.txt.

DIN notes that the generics facilities being developed (cf. https://github.com/j3-fortran/generics) is very powerful, but appears not easy to explain and use. Some consideration should be given to keep the feature orthogonal to that outlined above.

DIN-6 Proposal for UNSIGNED type

Proposal by: Thomas König

Introduction

Unsigned integers are a basic data type used in many programming languages, like C. Arithmetic on them is typically performed modulo 2^n for a datatype with n bits. They are useful for a range of applications, including, but not limited to

- hashing

- cryptography (including multi-precision arithmetic)

- image processing

- binary file I/O

- interfacing to the operating system

- signal processing

- data compression

Introduction of unsigned integers should not repeat the mistakes of languages like C, and syntax and functionality should be familiar to people who today use unsigned types in other programming languages.

Further information, use cases, basic specifications

These are contained in the J3 paper https://j3-fortran.org/doc/year/24/24-102.txt