IEC 60559 SUPPLEMENTARY ATTRIBUTES

N2421
WG 14 - Ithaca
October 21 – 25, 2019

C FP group

Proposal for C2X

- N2407
- Add ISO/IEC TS 18661-5abc supplementary attributes to C2X.
- IEC 60559 recommends language standards provide block-scope attributes to control expression evaluation, value-changing optimizations, and reproducible results.
- TS 18661-5abc provides these attributes as standard pragmas in **<fenv.h>**, like existing FP pragmas.
- This proposal does not include TS 18661-5d alternate exception handling.
- Attributes are intended to address three problems with FP programming …

Problem 1

Porting floating-point code between platforms and tool sets, including debugging ported code

- Program development tools typically provide controls to manage optimizations and evaluation methods.
- These controls are implementation specific, both in syntax and semantics, and are often vaguely defined.
- It's difficult to impossible to map controls on one system to equivalent ones on another.
- Standard pragmas for evaluation methods and optimizations are intended to address this problem.

Problem 2

Balancing performance against precision and reliability

- Current implementation-specific controls are usually compiler options that apply to the whole translation unit.
- However, many programs need aggressive optimizations only for relatively small performance-critical blocks.
- Applying the optimizations where they aren’t needed unnecessarily risks floating-point anomalies throughout the entire program.
- Similarly, extra precision might be needed only in relatively small precision-critical blocks.
- Using extra precision throughout the program might unnecessarily degrade performance.
- The block-scope semantics of the pragmas address this problem.

Problem 3

Obtaining reproducible results (on same or different platforms)

- Some users want results that are the same on different platforms and that remain the same after tool set updates.
- Usually variations in floating-point results are harmless, but not always, and the cost to determine whether a difference is benign or the result of a serious bug can be great.
- Potential causes of differences in floating-point results are many and difficult for most programmers to avoid.
- A pragma and guidance for reproducible results is intended to help with this problem.

5a – Evaluation methods

The following pragmas provide the preferredWidth attributes recommended for language standards by IEC 60559:

#pragma STDC FENV_FLT_EVAL_METHOD *width*

- *width* indicates a supported evaluation method for which macro **FLT_EVAL_METHOD** has the value *width*.
- Requires support for *width* equal 0 (evaluate to wider of **float** and type), allows support for other values.

5a – Evaluation methods (2)

#pragma STDC FENV_DEC_EVAL_METHOD *width*

- Like **FENV_FLT_EVAL_METHOD**, but for decimal.
- *width* indicates a supported evaluation method for which macro **DEC_EVAL_METHOD** has the value *width*.
- Requires support for *width* equal 1 (evaluate to wider of **_Decimal64** and type), allows support for other values.

5a also specifies a user definable macro **__STDC_TGMATH_OPERATOR_EVALUATION__** to have tgmath macros follow the evaluation method like operators do -- to allow wide evaluation that is consistent for all FP operations.

5b – Optimizations

The following pragmas provide value-changing-optimization attributes recommended for language standards by IEC 60559:

#pragma STDC FENV_ALLOW_ASSOCIATIVE_LAW *on-off-switch*

- $x + (y + z) = (x + y) + z$
- $x * (y * z) = (x * y) * z$

#pragma STDC FENV_ALLOW_DISTRIBUTIVE_LAW *on-off-switch*

- $x *(y + z) = (x * y) + (x * z)$
- $x *(y - z) = (x * y) - (x * z)$
- $(x + y) / z = (x / z) + (y / z)$
- $(x - y) / z = (x / z) - (y / z)$

5b – Optimizations (2)

#pragma STDC FENV_ALLOW_MULTIPLY_BY_RECIPROCAL *on-off-switch*

- x / y = x *(1 / y)

#pragma STDC FENV_ALLOW_CONTRACT_FMA *on-off-switch*

- Contract (compute with just one rounding) floating-point multiply and add or subtract (with the result of the multiply).
- x * y + z   x * y − z
- x + y * z   x − y * z

#pragma STDC FENV_ALLOW_CONTRACT_OPERATION_CONVERSION *on-off-switch*

- Contract a floating-point operation and a conversion (of the result of the operation), e.g., flt_var = dbl_var * dbl_var.

5b – Optimizations (3)

#pragma STDC FENV_ALLOW_CONTRACT *on-off-switch*

- Equivalent to **FP_CONTRACT** pragma in **<math.h>** - includes effects of two “contract” pragmas above.

#pragma STDC FENV_ALLOW_ZERO_SUBNORMAL *on-off-switch*

- Replace subnormal operands and results by zero.

#pragma STDC FENV_ALLOW_VALUE_CHANGING_OPTIMIZATION *on-off-switch*

- Equivalent to all the optimization pragmas above.

Optimization pragmas allow but do not require the optimizations.

5c – Reproducibility

The following pragma provides the reproducible-results attribute recommended for language standards by IEC 60559:

#pragma STDC FENV_REPRODUCIBLE *on-off-switch*

Implies effects of

- **#pragma STDC FENV_ACCESS ON**
- **#pragma STDC FENV_ALLOW_VALUE_CHANGING_OPTIMIZATION OFF**

and if **__STDC_IEC_60559_BFP__** is defined

- **#pragma STDC FENV_FLT_EVAL_METHOD 0**

and if **__STDC_IEC_60559_DFP__** is defined

- **#pragma STDC FENV_DEC_EVAL_METHOD 1**

5c – Reproducibility

- Recommends a diagnostic message if the source code uses a language or library feature whose results may not be reproducible.

- Includes guidelines for code intended to be reproducible, e.g.,
  - The code does not contain any use that may result in undefined behavior. The code does not depend on any behavior that is unspecified, implementation-defined, or locale-specific.
  - The code does not use the **long double** type.
  - The code does not depend on the payloads (F.10.13) or sign bits of quiet NaNs.
  - The code does not use signaling NaNs.

…

Notes

- A low-quality or initial implementation could have a conformance mode where only **FLT_EVAL_METHOD** equal 0 is supported, optimizations are disabled, and pragmas are ignored.
- TS 18661-5abc could be recast as an annex to C2X.
- 5a and 5b are essentially independent of each other and of 5c. 5c depends on 5a and 5b, at least as currently written.

Publication

ISO/IEC TS 18661-5:2016, Information Technology — Programming languages, their environments, and system software interfaces — Floating-point extensions for C — Part 5: Supplementary attributes.