Technical API Documentation: `high_precision_integer_mod`

Akhil Akkapelli

July 26, 2025

Contents

1 Introduction

This document provides a detailed technical API documentation for the Fortran module `high_precision_integer_mod`. This module implements an arbitrary-precision integer arithmetic system by defining the `high_precision_int` derived type. It includes routines for conversion, and overloads standard arithmetic and comparison operators to enable seamless high-precision computations. Large integers are represented as coefficients in base 2^{32}.

2 High-Precision Integer Representation: `high_precision_int`

A high-precision integer N is modeled as a 4-tuple:

$$\text{hpi} = (\mathbf{c}, \ell, L, s)$$

with the following components:

- $\mathbf{c} = (c_1, \ldots, c_L) \in [0, B)^L$: array of digits in base $B = 2^{32}$
- $\ell \in \{1, 2, \ldots, L\}$: number of significant digits
- $L \in \mathbb{N}$: total allocated digit capacity
- $s \in \{-1, 0, +1\}$: sign of the number

The value represented is:

$$N = \begin{cases} 0, & \text{if } s = 0 \\ s \cdot \sum_{i=1}^{\ell} c_i B^{i-1}, & \text{otherwise} \end{cases}. \tag{1}$$

This representation satisfies:

- $0 \leq c_i < B \quad \forall i \in \{1, \ldots, L\}$
- If $s = 0$, then $\ell = 1$ and $c_1 = 0$
- If $s \neq 0$, then $c_\ell \neq 0$

Fortran Type Definition

This mathematical structure is implemented using the following Fortran derived type:

```fortran
TYPE high_precision_int(len)
  INTEGER, LEN      :: len           ! L : total capacity
  INTEGER           :: ncoeffs = 0   ! l : significant length
  INTEGER(KIND=8)   :: coeffs(len)   ! c_1, c_2, ..., c_L: base-B digits
  INTEGER(KIND=1)   :: sign          ! s: sign
END TYPE high_precision_int
```

3 Normalization Procedure: normalize_hpi

The subroutine **normalize_hpi** transforms a high-precision number N into its canonical internal representation by enforcing coefficient bounds, propagating carry, and trimming insignificant digits.

Input/Output

$$\text{Input \& Output: } \texttt{hpi}(\mathbf{c}, \ell, L, s), \quad N = s \cdot \sum_{i=1}^{\ell} c_i B^{i-1}$$

```fortran
SUBROUTINE normalize_hpi(hpi)
  TYPE(high_precision_int(*)), INTENT(INOUT) :: hpi

  INTEGER(KIND=8), PARAMETER :: MASK32 = INT(Z'FFFFFFFF', KIND=8)

  INTEGER(KIND=8) :: carry              ! Overflow from previous digit
  INTEGER(KIND=8) :: current_coeff_val ! Intermediate value for digit + carry
  INTEGER         :: i                  ! Index into coefficient array
  INTEGER         :: last_nonzero_idx   ! Tracks final non-zero digit index
```

Step 1: Handle zero-length input

If $\ell = 0$,

$$c_1 = 0, \quad \ell = 1, \text{ and } s = 0 \quad \Rightarrow \quad \underline{\texttt{hpi} = ([0], 1, L, 0)}$$

```fortran
  IF (hpi%ncoeffs == 0) THEN
    hpi%ncoeffs   = 1
    hpi%coeffs(1) = 0_8
    hpi%sign      = 0_1
    RETURN
  END IF
```

Step 2: Normalize coefficients and propagate carry

$$\text{carry} := 0, \quad i := 1, \quad \text{last\_nonzero\_idx} := 1$$

```fortran
  carry = 0_8
  i = 1
  last_nonzero_idx = 1
```

For each index i:

$$i > L \Rightarrow \text{ERROR (exceeds allocated capacity)}$$

$$v_i = \begin{cases} c_i + \text{carry} & \text{if } i \le \ell \\ \text{carry} & \text{if } i > \ell \Rightarrow \ell := i \end{cases}$$

$$c_i := v_i \bmod B, \quad \text{carry} := \left\lfloor \frac{v_i}{B} \right\rfloor$$

```
    DO WHILE (i <= hpi%ncoeffs .OR. carry /= 0_8)
      IF (i > hpi%len) THEN
        STOP "FATAL ERROR in normalize_hpi: Overflow. Integer exceeds allocated
    length capacity."
      END IF

      IF (i <= hpi%ncoeffs) THEN
        current_coeff_val = hpi%coeffs(i) + carry
      ELSE
        current_coeff_val = carry
        hpi%ncoeffs = i
      END IF

      hpi%coeffs(i) = IAND(current_coeff_val, MASK32)
      carry = ISHFT(current_coeff_val, -32)
```

Track the last non-zero index:

$$c_i \neq 0 \Rightarrow \text{last\_nonzero\_idx} := i$$

```
      IF (hpi%coeffs(i) /= 0_8) THEN
        last_nonzero_idx = i
      END IF

      i = i + 1
    END DO
```

Step 3: Finalize logical length

$$\ell := \text{last\_nonzero\_idx}$$

```
    hpi%ncoeffs = last_nonzero_idx
```

Step 4: Adjust the sign

$$s := \begin{cases} 0 & \text{if } \ell = 1 \text{ and } c_1 = 0 \\ +1 & \text{if } s = 0 \text{ and } (\ell > 1 \text{ or } c_1 \neq 0) \\ s & \text{otherwise} \end{cases}$$

```
    IF (hpi%ncoeffs == 1 .AND. hpi%coeffs(1) == 0_8) THEN
      hpi%sign = 0_1
    ELSE IF (hpi%sign == 0_1) THEN
      hpi%sign = 1_1
    END IF
  END SUBROUTINE normalize_hpi
```

Resulting Representation

After normalization, the high-precision integer is in canonical form:

$$\underline{\texttt{hpi} = (\mathbf{c}, \ell, L, s)}$$

where:

- $\ell \geq 1$ and $c_\ell \neq 0$ if $s \neq 0$
- $\ell = 1$, $c_1 = 0$, and $s = 0$ if the number is zero
- All coefficients satisfy $0 \leq c_i < B$

Thus, the number represented is:

$$N = s \cdot \sum_{i=1}^{\ell} c_i B^{i-1}$$

4 HPI Constructor from Integer: `new_hpi_from_integer`

Constructs a high-precision number from a 64-bit signed integer by extracting base-2^{32} digits and assigning sign and logical length accordingly.

Input/Output

$$\text{Input: } x \in \mathbb{Z}^{64}$$

$$\text{Output: } \texttt{hpi}(\mathbf{c}, \ell, L, s), \quad \text{represents } x \text{ using } \texttt{hpi}\text{: } x = s \cdot \sum_{i=1}^{\ell} c_i B^{i-1}, \text{ with base } B = 2^{32}$$

```
FUNCTION new_hpi_from_integer(x_in) RESULT(hpi_out)
  INTEGER(KIND=8), INTENT(IN)              :: x_in

  TYPE(high_precision_int(:)), ALLOCATABLE :: hpi_out

  INTEGER          :: min_required_len
  INTEGER(KIND=1)  :: final_sign
  INTEGER(KIND=8)  :: mag_x
```

Constants and Parameters

$$B := 2^{32}, \quad M := -2^{63}, \quad \text{MASK32} := 2^{32} - 1 = 0\text{xFFFFFFFF}$$

```
  INTEGER(KIND=8), PARAMETER :: HIGH_PRECISION_BASE = 2_8**32
  INTEGER(KIND=8), PARAMETER :: MASK32 = INT(Z'FFFFFFFF', KIND=8)
  INTEGER(KIND=8), PARAMETER :: MOST_NEGATIVE_I8 = -HUGE(0_8) - 1_8
```

Step 1: Determine required coefficient length

$$\ell_{\min} = \begin{cases} 1 & \text{if } x = 0 \\ 2 & \text{else if } x = M \\ 1 & \text{else if } |x| < B \\ 2 & \text{else} \end{cases}$$

```
  IF (x_in == 0_8) THEN
    min_required_len = 1
  ELSE IF (x_in == MOST_NEGATIVE_I8) THEN
    min_required_len = 2
  ELSE IF (ABS(x_in) < HIGH_PRECISION_BASE) THEN
    min_required_len = 1
  ELSE
    min_required_len = 2
  END IF
```

Step 2: Allocate structure and Initialize coefficeints

$$L := \ell, \quad \text{initialize } \mathbf{c} = \mathbf{0}, \quad \ell = \ell_{\min}$$

```
  ALLOCATE(high_precision_int(len=min_required_len) :: hpi_out)
  hpi_out%coeffs = 0_8
  hpi_out%ncoeffs = min_required_len
```

Step 3: Assign sign and coefficient values

$$\text{If } x = 0: \quad s = 0, \quad \mathbf{c} = [0]$$

```
    IF (x_in == 0_8) THEN
      hpi_out%sign = 0_1
```

$$\text{Else if } x = M: \quad s = -1, \quad c_1 = 0, \quad c_2 = B \gg -1$$

```
    ELSE IF (x_in == MOST_NEGATIVE_I8) THEN
      hpi_out%sign = -1_1
      hpi_out%coeffs(1) = 0_8
      hpi_out%coeffs(2) = ISHFT(HIGH_PRECISION_BASE, -1)
```

$$\text{Else: } s = \text{sign}(x), \quad |x| = \text{mag}_x, \quad c_1 = |x| \bmod B, \quad c_2 = \left\lfloor \frac{|x|}{B} \right\rfloor \text{ if } |x| \geq B$$

```
    ELSE
      hpi_out%sign = SIGN(1_1, x_in)
      mag_x = ABS(x_in)
      hpi_out%coeffs(1) = IAND(mag_x, MASK32)
      IF (min_required_len == 2) THEN
        hpi_out%coeffs(2) = ISHFT(mag_x, -32)
      END IF
    END IF

  END FUNCTION new_hpi_from_integer
```

$$\underline{\texttt{hpi} = (\mathbf{c}, \ell, L, s)} \Rightarrow N = s \cdot \sum_{i=1}^{\ell} c_i B^{i-1} \equiv x$$

5 HPI to Integer Converter: `hpi_to_integer`

Converts a high-precision number to a 64-bit integer, if the value is within its representable range.

Input/Output

$$\text{Input: } \texttt{hpi} = (\mathbf{c}, \ell, L, s)$$

$$\text{Output: } x \in \mathbb{Z}^{64}, \quad \text{such that if representable, } x = s \cdot \sum_{i=1}^{\ell} c_i' B^{i-1}, \text{ with base } B = 2^{32}$$

```
    INTEGER(KIND=8), PARAMETER :: MAX_POS_INT8 = 2_8**31 - 1_8
    INTEGER(KIND=8), PARAMETER :: MAX_NEG_INT8 = 2_8**31

    IF (hpi%sign == 0_1) THEN
```

Constants and Parameters

$$C_1^{\text{max+}} := 2^{31} - 1, \quad C_1^{\text{max-}} := 2^{31}$$

```
    SELECT CASE (hpi%ncoeffs)
    CASE (1)
```

Step 1: Check for zero

$$\text{If } s = 0 \Rightarrow \texttt{hpi}([0], 1, L, 0) \equiv \underline{x := 0}$$

```
    CASE (2)
      c0 = hpi%coeffs(1)
      c1 = hpi%coeffs(2)
```

Step 2: Compute absolute value based on number of coefficients

$$\text{If } \ell = 1 : |N| = c_1$$

```
          STOP "FATAL ERROR in hpi_to_integer: Positive value is too large to fit in an INTEGER(KIND=8)."
        ELSE IF (hpi%sign == -1_1 .AND. (c1 > ISHFT(MAX_NEG_INT8, 1) .OR. (c1 == MAX_NEG_INT8 .AND. c0 > 0_8))) THEN
          STOP "FATAL ERROR in hpi_to_integer: Negative value is too small to fit in an INTEGER(KIND=8)."
```

$$\text{Else if } \ell = 2 : \quad c_0' := c_0, \quad c_1' := c_1$$

```
        END IF

      abs_val_as_int8 = c0 + ISHFT(c1, 32)
```

$$\text{If } s = +1 \quad \wedge \quad (c_1' \cdot B + c_0') > C_1^{\text{max+}} \Rightarrow \text{ERROR: overflow}$$

$$\text{Else if } s = -1 \quad \wedge \quad (c_1' \cdot B + c_0') > C_1^{\text{max-}} \Rightarrow \text{ERROR: underflow}$$

```
      STOP "FATAL ERROR in hpi_to_integer: Value has more than 2 coefficients and cannot fit in an INTEGER(KIND=8)."
    END SELECT

    x = abs_val_as_int8 * hpi%sign
```

$$|N| = c_0' + c_1' \cdot B$$

$$\text{Else if } \ell > 2 \Rightarrow \text{STOP: cannot fit in 64-bit signed integer}$$

```
  FUNCTION hpi_to_string(hpi_in) RESULT(str_out)
    TYPE(high_precision_int(*)), INTENT(IN) :: hpi_in
    CHARACTER(LEN=:), ALLOCATABLE           :: str_out
```

Step 3: Apply sign

$$x = s \cdot |N|$$

```
    ! --- Local variables ---
    TYPE(high_precision_int(:)), ALLOCATABLE :: temp_hpi
    ! Process number in chunks of 9 digits for efficiency. 10**9 fits in a 32-bit integer.
```

Resulting Output

$$\underline{x = s \cdot \sum_{i=1}^{\ell} c_i' B^{i-1},} \quad \text{where} \quad x \in \mathbb{Z}^{64} \equiv [-2^{63}, 2^{63} - 1]$$