8.6 Floating point arithmetic

(Original section[1])

GPUs today perform most arithmetic operations in the programmable processor cores using IEEE 754-compatible single precision 32-bit floating-point operations (see COD Chapter 3 (Arithmetic for Computers)). The fixed-point arithmetic of early GPUs was succeeded by 16-bit, 24-bit, and 32-bit floating-point, then IEEE 754-compatible 32-bit floating-point. Some fixed-function logic within a GPU, such as texture-filtering hardware, continues to use proprietary numeric formats. Recent GPUs also provide IEEE 754-compatible double-precision 64-bit floating-point instructions.

Supported formats

The IEEE 754 standard for floating-point arithmetic specifies basic and storage formats. GPUs use two of the basic formats for computation, 32-bit and 64-bit binary floating-point, commonly called single precision and double precision. The standard also specifies a 16-bit binary storage floating-point format, *half precision*. GPUs and the Cg shading language employ the narrow 16-bit half data format for efficient data storage and movement, while maintaining high dynamic range. GPUs perform many texture filtering and pixel blending computations at half precision within the texture filtering unit and the raster operations unit. The OpenEXR high dynamic-range image file format developed by Industrial Light and Magic [2003] uses the identical half format for color component values in computer imaging and motion picture applications.

Half precision: A 16-bit binary floating-point format, with 1 sign bit, 5-bit exponent, 10-bit fraction, and an implied integer bit.

Basic arithmetic

Common single-precision floating-point operations in GPU programmable cores include addition, multiplication, *multiply-add*, minimum, maximum, compare, set predicate, and conversions between integer and floating-point numbers. Floating-point instructions often provide source operand modifiers for negation and absolute value.

Multiply-add (MAD): A single floating-point instruction that performs a compound operation: multiplication followed by addition.

The floating-point addition and multiplication operations of most GPUs today are compatible with the IEEE 754 standard for single precision FP numbers, including *not-a-number* (NaN) and infinity values. The FP addition and multiplication operations use IEEE round-to-nearest-even as the default rounding mode. To increase floating-point instruction throughput, GPUs often use a compound multiply-add instruction (`mad`). The multiply-add operation performs FP multiplication with truncation, followed by FP addition with round-to-nearest-even. It provides two floating-point operations in one issuing cycle, without requiring the instruction scheduler to dispatch two separate instructions, but the computation is not fused and truncates the product before the addition. This makes it different from the fused multiply-add instruction discussed in COD Chapter 3 (Arithmetic for Computers) and typically flush denormalized source operands to sign-preserved zero, and they flush results that underflow the target output exponent range to sign-preserved zero after rounding.

Specialized arithmetic

GPUs provide hardware to accelerate special function computation, attribute interpolation, and texture filtering. Special function instructions include cosine, sine, binary exponential, binary logarithm, reciprocal, and reciprocal square root. Attribute interpolation instructions provide efficient generation of pixel attributes, derived from plane equation evaluation. The *special function unit* (SFU) introduced in COD Section B.4 (Multithreaded multiprocessor architecture) computes special functions and interpolates planar attributes [Oberman and Siu, 2005].

Special function unit (SFU): A hardware unit that computes special functions and interpoles planar attributes.

Several methods exist for evaluating special functions in hardware. It has been shown that quadratic interpolation based on Enhanced Minimax Approximations is a very efficient method for approximating functions in hardware, including reciprocal, reciprocal square-root, $\log_2 x$, 2^x, sin, and cos.

We can summarize the method of SFU quadratic interpolation. For a binary input operand X with *n*-bit significand, the significand is divided into two parts: X_u is the upper part containing *m* bits, and X_l is the lower part containing *n-m* bits. The upper *m* bits X_u are used to consult a set of three lookup tables to return three finite-word coefficients C_0, C_1, and C_2. Each function to be approximated requires a unique set of tables. These coefficients are used to approximate a given function f(X) in the range $X_u \le X < X_u + 2^{-m}$ by evaluating the expression:

$$f(X) = C_0 + C_1 X_1 + C_2 X_1^{\,2}$$

The accuracy of each of the function estimates ranges from 22 to 24 significand bits. Example function statistics are shown in the figure below.

Figure 8.6.1: Special function approximation statistics (COD Figure B.6.1).

For the NVIDIA GeForce 8800 *special function unit* (SFU).

Function	Input interval	Accuracy (good bits)	ULP* error	% exactly rounded	Monotonic
1/x	[1, 2)	24.02	0.98	87	Yes
1/sqrt(x)	[1, 4)	23.40	1.52	78	Yes
2^x	[0, 1)	22.51	1.41	74	Yes
$\log_2 x$	[1, 2)	22.57	N/A**	N/A	Yes
sin/cos	[0, π/2)	22.47	N/A	N/A	No

*ULP: unit in the last place.**N/A: not applicable.

The IEEE 754 standard specifies exact-rounding requirements for division and square root; however, for many GPU applications, exact compliance is not required. Rather, for those applications, higher computational throughput is more important than last-bit accuracy. For the SFU special functions, the CUDA math library provides both a full accuracy function and a fast function with the SFU instruction accuracy.

Another specialized arithmetic operation in a GPU is attribute interpolation. Key *attributes* are usually specified for vertices of primitives that make up a scene to be rendered. Example attributes are color, depth, and texture coordinates. These attributes must be interpolated in the

(x,y) screen space as needed to determine the values of the attributes at each pixel location. The value of a given attribute *U* in an (x, y) plane can be expressed using plane equations of the form:

$$U(x,y) = A_u x + B_u y + C_u$$

where *A*, *B*, and *C* are interpolation parameters associated with each attribute *U*. The interpolation parameters *A*, *B*, and *C* are all represented as single-precision floating-point numbers.

Given the need for both a function evaluator and an attribute interpolator in a pixel shader processor, a single SFU that performs both functions for efficiency can be designed. Both functions use a sum of products operation to interpolate results, and the number of terms to be summed in both functions is very similar.

Texture operations

Texture mapping and filtering is another key set of specialized floating-point arithmetic operations in a GPU. The operations used for texture mapping include:

1. Recieve texture address (s, t) for the current screen pixel (x, y), where s and t are single-precision floating-point numbers.
2. Compute the level of detail to identify the correct texture *MIP-map* level.
3. Compute the trilinear interpolation fraction.
4. Scale texture address (s, t) for the selected MIP-map level.
5. Access memory and retrieve desired texels (texture elements).
6. Perform filtering operation on texels.

MIP-map: A Latin phrase *multum in parvo*, or much in a small space. A MIP-map contains precalculated images of different resolutions, used to increase rendering speed and reduce artifacts.

Texture mapping requires a significant amount of floating-point computation for full-speed operation, much of which is done at 16-bit half precision. As an example, the GeForce 8800 Ultra delivers about 500 GFLOPS of proprietary format floating-point computation for texture mapping instructions, in addition to its conventional IEEE single-precision floating-point instructions. For more details on texture mapping and filtering, see Foley and van Dam [1995].

Performance

The floating-point addition and multiplication arithmetic hardware is fully pipelined, and latency is optimized to balance delay and area. While pipelined, the throughput of the special functions is less than the floating-point addition and multiplication operations. Quarter-speed throughput for the special functions is typical performance in modern GPUs, with one SFU shared by four SP cores. In contrast, CPUs typically have significantly lower throughput for similar functions, such as division and square root, albeit with more accurate results. The attribute interpolation hardware is typically fully pipelined to enable full-speed pixel shaders.

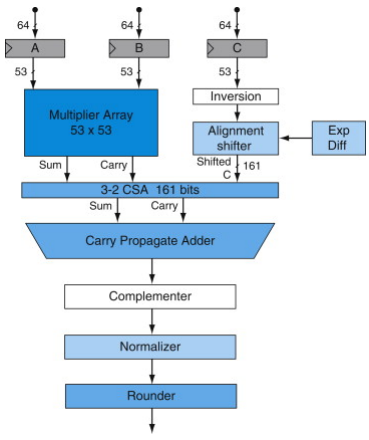
Double precision

Newer GPUs such as the Tesla T10P also support IEEE 754 64-bit double-precision operations in hardware. Standard floating-point arithmetic operations in double precision include addition, multiplication, and conversions between different floating-point and integer formats. The 2008 IEEE 754 floating-point standard includes specification for the *fused-multiply-add* (FMA) operation, as discussed in COD Chapter 3 (Arithmetic for Computers). The FMA operation performs a floating-point multiplication followed by an addition, with a single rounding. The fused multiplication and addition operations retain full accuracy in intermediate calculations. This behavior enables more accurate floating-point computations involving the accumulation of products, including dot products, matrix multiplication, and polynomial evaluation. The FMA instruction also enables efficient software implementations of exactly rounded division and square root, removing the need for a hardware division or square root unit.

A double-precision hardware FMA unit implements 64-bit addition, multiplication, conversions, and the FMA operation itself. The architecture of a double-precision FMA unit enables full-speed denormalized number support on both inputs and outputs. The figure below shows a block diagram of an FMA unit.

Figure 8.6.2: Double-precision fused-multiply-add (FMA) unit (COD Figure B.6.2).

Hardware to implement floating-point A × B + C for double precision.



As shown in the figure above, the significands of A and B are multiplied to form a 106-bit product, with the results left in carry-save form. In parallel, the 53-bit addend C is conditionally inverted and aligned to the 106-bit product. The sum and carry results of the 106-bit product are summed with the aligned addend through a 161-bit-wide *carry-save adder* (CSA). The carry-save output is then summed together in a carry-propagate adder to produce an unrounded result in nonredundant, two's complement form. The result is conditionally recomplemented, so as to return a result in sign-magnitude form. The complemented result is normalized, and then it is rounded to fit within the target format.

(*1) This section is in original form.

Provide feedback on this section