Arm Neon Intrinsics:

The following tutorial will briefly go over what arm neon intrinsics are, how to set them up, and best practices to optimize performance.

What is it?

Neon Intrinsics is an architecture extension available on many ARM processors that a programmer can use to perform single instruction multiple data operations. Rather than a traditional arithmetic instruction, where a CPU performs an operation on a single set of data (two operands and a result), NEON instructions perform operations on a data vectors, sized to 128bits. For applications where the same operation needs to be made on a large set of data (eg matrix math or computer vision), the neon engine can streamline the process, increasing your program's performance.

Basic Functions:

First, ensure you've included the header file in your code:

```
#include <arm_neon.h> //include the neon library
```

Since arm neon has its own set of registers where data is stored and operated on, a programmer must use the arm neon data types. These are usually in the format of: {data_type}x{num}_t where {data_type} is your typical data type (uint16, float32, int16, etc) and {num} is the number stored in the register. So for eight 16bit ints, you'd have: int16x8_t. Note that this data is sized to 64 or 128 bits, so size accordingly. Ideally you operate on as large of a set of data at once.

You declare these data types like in any other program, but to operate on them not using the neon engine you need to convert them into arrays. Same for the other way around. Neon cannot interpret an array of four 32 bit floats normally. This is where the load and store instructions come into play:

```
//load in an array
vld1q_f32(float *values) //load in an array of floats
vld1q_u16(uint16_t *values) //load in an array of uint16_ts
vldq_s16(int16_t *values) //load in an array of int16_ts

//set all lanes to a specific value
vld1q_dup_f32(float *val)
vld1q_dup_u16(uint16_t *val)
vld1q_dup_s16(int16_t *val)
```

```
//set all lanes to a hardcoded value
vmovq_n_f32(1.5f);
vmovq_n_u16(4);
vmovq_n_s16(-4);

//store a vector into an array
vst1q_f32(float *values, vector);
vst1q_u16(uint16_t *values, vector);
vst1q_s16(int16_t *values, vector);
```

To perform arithmetic on your vectors, arm neon provides a variety of functions:

```
vaddq_f32(v1, v2); // sum two float32 vectors
vmulq_f32(v1, v2); // multiply two float32 vectors
vmulq_n_f32(v, s); //multiply vector lanes by a scalar
vmlaq_n_f32(v1, v2, s); //multiply v2 by scalar s and accumulate into v1
```

Takeaways

When designing your program using arm neon intrinsics, be sure to optimize your program to work on as much data as possible without unnecessarily complicating other aspects. By reducing the number of operations, your program will run closer to real time speeds.