Automatic RelaxedPrecision Decoration and Conversion in Spirv-Opt

Greg Fischer, LunarG
September 2019 - Revision 1

Executive Summary

Two new passes have been added to spirv-opt to automatically convert SPIR-V shaders to utilize RelaxedPrecision semantics. These passes allow the developer to take advantage of lower precision computations without having to make changes to the shader source. The first pass can be used to automatically convert a whole SPIR-V shader to use RelaxedPrecision, which can be advantageous on both mobile and desktop GPUs. The second pass can be used to automatically convert a RelaxedPrecision-decorated shader to explicitly use 16-bit precision on devices which do not support RelaxedPrecision, but do support the VK_KHR_shader_float16_int8 extension, such as iOS/MoltenVK.

Introduction

When a 32-bit floating point (float32) SPIR-V executable instruction is decorated with RelaxedPrecision, the result is still a float32 value, but the driver has permission to only compute as few as 16-bits of precision. This can improve performance and/or reduce power consumption if the application does not require the full precision for that instruction. This is a fairly easy change to make as it does not require changes to variable data types.

In GLSL, a programmer can quickly maximize use of RelaxedPrecision by defaulting to medium precision using “precision mediump float;” at the top of a fragment shader. However, given HLSL or SPIR-V shaders this process is more involved. To this end, we have added the `--relax-float-ops` pass which applies the RelaxedPrecision decoration to all float32 executable instructions in a SPIR-V file, including all image references. All interface variables retain their original types, although their loads are relaxed.

Unfortunately, some Vulkan drivers such as MoltenVK ignore the RelaxedPrecision directive, usually because the underlying implementation (in the case of MoltenVK, Metal), does not support the RelaxedPrecision semantics. If it is the situation, however, that the underlying

implementation does support a true 16-bit floating point type (as it is with Metal), benefits of RelaxedPrecision semantics can be achieved by converting RelaxedPrecision instructions to true 16-bit-type instructions using the `VK_KHR_shader_float16_int8` extension.

To this end, the `--convert-relaxed-to-half` pass has been added to `spirv-opt`. This pass translates all arithmetic float32 instructions decorated with RelaxedPrecision to 16-bit floating point (float16 aka half) instructions, adding additional conversion instructions for operands and results where needed.

How to apply RelaxedPrecision to a whole SPIR-V shader

The RelaxedPrecision decoration is applied to executable instructions in SPIR-V by `glslangValidator` when it sees usage of the mediump precision variables in GLSL. Both `glslangValidator` and `dxc` will do so for the min16float type in HLSL.

If you are starting with a SPIR-V file and you wish to apply the RelaxedPrecision decoration to all float32 executable instructions, perform the following:

```
spirv-opt.exe --relax-float-ops -o f.opt.relax.spv f.opt.spv
```

All variables, including interface variables, retain their original types and are not decorated, however, the loads from these variables are decorated.

How to convert RelaxedPrecision to half

The `spirv-opt --convert-relaxed-to-half` pass is most effective if it is performed after “standard” size optimizations are applied to the SPIR-V, specifically optimizations which remove function scoped variable loads and stores and composite inserts and extracts. Both glslangValidator and DXC apply these optimizations by default as part of HLSL compilation. The `-Os` option of glslangValidator will perform these for GLSL shaders. If you are starting with an unoptimized SPIR-V file f.spv, the following recipe would be equivalent:

```
spirv-opt.exe --eliminate-dead-branches --merge-return
--inline-entry-points-exhaustive --eliminate-dead-functions
--scalar-replacement --convert-local-access-chains
--eliminate-local-single-block --eliminate-local-single-store
--simplify-instructions --eliminate-dead-code-aggressive --vector-dce
--eliminate-dead-inserts --eliminate-dead-code-aggressive
--eliminate-dead-branches --merge-blocks
--eliminate-local-multi-store --if-conversion --simplify-instructions
```

```
--eliminate-dead-code-aggressive --vector-dce
--eliminate-dead-inserts --redundancy-elimination
--eliminate-dead-code-aggressive --cfg-cleanup -o f.opt.spv f.spv
```

Once you have an optimized SPIR-V, perform the following:

```
spirv-opt.exe --convert-relaxed-to-half -o f.opt.half.spv f.opt.spv
```

This converts all relaxed, arithmetic float32-type instructions to float16-type instructions, adding float32->float16 and float16->float32 conversion of operands and results where necessary. The RelaxedPrecision decoration is removed from any converted instructions. Instructions which operate on images such as sample instructions are not converted.

Finally, you will ultimately want to perform the following optimization passes to cleanup any unnecessary conversions generated by `--convert-relaxed-to-half`:

```
spirv-opt.exe --simplify-instructions --redundancy-elimination
--eliminate-dead-code-aggressive -o f.opt.half.clean.spv
f.opt.half.spv
```

When should RelaxedPrecision be converted to half?

Conversion of RelaxedPrecision to half should be done when a driver does not support (ignores) the RelaxedPrecision decoration, but **does** support a true float16 type for arithmetic operations. This is true most significantly for **MoltenVK**, however other drivers may silently be ignoring RelaxedPrecision decorations as well. If applying RelaxedPrecision decorations to a shader does not give any performance improvement or power reduction, you may wish to try converting it to half.

What are the potential benefits of converting RelaxedPrecision to half?

Since GPUs and drivers vary greatly in the benefits of half-precision and the cost of conversion computation, it is hard to predict the overall profit. Our one experience gave good results.

When developing this option, we worked with a MoltenVK developer targeting a number of mobile devices. While the older devices saw only about a 1% improvement in frame rate, the newer devices saw around 12% with the iPhone8 seeing a 13.68% improvement. This improvement was seen using --relax-float-ops to relax all float32 instructions, but one

computation had to be un-relaxed by hand as it caused visual artifacts. No other tuning was performed.

Another benefit of conversion to half is that the resulting code will have the same precision across all platforms, aiding in portability.

What are the potential pitfalls of converting RelaxedPrecision to half?

It is possible that converting RelaxedPrecision to half could cause a shader and its application to run slower. In general, the longer the computation sequence, the less additional conversion is needed and the more profitable the conversion. Very short sequences may see a slowdown and may need to be unrelaxed.

When should RelaxedPrecision **not** be converted to half?

It may be disadvantageous to convert RelaxedPrecision to half when a driver **does** support the RelaxedPrecision decoration. The conversion would prevent true RelaxedPrecision behavior, which might be faster.

How to apply RelaxedPrecision to a whole SPIR-V shader

The RelaxedPrecision decoration is applied to executable instructions in SPIR-V by `glslangValidator` when it sees usage of the mediump precision variables in GLSL. Both `glslangValidator` and `dxc` will do so for the min16float type in HLSL.

If you are starting with a SPIR-V file and you wish to apply the RelaxedPrecision decoration to all float32 executable instructions, perform the following:

```
spirv-opt.exe --relax-float-ops -o f.opt.relax.spv f.opt.spv
```

All variables, including interface variables, retain their original types and are not decorated, however, the loads from these variables are decorated.

When should RelaxedPrecision **not** be applied?

Some computations just require 32-bit precision. One possible workaround is to move the computation from the “relaxed” fragment shader into the “non-relaxed” vertex shader, if possible.

Acknowledgements

The author would like to thank Dan Ginsburg at Valve for providing workloads and feedback during the project development.

Document Change Log

9/6/19 - First version