Looking at OpenCL Assembly Language

Mike Bailey
mjb@cs.oregonstate.edu
Oregon State University

How to Extract the OpenCL Assembly Language

This binary can then be used in a call to clCreateProgramWithBinary()
Things Learned from Examining OpenCL Assembly Language

- The points, vectors, and colors were typedef’d as float4’s, but the compiler realized that they were being used as float3’s, and didn’t bother with the 4th element.
- The floats’s were not SIMD’ed. (We actually knew this already, since NVIDIA doesn’t support vector operations in their GPUs.) There is still an advantage in coding this way, even if just for readability.
- The function calls were all in-lined. (This makes sense – the OpenCL spec says “no recursion”, which implies “no stack”, which would make function calls difficult.)
- Defining G, DT, and Sphere1 as constant memory types was a mistake. It got the correct results, but the compiler didn’t take advantage of them being constants. Changing them to type const threw compiler errors because of their global scope. Changing them to const and moving them into the body of the kernel function Particle did result in compiler optimizations.
- The sqrt(x²+y²+z²) assembly code is amazingly involved. I can only hope that there is a good reason. Use fast_sqrt( ), fast_normalize( ), and fast_length( ) when you can.
- The compiler did not do a good job with expressions-in-common. I had really hoped it would figure out that detecting if a point was in a sphere and determining the unitized surface normal at that point were mostly the same operation, but it didn’t.
- There is a 4-argument fused-multiply-add instruction (d = a*b + c, one instruction in hardware). The compiler took great advantage of it.