# Looking at OpenCL Assembly Language

**Mike Bailey**

mjb@cs.oregonstate.edu

**Oregon State University**




## How to Dump OpenCL Assembly Language

```
size_t   size;
status = clGetProgramInfo( Program, CL_PROGRAM_BINARY_SIZES, sizeof(size_t), &size, NULL );
PrintCLError( status, "clGetProgramInfo (1):" );


unsigned char * binary  =  new unsigned char [ size ];
status = clGetProgramInfo( Program, CL_PROGRAM_BINARIES, size, &binary, NULL );
PrintCLError( status, "clGetProgramInfo (2):" );


FILE * fpbin  =  fopen( CL_BINARY_NAME, "wb" );
if( fpbin == NULL )
{
      fprintf( stderr, "Cannot create '%s'\n", CL_BINARY_NAME );
}
else
{
      fwrite( binary, 1, size, fpbin );
      fclose( fpbin );
}
delete [ ]  binary;
```

This binary can then be used in a call to *clCreateProgramWithBinary( )*

## NVIDIA OpenCL Assembly Language Sample

```
ld.global.v4.f32    {%f188, %f189, %f190, %f191}, [%r1];        //  load dPobj[ gid ]
ld.global.v4.f32    {%f156, %f157, %f158, %f159}, [%r2];        //  load   dVel[ gid ]

mov.f32     %f17, 0f3DCCCCCD;                                   // DT

fma.rn.f32   %f248, %f156, %f17, %f188;                         // (p + v*DT).x
fma.rn.f32   %f249, %f157, %f17, %f189;                         // (p + v*DT).y
fma.rn.f32   %f250, %f158, %f17, %f190;                         // (p + v*DT).z

mov.f32     %f18, 0fBD48B43B;                                   // .5 * G.y * DT * DT
mov.f32     %f19, 0f00000000;                                   // 0., for .x and .z

add.f32     %f256, %f248, %f19;                                 // (p + v*DT).x + 0.
add.f32     %f257, %f249, %f18;                                 // (p + v*DT).y + .5 * G.y * DT * DT
add.f32     %f258, %f250, %f19;                                 // (p + v*DT).z + 0.

mov.f32     %f20, 0fBF7AE148;                                   // G.y * DT

add.f32     %f264, %f156, %f19;                                 // v.x + 0.
add.f32     %f265, %f157, %f20;                                 // v.y + G.y * DT
add.f32     %f266, %f158, %f19;                                 // v.z + 0.
```

mjb – March 16, 2015

## Things Learned from Examining OpenCL Assembly Language on an NVIDIA GTX 480

• The points, vectors, and colors were typedef'ed as float4's, but the compiler realized that they were being used as float3's, and didn't bother with the 4th element.

• The float*n*'s were not SIMD'ed. (We actually knew this already, since NVIDIA doesn't supported 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.

• sqrt($x^2+y^2+z^2$) 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 floating-point-multiply-and-add instruction (d = a + b*c ), which the compiler took great advantage of.