Looking at OpenCL Assembly Code

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

How to Extract the OpenCL Assembly Language

```c
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'
", CL_BINARY_NAME );
}
else
{
    fwrite( binary, 1, size, fpbin );
    fclose( fpbin );
}
delete [] binary;
```

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

Particles.cl, I

```c
typedef float4 point;
typedef float4 vector;
typedef float4 color;
typedef float4 sphere;
constant float4 G = (float4) ( 0., -9.8, 0., 0. );
constant float DT = 0.1;
constant sphere Sphere1 = (sphere)( -100., -800., 0., 600. );
```

Particles.cl, II

```c
kernel void Particle( global point * dPobj, global vector * dVel, global color * dCobj )
{
    int gid = get_global_id( 0 ); // particle #
    point p = dPobj[gid];
    vector v = dVel[gid];
    point pp = p + v*DT + .5*DT*DT*G; // p'
    vector vp = v + G*DT; // v'
    dPobj[gid] = pp;
    dVel[gid] = vp;
}
```
vector Bounce( vector in, vector n )
{
    n.w = 0.;
    n = normalize( n );
    vector out = in - 2. * n * dot( in.xyz, n.xyz );
    out.w = 0.;
    return out;
}

vector BounceSphere( point p, vector v, sphere s )
{
    vector n;
    n.xyz = fast_normalize( p.xyz - s.xyz);
    n.w = 0.;
    return Bounce( in, n );
}

/NVIDIA OpenCL Assembly Language Sample/

FMA = "Fused Multiply-Add"

D = A + (B*C);

A "normal" multiply-add compilation would handle this as:

tmp = B*C;
D = A + tmp;

A "fused" multiply-add does it all at once, that is, when the low-order bits of B*C
are ready, they are immediately added into the low-order bits of A at the same
time that the higher-order bits of B*C are being multiplied.

Consider a Base 10 example:  789 + ( 123*456 )

\[
\begin{array}{c}
123 \\
\times \quad 456 \\
\hline
 \quad 615 \\
 \quad 738 \\
+ \quad 789 \\
\hline
\quad 56877 \\
\end{array}
\]

Note: In the lower bits of the result, “Normal” A+(B*C) ≠ “FMA” A+(B*C)