

Structured programming

GLSL (/ HLSL / Cg)

- Compiled by command
 - Fast switching between compiled kernels
- Loading and "calling" as in shader assembly

```
uniform vec4 insideColor;
uniform samplerID outsideColorTable;
uniform float maxIterations;

void main ()
{
    vec2 c = gl_TexCoord[0].xy;
    vec2 z = c;
    gl_FragColor = insideColor;

    for (float i = 0; i < maxIterations; i += 1.0)
    {
        z = vec2(z.x*z.x - z.y*z.y, 2.0*z.x*z.y) + c;
        if (dot(z, z) > 4.0)
        {
            gl_FragColor = textureID(outsideColorTable,
                                    i / maxIterations);
            break;
        }
    }
}
```

Shared memory

OpenCL (/ DirectX compute shaders)

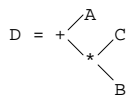
- Shared data within a threadblock
- Explicit synchronization
 - Race conditions
- Thread-driven computation
 - Number of threads determined by programmer
 - Explicit looping within threads

```
__local float4 *shared_pos;
...
int index = get_global_id(0);
int local_id = get_local_id(0);
int tile_size = get_local_size(0);
...
int i, j;
for (i = 0; i < bodies; i += tile_size, tile++)
{
    size_t l_idx = (tile * tile_size + local_id);
    float4 l_pos = l_pos[l_idx];
    shared_pos[local_id] = l_pos;
    barrier(CLK_LOCAL_MEM_FENCE);
    for (j = 0; j < tile_size; j++)
        force = ComputeForce(force, shared_pos[j++],
                             pos, softening_squared);
    barrier(CLK_LOCAL_MEM_FENCE);
}
```

Implicit compilation

- Source is standard C++
 - Single source file
- Kernel is built at run-time through overloading
 - Retained mode: do not execute, but build history of computations

$d = a + b * c$ $a=1, b=2, c=3, d=7$
 $D = A + B * C$ A, B, C objects



Implicit compilation

RapidMind (Sh)

- Macros for unoverloadable operations
- Implicit communication
 - Read & write instead of transfer
 - Asynchronous execution

```
Array<2, ValueIf> A(m,1);
Array<2, ValueIf> B(i,n);
Array<2, ValueIf> C(m,n);

Program mm = BEGIN {
    In<ValueIf> c = ValueIf(0.);
    ValueIf k;

    // Computation of C[i,j]
    RM_FOR (k=0, k < ValueIf(1), k++) {
        c += A[Value2i(ind(0),k)]*B[Value2i(k,ind(1))];
    } RM_ENDFOR;
} END;

C = mm(grid(m,n));
```

Single-level language

CUDA

- Kernel is just a function
 - No variables holding code
- Extension to C/C++
- Requires dedicated compiler

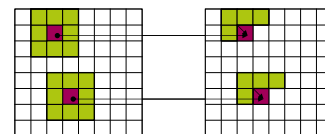
```
__global__ void
paradd(float *in, float *out, int size)
{
    const int stride = blockDim.x * gridDim.x;
    const int start = threadIdx.x * blockDim.x +
        threadIdx.x;
    __shared__ float accum[THREADS];

    accum[threadIdx.x] = 0;
    for (int ii=start; ii < size; ii += stride)
        accum[threadIdx.x] += in[ii];

    __syncthreads();
    if (!threadIdx.x)
    {
        float res = 0;
        for (int ii = 0; ii < blockDim.x; ii++)
            res += accum[ii];
        out[blockIdx.x] = res;
    }
}
```

Stream programming

- Notion of data shape
 - Restricts access pattern
- Can be extended to different access patterns
 - Recursive neighborhood, stack, etc.
 - Dependent on hardware



Stream programming

Brook(GPU)

- Gather streams for random access

```
kernel void lens_correction(float img[1]),
                           iter float2 it<,
                           out float o_img<*,
                           float2 k,
                           float2 mid,
                           float n)
{
    float2 d = abs(it-mid)/n;
    float r2 = dot(d,d);
    float corr = 1.f + r2 * k.x + r2 * r2 * k.y;
    o_img = img[(it-mid) * corr + mid];
}

float img<xsize>, ysize>;
float o_img<xsize, ysize>;
streamRead(img, input);
lens_correction(img, it, o_img, float2(k1, k2),
               float2(xsize/2.f, ysize/2.f),
               n);
streamWrite(o_img, output);
```

Annotation

PGI Accelerator (/ CAPS HMPP)

- Inspired by HPF & OpenMP
- Just add pragmas
 - Can still compile under other compilers
 - Incremental upgrade path
- Compiler is not all-knowing
 - Directives may need to be specific
 - Manually restructure loops

```
typedef float *restrict *restrict MAT;
void
smooth(MAT a, MAT b, float w0, float w1, float w2,
       int n, int m, int niters)
{
    int i, j, iter;
    #pragma acc region
    {
        for( iter = 1; iter < niters; ++iter )
        {
            for( i = 1; i < n-1; ++i )
                for( j = 1; j < m-1; ++j )
                    a[i][j] = w0 * b[i][j] +
                    w1 * (b[i-1][j] + b[i+1][j]) + b[i][j-1] + b[i][j+1] +
                    w2 * (b[i-1][j-1] + b[i-1][j+1] +
                    b[i+1][j-1] + b[i+1][j+1]);
            for( i = 1; i < n-1; ++i )
                for( j = 1; j < m-1; ++j )
                    b[i][j] = a[i][j];
        }
    }
}
```

Accelerator library

Jacket

- All GPU code is encapsulated in library calls
- GPU memory management
- Data conversion = transfer
- Matlab toolbox
- JIT removes overhead
 - Avoid multiple passes
 - Similar to RapidMind retained mode
- Data type determines CPU or GPU execution

```
addpath <jacket_root>/engine
NSET = 1000000;
X = randi( 1, NSET );
Y = randi( 1, NSET );
distance_from_zero = sqrt( X.*X + Y.*Y );
inside_circle = (distance_from_zero <= 1);
pi = 4 * sum(inside_circle) / NSET
pi =
    3.1421
```


Summary

	Struc-tured	Kernels	Lvls	Platform	Compi-lation	Kernel JIT	Comms	Host comms
ASM		✓	2	Library	Explicit	✓	✗	Explicit
GLSL	✓	✓	2	Library	Explicit	✓	✗	Explicit
OpenCL	✓	✓	2	Library	Explicit	✓	Explicit	Explicit
Sh	✓	✓	2	Library	Implicit	✓	Implicit	Implicit
CUDA	✓	✓	1	Compiler	Implicit		Explicit	Explicit
Brook	✓	✓	1	Compiler	Implicit		✗	Implicit
PGI	✓	✓	1	Compiler	Implicit		Implicit	Implicit
Jacket	✓		1	Toolbox	Implicit	✓	Implicit	Implicit

Conclusion

- There are many GPU programming languages
- Some use radically different programming paradigms
 - Often trading efficiency for ease of use
- Paradigm shift often restricted to GPU kernels
 - But future multi-GPU and task parallel code may change that
- Programmer effort will always be required
 - Cannot simply rely on compiler
- Look around before you choose a language

Questions?



Example sources

- Vendors
- <http://cs.anu.edu.au/~Hugh.Fisher/shaders/>
- http://www.ozone3d.net/tutorials/mandelbrot_set_p4.php
- http://developer.apple.com/mac/library/samplecode/OpenCL_NBody_Simulation_Example
- http://www.prace-project.eu/documents/06_rapidmind_vw.pdf