Large Histograms & Non-separable Filters

General Purpose GPU – GLSL/CUDA/OpenCL

Stefan Lindholm
ITN 2010-06-03

Questions:
1) Why is CUDA/OpenCL easier to optimize for small histograms?
2) What allows the actual system speedup to surpass the benchmarked speedup for the Large Histogram implementation?
3) Name one reason CUDA might be a better option than OpenCL for Non-separable Filters?
• Direct Volume Rendering (DVR) for visualization of medical data

• Transfer Function classification of tissues

• Current research
  • Dual-energy Computed Tomography (DECT)
  • Spatial Conditioning (material presence functions)
  • Boundary preserving smoothing filters (anisotropic diffusion)

• Large Histograms

• Non-separable Filters
Large Histograms

- 64bin or 256bin histograms: Calculate and sum multiple small histograms, multiple examples by Nvidia
- Arbitrary size/MD histograms: Local memory overflow (OpenCL/CUDA), incrementations of global memory requires atomic operations

GLSL

- Fragments increase buffers and are atomic (GLSL)
  - "Render" one glPoint per voxel and use vertex shader for bin transformations
  - Requires ~ 134’000’000 GL calls and 134’000’000 texture lookups

- Reduce GL calls
  - Put 134’000’000 glPoints in a Display List or VBO
  - Reduces GL calls but costs 134 MB of VRAM for non-information (0.0.0, 0.0.1, 0.0.2 ...)
  - Geometry shaders? Instancing?

- Reduce texture lookups
  - Interpret the volume as a VBO and get the voxel values directly in gl_Vertex
  - Reduces both GL calls and texture lookups
  - Total call reduction by more than 268’000’000 : 1
  - ~20X benchmarked speedup versus CPU
  - >8000X actual system speedup by in-thread processing
Large Histograms

- Verdict: OpenCL/CUDA implementations not trivial for arbitrary size histograms - GLSL works fine for histograms up to 2D
Non-separable Filters

- **Separable**: Treat data as 1D and filter, keep track of boundary points, multiple examples by Nvidia

- **Non-separable**: Copy MD patch ($N_x\times N_y \times N_z$) to shared mem - problematic since patches need local padding that easily breaks optimal memory layouts (x16)

- `async_work_group_copy`
  - $N_y \times N_z$ calls (possible to minimize)
  - Slow performance even for "ideal" cases

- **Manual copy**
  - 1 copy per thread, works fairly well

- **Local padding**
  - Easily introduces costly offsets to optimal mem layout (x16)
  - 26 additional border conditions (6 sides 12 edges 8 corners)
  - Up to 8 copies (1 self 3 sides 3 edges 1 corner) for a single thread

- **Global padding**
  - Non pow2 images need padding in at least two dims (mem layout)
  - `row_pitch` / `slice_pitch` requires 2D/3D image types in OpenCL (works with buffers in CUDA)
  - Image types (no buffer indexing, vector returns, etc.) or convert/copy
  - Pad on the cpu (can be costly)
event_t e_copy;
for (int i = 0; i < patch_height; i++)
{
    e_copy = async_work_group_copy(t_data+i*patch_width, i_ptr+offset+i*W, patch_width, (i==0)?(event_t)0:e_copy);
}
wait_group_events(1, &e_copy);

//////////////////////////////////////////////////////////////////////////
// Center point

for (int u = 0; u < patch_height; u++)
{
    t_data[tdx] = i_ptr[gdx];
    for (int v = 0; v < patch_height; v++)
    {
        t_data[vdx] = i_ptr[gdx];
        for (int w = 0; w < patch_height; w++)
        {
            if (u == 0 && v == 0 && w == 0) t_data[0x0] = i_ptr[0x0];
            if (u == 0 && v == 0 && w == 1) t_data[0x1] = i_ptr[0x1];
            if (u == 0 && v == 1 && w == 0) t_data[0x2] = i_ptr[0x2];
            if (u == 0 && v == 1 && w == 1) t_data[0x3] = i_ptr[0x3];
            if (u == 1 && v == 0 && w == 0) t_data[0x4] = i_ptr[0x4];
            if (u == 1 && v == 0 && w == 1) t_data[0x5] = i_ptr[0x5];
            if (u == 1 && v == 1 && w == 0) t_data[0x6] = i_ptr[0x6];
            if (u == 1 && v == 1 && w == 1) t_data[0x7] = i_ptr[0x7];
        }
    }
}

Verdict: OpenCL is fast but easily becomes hardcoded or non-intuitive.
- Future
  - More pipeline steps in CUDA/OpenCL (converted from GLSL)
  - Reduction processes / Filter kernels
  - Static / dynamic memory
  - GLSL vs. CUDA/OpenCL

- GPGPU in medical visualization
  - Medical workstations tend to have fancy GPUs
  - Size of datasets is an issue
  - OpenCL / CUDA are very very optimized

- Large Histograms
- Non-separable Filters
Linköping University
expanding reality

www.liu.se