GPU Acceleration in ITK v4

ITK v4 Fall meeting
Nov 8th 2010
Won-Ki Jeong
Overview

• GPU as a fast co-processor
  – Massively parallel
  – Huge speed up for certain types of problem
  – Physically independent system
**FIGURE 46.9**
Running time comparison of anisotropic diffusion filter.

<table>
<thead>
<tr>
<th># iter</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>0.225</td>
<td>0.308</td>
<td>0.479</td>
<td>0.82</td>
</tr>
<tr>
<td>1 CPU</td>
<td>14.9</td>
<td>29.7</td>
<td>59.3</td>
<td>122</td>
</tr>
<tr>
<td>2 CPU</td>
<td>9.78</td>
<td>18.6</td>
<td>38.2</td>
<td>76.7</td>
</tr>
<tr>
<td>3 CPU</td>
<td>7.89</td>
<td>17</td>
<td>32.2</td>
<td>64.2</td>
</tr>
<tr>
<td>4 CPU</td>
<td>7.13</td>
<td>13.7</td>
<td>27.7</td>
<td>58.2</td>
</tr>
<tr>
<td>5 CPU</td>
<td>6.3</td>
<td>12.5</td>
<td>25.2</td>
<td>48.2</td>
</tr>
<tr>
<td>6 CPU</td>
<td>5.76</td>
<td>10.4</td>
<td>22.4</td>
<td>43.9</td>
</tr>
<tr>
<td>7 CPU</td>
<td>4.89</td>
<td>9.86</td>
<td>20.1</td>
<td>40.2</td>
</tr>
<tr>
<td>8 CPU</td>
<td>4.08</td>
<td>9.49</td>
<td>19.2</td>
<td>36.9</td>
</tr>
<tr>
<td>Speedup</td>
<td>18x</td>
<td>30.8x</td>
<td>40x</td>
<td>45x</td>
</tr>
</tbody>
</table>

**FIGURE 46.11**
Running time comparison of Median diffusion filter.

<table>
<thead>
<tr>
<th>Kernel size</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>0.279</td>
<td>0.929</td>
<td>2.24</td>
<td>4.42</td>
</tr>
<tr>
<td>1 CPU</td>
<td>6.55</td>
<td>14.4</td>
<td>42.3</td>
<td>99.5</td>
</tr>
<tr>
<td>2 CPU</td>
<td>4.03</td>
<td>11.2</td>
<td>21.4</td>
<td>48.4</td>
</tr>
<tr>
<td>3 CPU</td>
<td>3.14</td>
<td>7.35</td>
<td>16.5</td>
<td>34.4</td>
</tr>
<tr>
<td>4 CPU</td>
<td>2.24</td>
<td>5.88</td>
<td>12.8</td>
<td>28.1</td>
</tr>
<tr>
<td>5 CPU</td>
<td>6.3</td>
<td>5.91</td>
<td>11</td>
<td>23.6</td>
</tr>
<tr>
<td>6 CPU</td>
<td>5.16</td>
<td>4.94</td>
<td>9.04</td>
<td>20.3</td>
</tr>
<tr>
<td>7 CPU</td>
<td>4.72</td>
<td>4.65</td>
<td>8.32</td>
<td>18</td>
</tr>
<tr>
<td>8 CPU</td>
<td>4.46</td>
<td>3.99</td>
<td>7.92</td>
<td>16.6</td>
</tr>
<tr>
<td>Speedup</td>
<td>8x</td>
<td>4.2x</td>
<td>3.5x</td>
<td>3.7x</td>
</tr>
</tbody>
</table>
Overview

• GPU as a fast co-processor
  – Massively parallel
  – Huge speed up for certain types of problem
  – Physically independent system

• Problems
  – Memory management
  – Process management
  – Implementation
Proposal

• Provide GPU image data structure and framework
  – Developer only need to implement GPU kernel
  – ITK will do dirty jobs

• GPU image filter framework
  – GPU filter template
  – Pipelining support

• OpenCL
  – Industry standard
Plan

• GPU data structure
  – How to store data on the GPU

• GPU process framework
  – How to run GPU code in ITK

• Basic GPU operators
  – How to provide abstraction for GPU code
Plan

• GPU data structure
  – How to store data on the GPU

• GPU process framework
  – How to run GPU code in ITK

• Basic GPU operators
  – How to provide abstraction for GPU code
GPU Image Class

- Extension to current ITK image class
  - Two snapshots (CPU/GPU) of the image

```cpp
itk::image::GPUimage

itk::image
GetPixel()
SetPixel()
Iterators...

Synchronize()

CPU Memory
GetBufferpointer()

GPU Memory
GetGPUBufferpointer()
```
Synchronization

- **Implicit**
  - Use flag for checking up-to-date
    - *ex*) `SetPixel()` makes GPU memory outdated
    - *ex*) `GetGPUBufferPointer()` makes CPU memory outdated
  - Transparent to users

- **Deferred synchronization**
  - Synchronize only when needed
  - Minimize copy through PCIe bus
Pipelining

• Read/write

  Read from Disk → GPU Filter → GPU Filter → GPU Filter → GPU Filter → Write to Disk

  Synchronize() Synchronize() Synchronize()

• Combine CPU/GPU filters

  GPU Filter → GPU Filter → CPU Filter → CPU Filter → GPU Filter

  Synchronize() Synchronize()
GPU Image Types

• Buffer type
  - 1D array
  - Read/write by GPU kernel
  - Computation

• Image type
  - 1/2/3D array
  - Texture, framebuffer
  - Hardware interpolation
GPU Image Pixel Type

• Byte, integer, float
• 1/2/3/4 channels
• No template support in OpenCL
  – Pixel type must be predefined
  – Need to write GPU kernel for each pixel type
Plan

• GPU data structure
  – How to store data on the GPU

• GPU process framework
  – How to run GPU code in ITK

• Basic GPU operators
  – How to provide abstraction for GPU code
GPU Code Integration

• Wrapper for existing GPU languages
  – Pros: easier to implement as libraries
  – Cons: limit to a specific GPU language
  – ex) thrust, PyCUDA

• DSL compiled to target GPU architectures
  – Pros: abstraction independent from the GPU
  – Cons: need to write compiler, lots of work, introduce another layer of complication
  – ex) brook GPU
Design Ideas

• Things to keep in mind...
  – GPU kernel as a functor per pixel
  – CPU-GPU memory copy is expensive
  – SIMD/STMD architecture

• GPU computing model
  – Copy and keep large chunk of data to the GPU
  – Apply same operation to each pixel
  – Image filter!
GPU Process Object

• New base class for GPU image filter
  – Input is GPU image
  – Provide wrapper APIs for GPU
    • Kernel code
    • Arguments setup
    • Kernel launcher

• Developer only need to write per-pixel GPU kernel code
ITK Multithreading for GPU

• Extension to current ITK multithreading model
  – GPUGenerateData(): Single GPU
  – GPUThreadedGenerateData(): Multi-GPU

• ITK will handle
  – Data split and merge
  – Per-thread OpenCL context management
  – Per-GPU resource management
Example (pseudo code)

// GPU kernel source code: c = a + b
__kernel kernelcode(float* _a, float* _b, float* _c)
{
    const int idx = get_global_id(0);
    c[idx] = a[idx] + b[idx];
}

// itk GPU filter implementation
itk::GPUImageAddFilter(..)
{
    GPUImage _a, _b, _c;

    void GPUGenerateData(..)
    {
        SetKernelArg(0, _a);
        SetKernelArg(1, _b);
        SetKernelArg(2, _c);
        LaunchKernel();
    }
}
Plan

• GPU data structure
  – How to store data on the GPU

• GPU process framework
  – How to run GPU code in ITK

• Basic GPU operators
  – How to provide abstraction for GPU code
Abstractions

• Low-level: inside GPU kernel (OpenCL code)
  – Index computation
  – Neighborhood iterator

• High-level: common operators (image filters)
  – +, -, *, /
  – Assignment, copy
  – Reduction, inner product, prefix sum, etc
Low-level: Index Computation

• Typical index calculation in OpenCL code

```c
// 2D neighbor index
__kernel laplacian(float* _in, float*_out)
{
    const int width = get_global_size(0);
    const int xidx = get_global_id(0);
    const int yidx = get_global_id(1);

    int ct = yidx*width + xidx;
    int lf = yidx*width + xidx - 1;
    int rt = yidx*width + xidx + 1;
    int up = (yidx+1)*width + xidx;
    int dn = (yidx-1)*width + xidx;

    _out[center] = 0.25f*(_in[up]+_in[dn]+_in[lf]+_in[rt])
                   -_in[ct];
}
```
Low-level: Index Computation

- Using index function

```c
// 2D neighbor index
__kernel laplacian(float* _in, float * _out)
{
    int ct = get_center_index_2d();
    int lf = get_left_index_2d();
    int rt = get_right_index_2d();
    int up = get_up_index_2d();
    int dn = get_down_index_2d();

    _out[center] = 0.25f*(_in[up]+_in[dn]+_in[lf]+_in[rt]) - _in[ct];
}
```
Low-level: Neighborhood Iterator

- Convolution using for-loop

```c
//
// Convolution kernel size: (2*xwidth+1)*(2*ywidth+1)
//
__kernel convolution(float* _in, float * _out, float * _k,
    int xwidth, int ywidth)
{
    int kernelIdx = 0;
    const int width = get_global_size(0);
    int center = get_center_index_2d();
    for(int offY = -ywidth; offY <= ywidth; offY++)
    {
        for(int offX = -xwidth; offX <= xwidth; offX++)
        {
            int idx = offY*width + offY;
            _out[center] += _k[kernelIdx]*_in[idx];
            kernelIdx++;
        }
    }
}
```
Low-level: Neighborhood Iterator

- Convolution using offset index

```c
//
// Convolution kernel size: (2*xwidth+1)*(2*ywidth+1)
//
__kernel convolution(float* _in, float * _out, float * _k, int xwidth, int ywidth)
{
    int kernelIdx = 0;
    int center = get_center_index_2d();
    for(int offY = -ywidth; offY <= ywidth; offY++)
    {
        for(int offX = -xwidth; offX <= xwidth; offX++)
        {
            int idx = get_offset_index_2d(offX, offY);
            _out[center] += _k[kernelIdx]*_in[idx];
            kernelIdx++;
        }
    }
}
```
Low-level: Neighborhood Iterator

• Using neighbor iterator

```c
__kernel convolution(float* _in, float* _out, float* _k,
                     int xwidth, int ywidth)
{
    int kernelIdx = 0;
    const int width = get_global_size(0);
    int center = get_center_index_2d();
    neighbor_iterator itr = iterator_begin(xwidth, ywidth, center);
    while(itr.idx >= 0)
    {
        _out[center] += _k[kernelIdx] * _in[itr.idx];
        kernelIdx++;
        itr = iterator_next(itr, xwidth, ywidth);
    }
}
```
High-level: Image Operators

• Commonly used image operators
  – Addition, subtraction, division, multiplication
  – Reduction, inner product, prefix sum
  – Copy, assignment
  – Neighborhood image filter (convolution)

• Combine them to implement algorithms
  – ex) Conjugate-Gradient solver
Discussion

• Current OpenCL has some limitations
  – No compiler support for C++
  – Sub-optimal performance (5~10%)

• Should we support NVIDIA CUDA?
  – C++, template, function pointer
  – CUBLAS, CUFFT, NPP, Thrust

• Hardware-specific optimization
  – ATI, NVIDIA, Intel
Discussion

• Which ITK filters should be re-implemented for GPU?
  – Registration
  – Image filters
  – Level-set segmentation

• Hybrid implementation
  – Split jobs for CPU & GPU