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

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

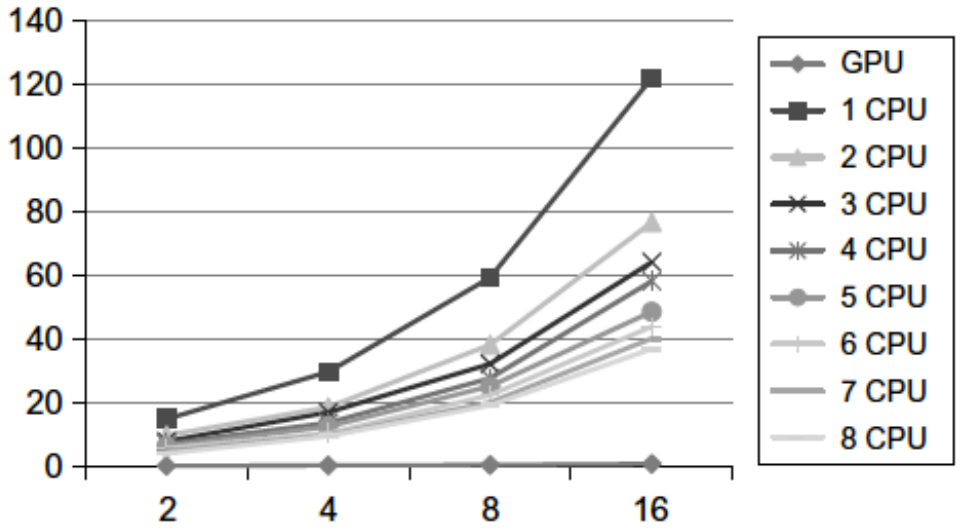


FIGURE 46.9

Running time comparison of anisotropic diffusion filter.

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

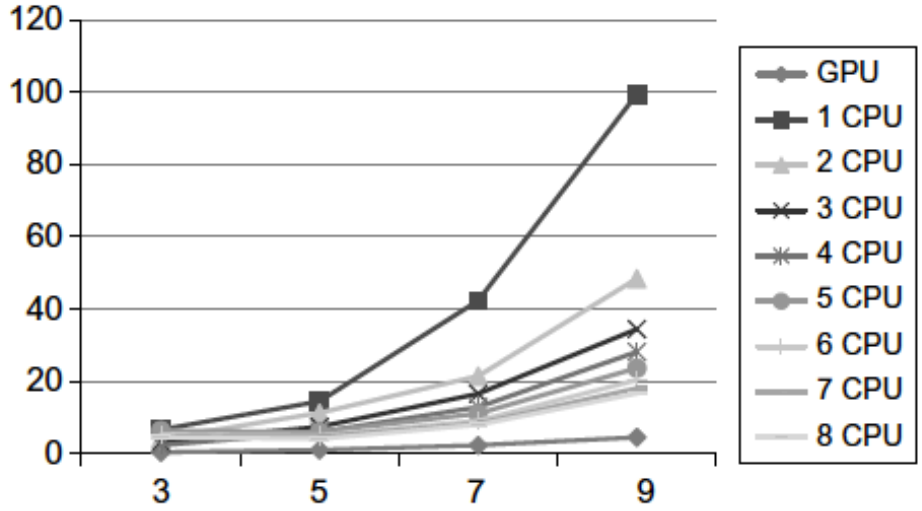


FIGURE 46.11

Running time comparison of Median diffusion filter.

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

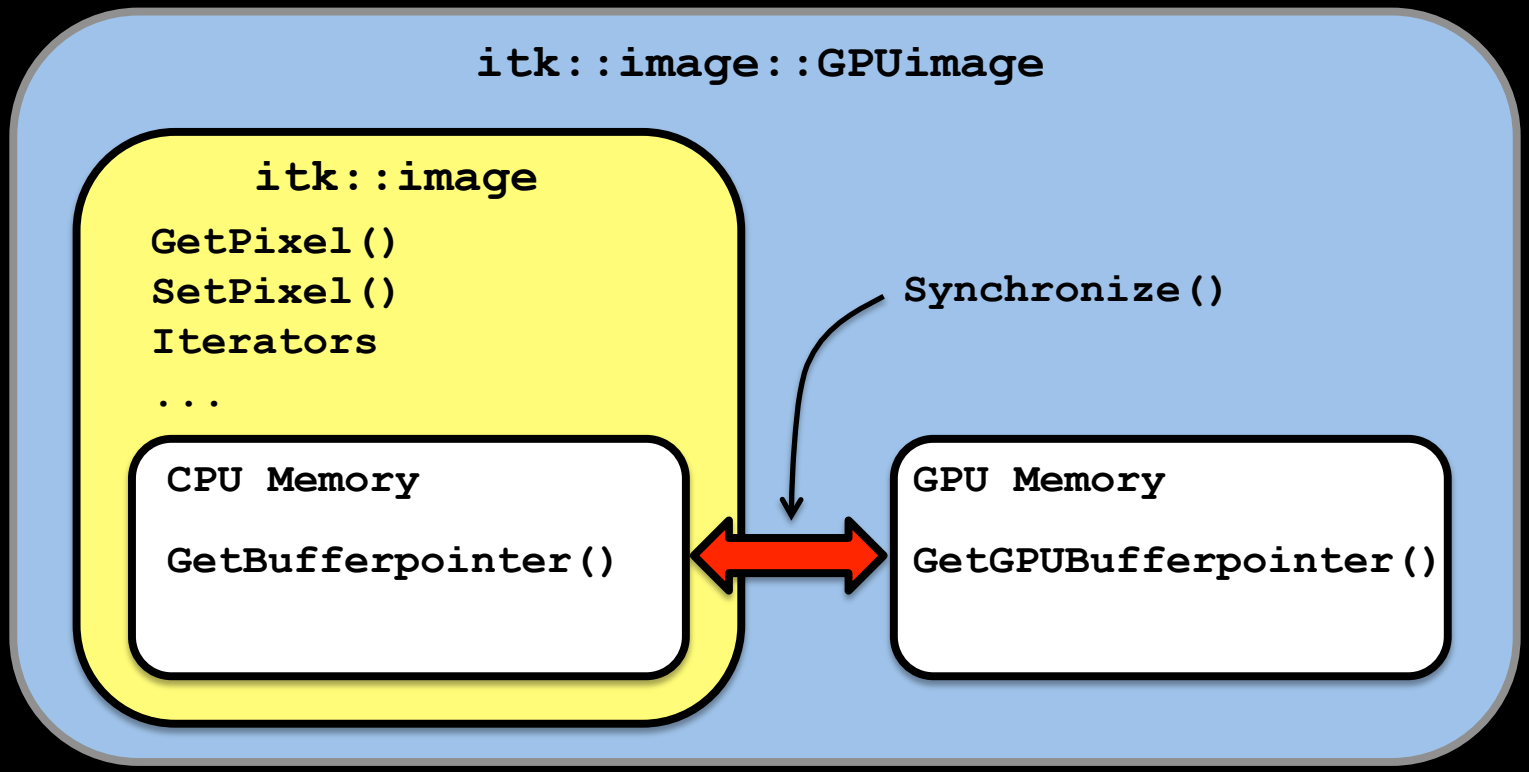
- GPU data structure
  - How to store data on the GPU
- GPU process framework
  - How to run GPU code in ITK
- Basic GPU operators
  - How to provide abstraction for GPU code

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

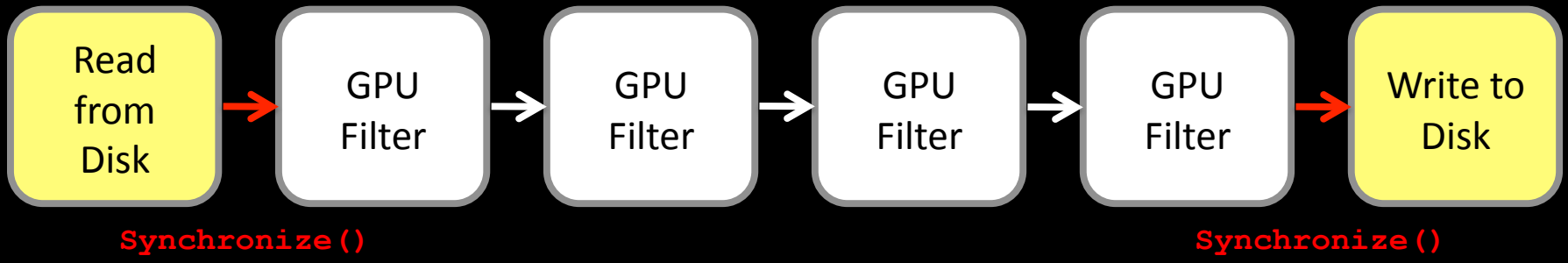


Synchronization

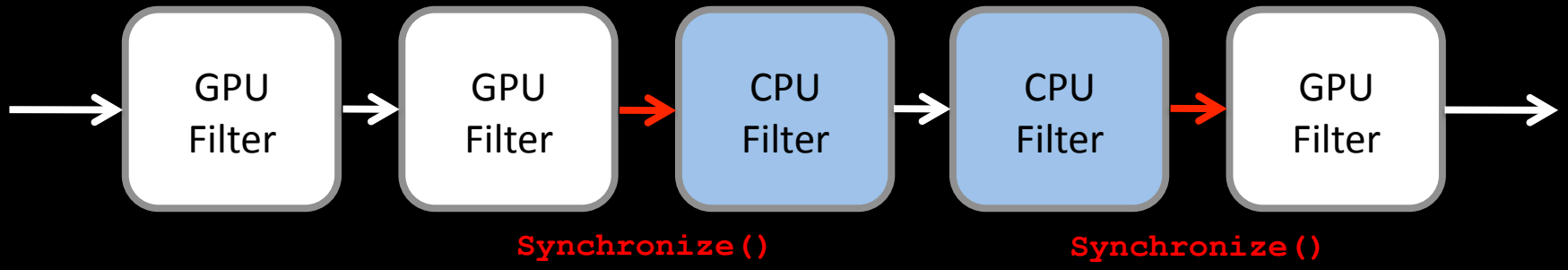
- Implicit
  - Use flag for checking up-to-date
    - ex) `SetPixel()` makes GPU memory outdated
    - ex) `GetGPUBufferPointer()` makes CPU memory outdated
  - Transparent to users
- Deferred synchronization
  - Synchronize only when needed
  - Minimize copy through PCIe bus

Pipelining

- Read/write



- Combine CPU/GPU filters



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 GPUGerateData(..)
    {
       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

```
// 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

```
// 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

```
//
// 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

```
//
// 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

```
//
// 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();
    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