Thrust++: Portable, Abstract Library for Medical Imaging Applications
Parallel Computing – Challenges and Solutions

What is Thrust++?
Thrust++ pipeline pattern
Demo
Thrust++ Data structures and Algorithms
Future Work
Parallel Computing
Real-time performance and scalability to Siemens products

Healthcare
- Advanced Coronary Analysis
  - Accelerated algorithms
  - Rich and interactive advanced visualization
  - Real-time analytics

Industry
- SINUMERIK CNC
  - Real-time embedded control
  - High precision simulations
  - Rich and interactive operator interfaces

Energy
- Fault Localization of Turbine blades
  - Detailed 3D simulations
  - Interactive CFD
  - Real-time algorithms for automation and control

I&C
- Security Solutions
  - Real-time image and video processing
  - High performance solutions for crowd simulations

Virtual Design of Steam Turbines
- Portable Diagnosis & Screening
  - Signal Conversion
- Various

Danger Management (Evacuation)
Thrust++ is based on Thrust\textsuperscript{1}

Thrust: Features

- Open source C++ template library for parallel platforms
- Abstraction
- Portability
- Widely used parallel abstraction library
- Boosts Productivity

Thrust Users

Agenda

Parallel Computing – Challenges and Solutions

What is Thrust++?

Thrust++ pipeline pattern

Demo

Thrust++ Data structures and Algorithms

Future Work
Extensions to Thrust

Thrust: Limitations

- Limited Data Structures
  - Lack of Multi-Dimensional Data Structures
  - Lack of Complex Data Structures
- Limited Algorithms
  - Generic for 1Dimensional data
- No Patterns

Extended Collection of Algorithms
FFT-1d, FIR, Convolution,…

Parallel Patterns
Pipeline pattern

Data-structures
Device Texture 1D/2D
Host Texture 1D/2D
Device Array 2D/3D
Host Array 2D/3D
Agenda

Parallel Computing – Challenges and Solutions

What is Thrust++?

**Thrust++ pipeline pattern**

Demo

Thrust++ Data structures and Algorithms

Future Work
Pipeline

- Example illustrates how a pipeline can be used to speed up computation. Pipelines occur in various domains

**Example of an Image Processing Pipeline**

- Stage 1: Load
- Stage 2: Scale
- Stage 3: Filter
- Stage 4: Display

Source: Parallel Programming with Microsoft .NET
Thrust++ Pipeline pattern

- Thrust++ pipeline pattern can be used to develop portable parallel stream based applications. It is an extension to FluenC²

- Key features
  - Abstraction from underlying hardware
  - Linear pipelines with multiple sources/sinks
  - Serial (in-order) and parallel (out-of-order) stages
  - Generic programming in STL style
  - Supports different back ends such as CUDA, TBB, OpenMP, and CPP

- Current version, pipeline pattern uses the following external schedulers
  - Sequential scheduler
  - Parallel scheduler (TBB)

Steps to set up the Thrust++ Pipeline

- **Include Header file**
  - `#include <thrust/patterns/pipeline/pipeline.hpp>`

- **Construct a network**
  - A network consists of a set of stages that are connected by communication channels.

- **Create Stages**
  - Inheriting from parent classes
    - **Source**: This type of stage has only output port.
    - **Serial**: This type of stage has input and output port. Only one instance of this stage can run at a time
    - **Parallel**: This type of stage has input and output port. Multiple instances of this type of stage can run in parallel. In general, processes that neither have any side effects nor maintain a state can safely be executed in parallel.
    - **Sink**: This type of stage has only input ports

- **Connect the stages**
  - E.g. `stage1.connect<0>(stage2.port<0>())`

- **Start the network**
  - E.g. `nw(10)`
Pipeline: CUDA

Requirements:
• CUDA operations must be in different, non-0, streams
• cudaMemcpyAsync with host from 'pinned' memory
• Sufficient resources must be available
• A blocked operation blocks all other operations in the queue, even in other streams

Stream Scheduling
• A CUDA operation is dispatched from the engine queue if:
  • Preceding calls in the same stream have completed,
  • Preceding calls in the same queue have been dispatched, and
  • Resources are available

Thrust++ Extensions
• Asynchronous Copies
• Pinned Allocation
• Streams support
Streams scheduling

Sequential Scheduler: Results into depth first

Compute queue

<table>
<thead>
<tr>
<th>Issue Order</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1-I1</td>
<td>S1-I2</td>
</tr>
<tr>
<td>S2-I1</td>
<td>S2-I2</td>
</tr>
<tr>
<td>S3-I1</td>
<td>S3-I2</td>
</tr>
<tr>
<td>S1-I2</td>
<td>S1-I3</td>
</tr>
<tr>
<td>S2-I2</td>
<td>S2-I3</td>
</tr>
<tr>
<td>S3-I2</td>
<td>S3-I3</td>
</tr>
</tbody>
</table>

Execution

<table>
<thead>
<tr>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Stage 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1-I1</td>
<td>S2-I1</td>
<td>S3-I1</td>
</tr>
<tr>
<td>S2-I1</td>
<td>S3-I1</td>
<td>S1-I2</td>
</tr>
<tr>
<td>S3-I1</td>
<td>S1-I2</td>
<td>S2-I2</td>
</tr>
<tr>
<td>S1-I2</td>
<td>S3-I2</td>
<td>S1-I3</td>
</tr>
<tr>
<td>S2-I2</td>
<td>S1-I3</td>
<td>S2-I3</td>
</tr>
<tr>
<td>S3-I2</td>
<td>S2-I3</td>
<td>S3-I3</td>
</tr>
</tbody>
</table>

Streams

- stream1
- stream2
- stream3

Streams Schedule:

- Default
- Stream 8
- Stream 9
- Stream 10
- Stream 11
- Stream 12
- Stream 13
- Stream 14
- Stream 15
- Stream 16
- Stream 17
- Stream 18
- Stream 19
Streams scheduling

TBB Scheduler: In ideal situation will result into breadth first

![Diagram of streams scheduling]

- Compute queue
  - Issue Order
    - S1-I1
    - S1-I2
    - S1-I3
    - S2-I1
    - S2-I2
    - S2-I3
    - S3-I1
    - S3-I2
    - S3-I3

- Time
  - Execution
    - S1-I1
    - S1-I2
    - S1-I3
    - S2-I1
    - S2-I2
    - S2-I3
    - S3-I1
    - S3-I2
    - S3-I3

- Streams
  - Default
  - Stream 8
  - Stream 9
  - Stream 10
  - Stream 11
  - Stream 12
  - Stream 13
  - Stream 14
  - Stream 15
  - Stream 16
### Event Matrix

<table>
<thead>
<tr>
<th>Pre Stage</th>
<th>Post Stage</th>
<th>Pre Stage</th>
<th>Post Stage</th>
<th>Pre Stage</th>
<th>Post Stage</th>
<th>Pre Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stage 0: Source</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Image-0</td>
<td>Image-1</td>
<td>Image-2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\times$</td>
<td>$W([0][0])$</td>
<td>$W([1][0])$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stage 1: Serial</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\times$</td>
<td>$R([0][0])$</td>
<td>$R([1][0])$</td>
<td>$R([2][0])$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$W([0][1])$</td>
<td>$R([1][1])$</td>
<td>$R([2][1])$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stage 2: Parallel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\times$</td>
<td>$R([0][2])$</td>
<td>$R([1][2])$</td>
<td>$R([2][2])$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$W([0][3])$</td>
<td>$R([1][3])$</td>
<td>$R([2][3])$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stage 3: Sink</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\times$</td>
<td>$R([0][3])$</td>
<td>$R([1][3])$</td>
<td>$R([2][3])$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Create event matrix of dimension $[\text{Max}_\text{Streams}][\text{Max}_\text{Stages}]$

- **Guarantees in-order execution**
  - $R$ (Post-stage) `cudaEventRecord`
  - $W$ (Pre-Stage) `cudaStreamWaitEvent`

Note: Parallel Stage has no wait event as it does not demand in-order execution.

```c
// Pre-Stage
if(image_id != 0 && (stage_kind != PARALLEL))
    cudaStreamWaitEvent(stream_id,kernelEvent[image_id-1][stageid],0);

// Post Stage
    cudaEventRecord(kernelEvent[image_id][stageid], stream_id);
```
Agenda

Parallel Computing – Challenges and Solutions

What is Thrust++?

Thrust++ pipeline pattern

Demo

Thrust++ Data structures and Algorithms

Future Work
Cone beam computed tomography (X-ray beam is a cone beam.)

The CBCT scanner rotates around the patient at different projection angles:
- Typically at a resolution of 1 or 0.5 degrees.
- A full scan of 360 degrees resulting in a total of 720 projections with resolution of 0.5 degrees.

A reconstruction algorithm is used to reconstruct the 3D image of interest from the projections.

A popular algorithm for cone beam computed tomography reconstruction is the Feldkamp algorithm.
Summary and Findings: CT Reconstruction

• Patterns
  • Texture usage should be avoided inside pipeline (*CUDA7.0 RC removes this limitation*)
  • Device to device copy is launched by default in zero stream. This breaks the pipeline
  • Parallel scheduler provides better overlap between stages
  • Complete C++11 support not available till CUDA 5.5 for linux

<table>
<thead>
<tr>
<th>Data</th>
<th>CUDA Memory Type</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Texture Memory</td>
<td>Takes advantage of hardware linear interpolation</td>
</tr>
<tr>
<td>Pre-computed</td>
<td>Shared Memory/Constant memory</td>
<td>Few pre-computed values repeatedly accessed by multiple threads.</td>
</tr>
<tr>
<td>values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>Global memory</td>
<td>N<em>N</em>N float values</td>
</tr>
</tbody>
</table>
Summary and Findings: CT Reconstruction

- **Hardware backend**
  - Nvidia K20c using CUDA 5.5

- **What could be the other reasons for the performance degradation with PARADigm?**
  - Thread configuration

Performance comparison of different implementations of back projection kernel

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Thread configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Back-projection</td>
<td>CUDA Native implementation</td>
</tr>
</tbody>
</table>
Summary and Findings: CT Reconstruction

Performance comparison of different implementations of back projection kernels. The PARADIGM implementation with launch configuration (512,512) has minimal performance degradation with that of the Native CUDA implementation.
Agenda

Parallel Computing – Challenges and Solutions

What is Thrust++?

Thrust++ pipeline pattern

Demo

Thrust++ Data structures and Algorithms

Future Work
Thrust++

Data Structures

- **Two Dimensional**
  - device/host_array2d
    - 2 Dimensional Data structure for host/device
  - device/host_image2d
    - Read Only data structure for 2D spatial locality
    - Supported interpolation
    - Supported only on Kepler +, CUDA 5.0 onwards (Bindless texture)

- **Three Dimensional**
  - device/host_array3d
    - 3 Dimensional Data structure for host/device

Texture Data Structure

```c
struct print_values {
  __host__ __device__
  float operator()(float val)
  {
    printf("\n %f", val);
    return val;
  }
};

thrust::device_vector<float> vec1(4);
thrust::sequence(vec1.begin(), vec1.end());

thrust::device_texture2d<float> tex(vec1, 2, 2);
thrust::for_each(tex.texture_begin().texture_end(), print_values());
```

- `cudaCreateTextureObject()`
- `cudaMemcpyToArray()`
- `tex1Dfetch()`
Agenda

Parallel Computing – Challenges and Solutions

What is Thrust++?

Thrust++ pipeline pattern

Demo

Thrust++ Data structure and Algorithms

Future Work
Future Work

• Non linear pipeline support for CUDA

• OpenCL backend
  • Support embedded space: Mali GPU
  • VexCL: https://github.com/ddemidov/vexcl
  • Boost.Compute: http://kylelutz.github.io/compute/

• Shared memory/Local Memory
  • Bulk library approach (http://on-demand.gputechconf.com/gtc/2014/presentations/S4673-thrust-parallel-algorithms-bulk.pdf)

• Other calculators
  • Occupancy is not always the best measure to get optimal performance
  • Two phase decomposition/Launchbox (http://nvlabs.github.io/moderngpu/intro.html)

Parallel software Abstractions for Rapid Adaptation, Deployment and Integration over Multicore/Manycore architectures
Contact

Thank you. For further information you can contact Parallel Systems India Lab:

**Giri Prabhakar**
Head of Research Group
Parallel Systems India Lab

E-mail: 
giri.prab@siemens.com

**Bharatkumar Sharma**
Lead Research Engineer
Parallel Systems India Lab

E-mail: 
bharatkumar.sharma@siemens.com
Backup
BackProjection

- Backprojection is the most compute intensive step in 3D image reconstruction amounting for 97% of execution time.
- For N*N*N volume, N*N threads are launched.
- A loop runs depth wise which captures contribution of each input convoluted image to the output volume as shown in Figure.
- The loop is unrolled to increase ILP to get better performance.

<table>
<thead>
<tr>
<th>Data</th>
<th>CUDA Memory Type</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Texture Memory</td>
<td>Takes advantage of hardware linear interpolation</td>
</tr>
<tr>
<td>Pre-computed values</td>
<td>Shared Memory/Constant memory</td>
<td>Few pre-computed values repeatedly accessed by multiple threads.</td>
</tr>
<tr>
<td>Output</td>
<td>Global memory</td>
<td>N<em>N</em>N float values</td>
</tr>
</tbody>
</table>
Software Stack

Parallel Algorithms
Parallel Patterns
Data Structures and Iterators
Power Monitor

Device Back-end Extensions

CUDA
OpenMP
TBB
...

Operating System & Device Drivers

Hardware (multi-core CPU + many-core device (GPU, MIC, …))

[1] Thrust, an open-source C++ library based on C++ STL will be extended to provide the Many-core Software Abstraction Layer
[2] Tools and utilities to monitor and optimize power (energy) usage