SIEMENS

Siemens Corporate Technology | March, 2015

# Thrust++: Portable, Abstract Library for Medical Imaging Applications

© Siemens AG 2015. All rights reserved

# Agenda

**Parallel Computing – Challenges and Solutions**

What is Thrust$^{++}$?

Thrust$^{++}$ pipeline pattern

Demo

Thrust$^{++}$ Data structures and Algorithms

Future Work

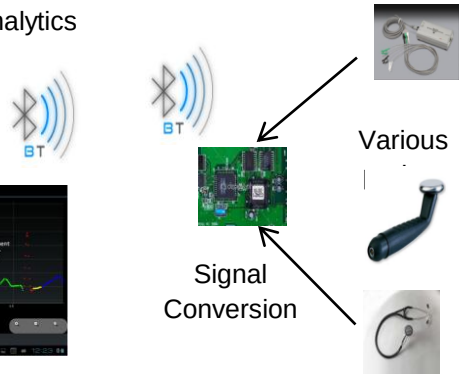

© Siemens AG 2015. All rights reserved

# Parallel Computing
## Real-time performance and scalability to Siemens products

### Healthcare

**Advanced Coronary Analysis**

- Accelerated algorithms
- Rich and interactive advanced visualization
- Real-time analytics




**Portable Diagnosis & Screening**

### Industry

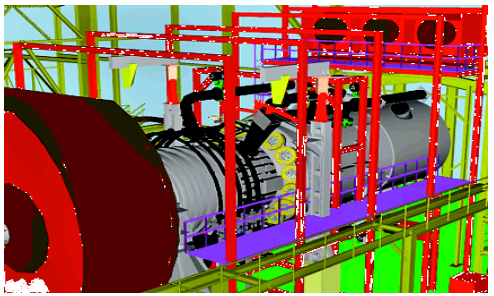
**SINUMERIK CNC**

- 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



**Virtual Design of Steam Turbines**

### I&C


**Security Solutions**

- Real-time image and video processing
- High performance solutions for crowd simulations

**Danger Management (Evacuation)**


© Siemens AG 2015. All rights reserved

# Thrust$^{++}$ is based on Thrust[1]

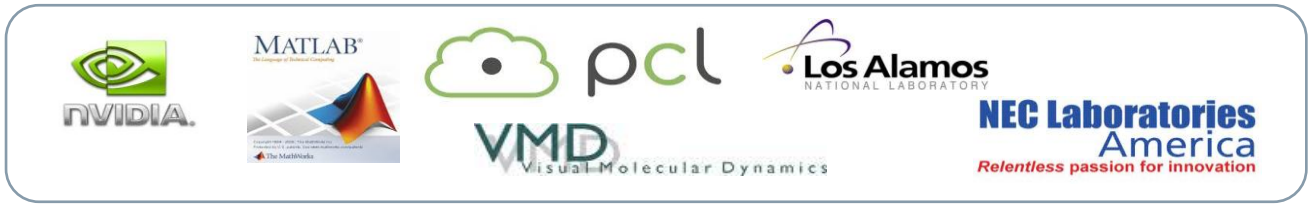
## Thrust: Features

**Open source C++ template library for parallel platforms**

**Abstraction** + **Portability** = **Boosts Productivity**

**Widely used parallel abstraction library**

**Thrust Users**




[1] Thrust: http://thrust.github.io/


© Siemens AG 2015. All rights reserved

# Agenda

Parallel Computing – Challenges and Solutions

**What is Thrust$^{++}$?**

Thrust$^{++}$ pipeline pattern

Demo

Thrust$^{++}$ Data structures and Algorithms

Future Work


© Siemens AG 2015. All rights reserved

# 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


© Siemens AG 2015. All rights reserved

SIEMENS

# Agenda

Parallel Computing – Challenges and Solutions

What is Thrust++?

**Thrust++ pipeline pattern**

Demo

Thrust++ Data structures and Algorithms

Future Work


© Siemens AG 2015. All rights reserved

# Pipeline

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

**Example of an Image Processing Pipeline**


Source: Parallel Programming with Microsoft .NET


© Siemens AG 2015. All rights reserved

# Thrust$^{++}$ Pipeline pattern

- Thrust$^{++}$ pipeline pattern can be used to develop portable parallel stream based applications. It is an extension to FluenC[2]

- 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

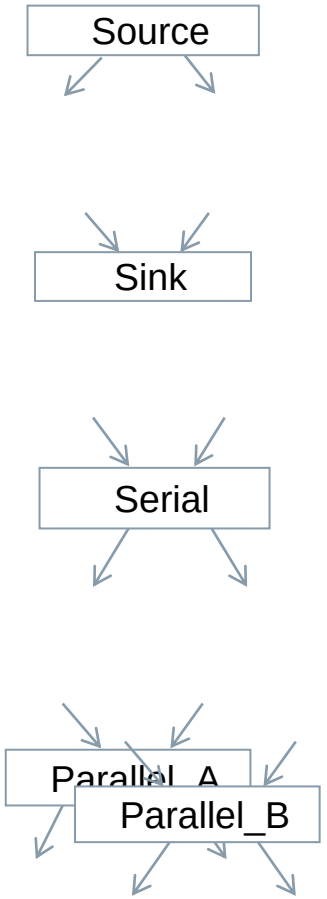
- Current version, pipeline pattern uses the following external schedulers
  - Sequential scheduler
  - Parallel scheduler (TBB)

[2]T. Schuele, “Efficient parallel execution of streaming applications on multi-core processors,” in Proceedings of the 19th International Euromicro Conference on Parallel, Distributed and Network-based Processing, PDP2011, Ayia Napa, Cyprus, 9-11 February 2011, 2011


© Siemens AG 2015. All rights reserved

# 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)





© Siemens AG 2015. All rights reserved

SIEMENS

# 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

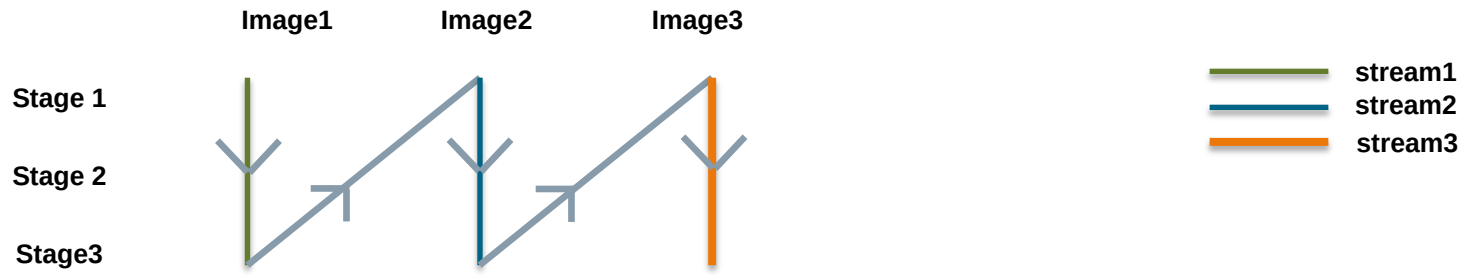
**Thrust++ Extensions**

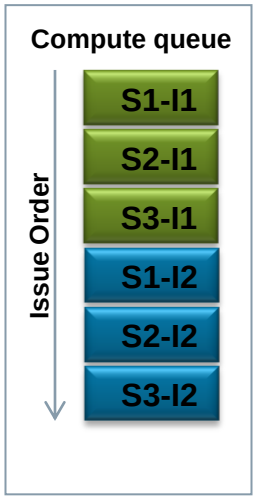
- Asynchronous Copies
- Pinned Allocation
- Streams support

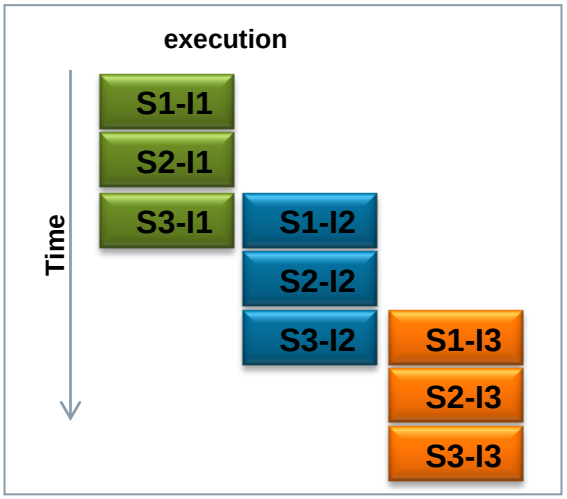

© Siemens AG 2015. All rights reserved

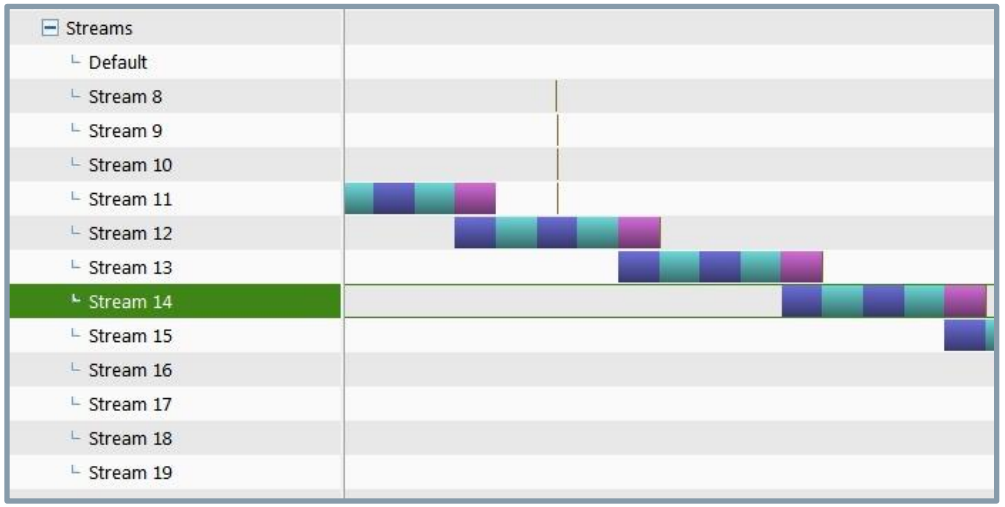
# Streams scheduling

## Sequential Scheduler: Results into depth first



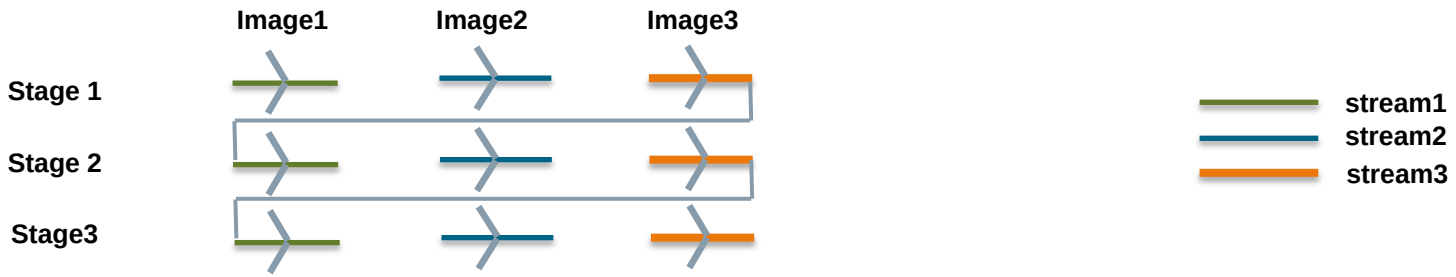





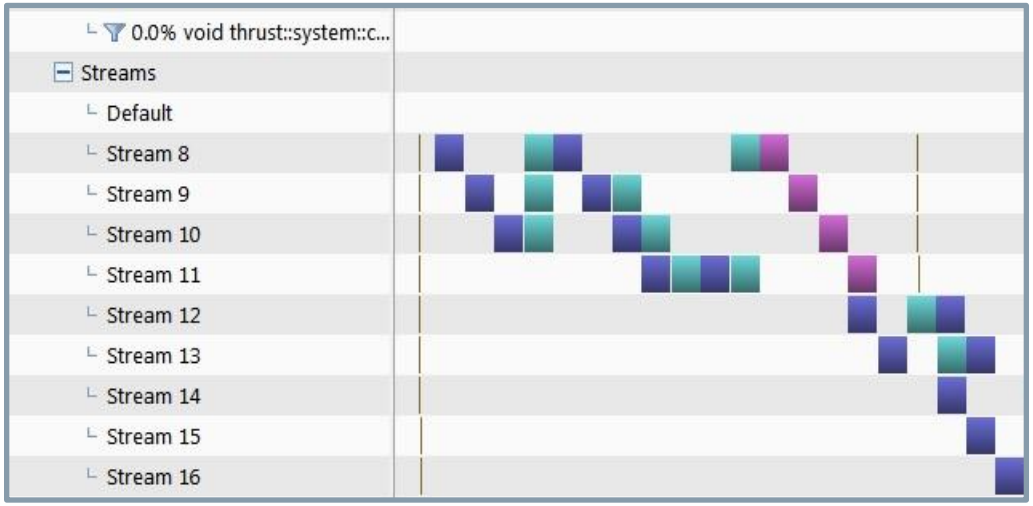
© Siemens AG 2015. All rights reserved

# Streams scheduling

**TBB Scheduler: In ideal situation will result into breadth first**









© Siemens AG 2015. All rights reserved

# Event Matrix

| | Image-0 | Image-1 | Image-2 |
|---|---|---|---|
| Pre Stage | | W([0][0]) | W([1][0]) |
| **Stage 0: Source** | | | |
| Post Stage | R([0][0]) | R([1][0]) | R([2][0]) |
| Pre Stage | | W([0][1]) | W([1][1]) |
| **Stage 1: Serial** | | | |
| Post Stage | R([0][1]) | R([1][1]) | R([2][1]) |
| Pre Stage | | | |
| **Stage 2: Parallel** | | | |
| Post Stage | R([0][2]) | R([1][2]) | R([2][2]) |
| Pre Stage | | W([0][3]) | W([1][3]) |
| **Stage 3: Sink** | | | |
| Post Stage | R([0][3]) | R([1][3]) | R([2][3]) |

**Create event matrix of dimension [Max_Streams][Max_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**

```
//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);
```


© Siemens AG 2015. All rights reserved

# Agenda

Parallel Computing – Challenges and Solutions

What is Thrust$^{++}$?

Thrust$^{++}$ pipeline pattern

**Demo**

Thrust$^{++}$ Data structures and Algorithms

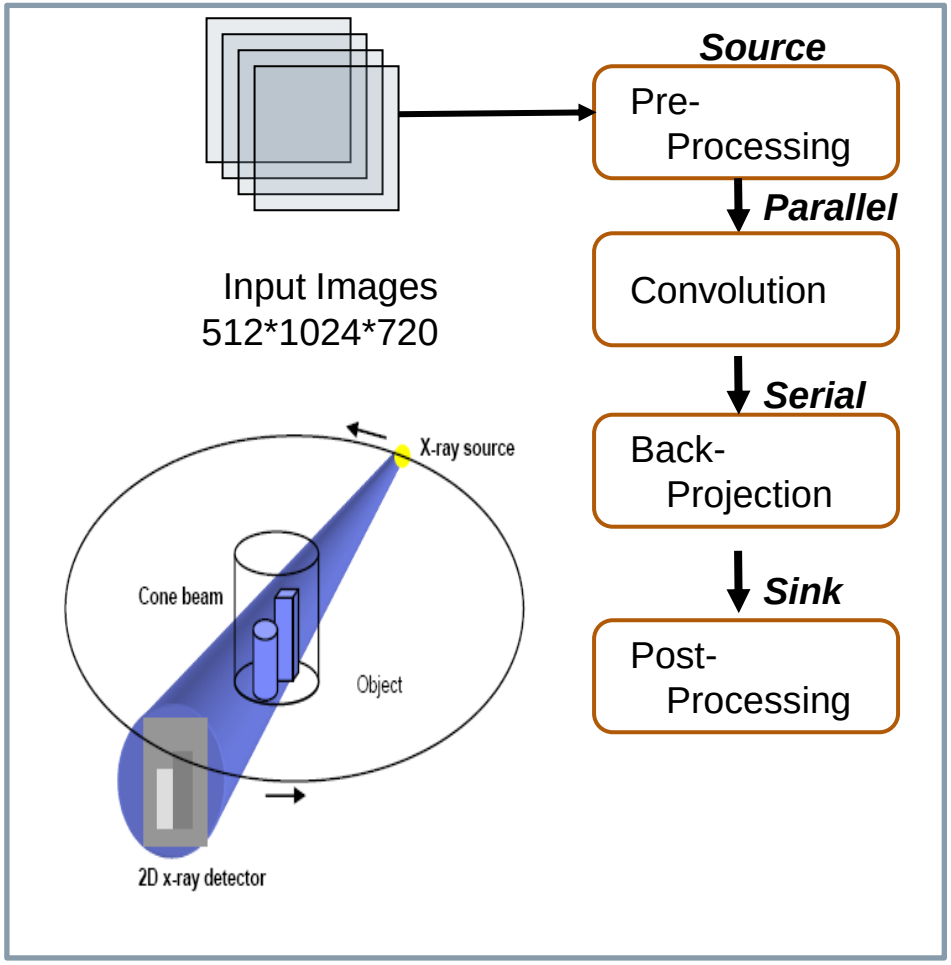
Future Work


© Siemens AG 2015. All rights reserved

# Demo Application to illustrate Pipeline pattern

## CT Reconstruction

- 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.





© Siemens AG 2015. All rights reserved

# Summary and Findings: CT Reconstruction

•Patterns

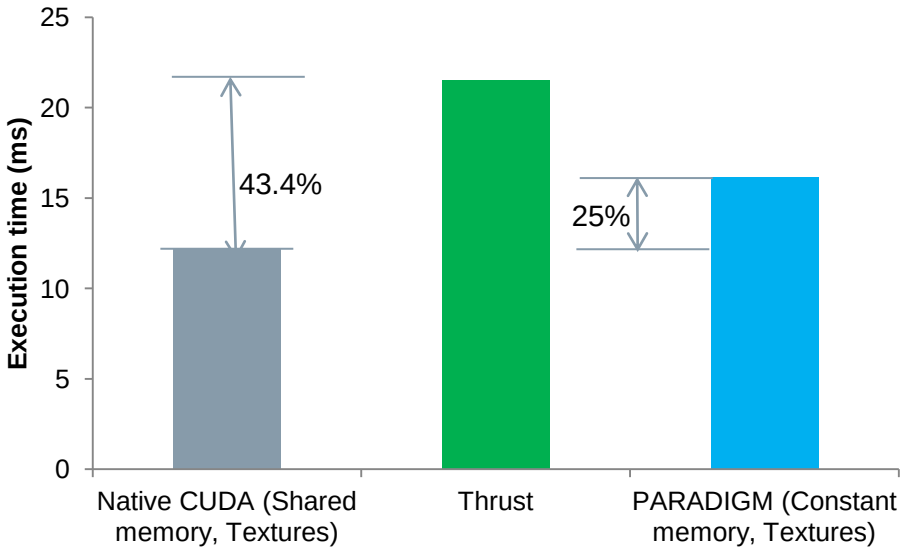
- Texture usage should be avoided inside pipeline (*CUDA7.0 RC removes this limitation*)
  - Device to device copy is launched by default in zero stream. This breaks the pipeline
- Parallel scheduler provides better overlap between stages
- Complete C++11 support not available till CUDA 5.5 for linux

| Data | CUDA Memory Type | Reason |
|---|---|---|
| Input | Texture Memory | Takes advantage of hardware linear interpolation |
| Pre-computed values | Shared Memory/Constant memory | Few pre-computed values repeatedly accessed by multiple threads. |
| Output | Global memory | N*N*N float values |


© Siemens AG 2015. All rights reserved

# Summary and Findings: CT Reconstruction

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



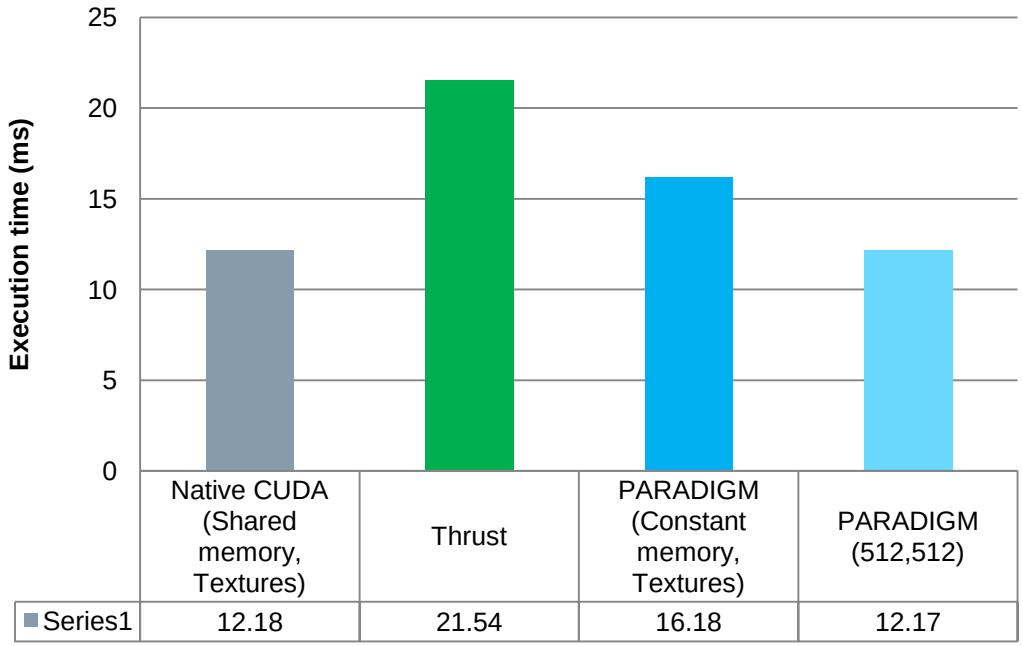

Performance comparison of different implementations of back projection kernel

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

| Kernel | Thread configuration | |
|---|---|---|
| | CUDA Native implementation | PARADIGM implementation |
| Back-projection | Block: [128 * 4]<br>Grid: [4 * 128] | Block: [768*1]<br>Grid: [26*1] |


© Siemens AG 2015. All rights reserved

# 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


© Siemens AG 2015. All rights reserved

# Agenda

Parallel Computing – Challenges and Solutions

What is Thrust$^{++}$?

Thrust$^{++}$ pipeline pattern

Demo

**Thrust$^{++}$ Data structures and Algorithms**

Future Work


© Siemens AG 2015. All rights reserved

# 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
    - Supports 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

```
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(), tex.
    texture_end(),print_values());
```

cudaCreateTextureObject ()

cudaMemcpyToArray()

tex1Dfetch()


© Siemens AG 2015. All rights reserved

# Agenda

Parallel Computing – Challenges and Solutions

What is Thrust$^{++?}$

Thrust$^{++}$ pipeline pattern

Demo

Thrust$^{++}$ Data structure and Algorithms

**Future Work**


© Siemens AG 2015. All rights reserved

# Future Work

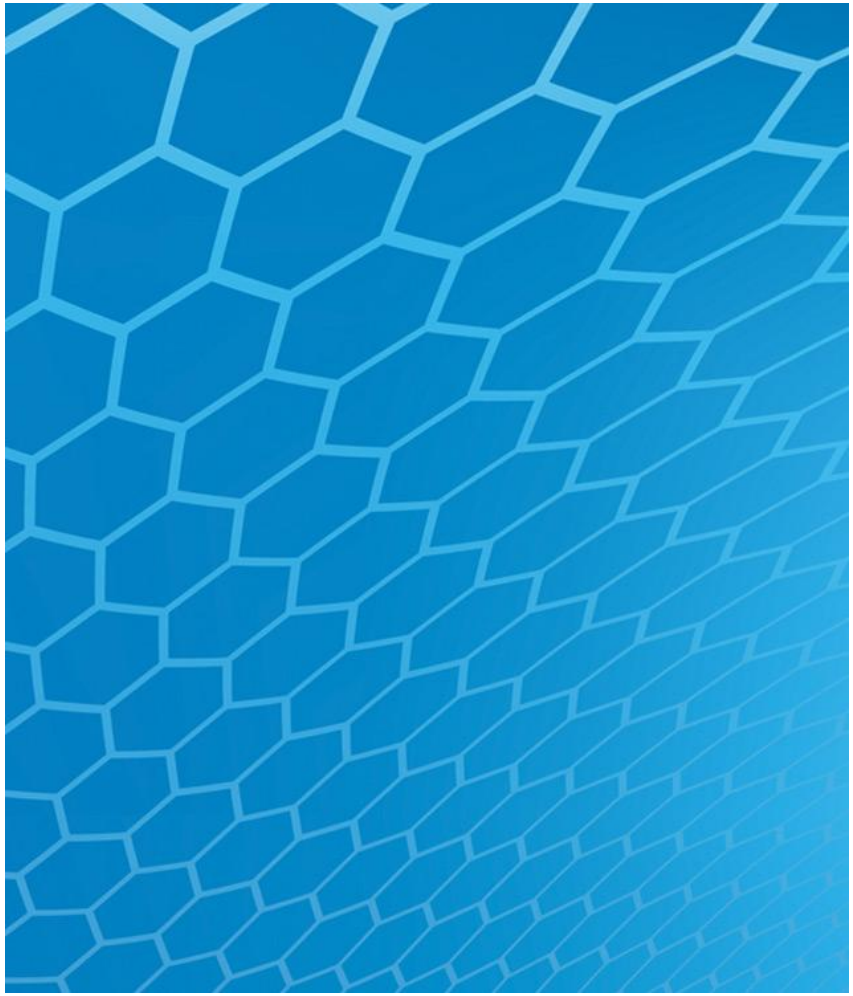
- Non linear pipeline support for CUDA
- OpenCL backend
  - Support embedded space: Mali GPU
  - VexCL: https://github.com/ddemidov/vexcl
  - Boost.Compute: http://kylelutz.github.io/compute/
- Shared memory/Local Memory
  - Bulk library approach (http://on-demand.gputechconf.com/gtc/2014/presentations/S4673-thrust-parallel-algorithms-bulk.pdf)
- Other calculators
  - Occupancy is not always the best measure to get optimal performance
    - Two phase decomposition/Launchbox ( http://nvlabs.github.io/moderngpu/intro.html )


**P**arallel software **A**bstractions for **R**apid **A**daptation, **D**eployment and **I**nte**G**ration over Multicore/Manycore architectures


© Siemens AG 2015. All rights reserved

# 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

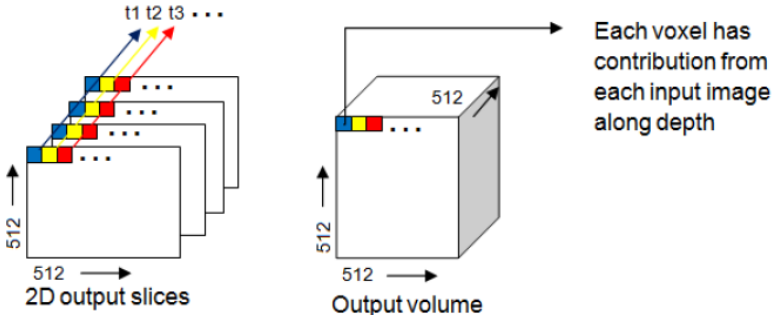

© Siemens AG 2015. All rights reserved

SIEMENS

# Backup

© Siemens AG 2015. All rights reserved

# BackProjection




| Data | CUDA Memory Type | Reason |
|---|---|---|
| Input | Texture Memory | Takes advantage of hardware linear interpolation |
| Pre-computed values | Shared Memory/Constant memory | Few pre-computed values repeatedly accessed by multiple threads. |
| Output | Global memory | N*N*N float values |

•Backprojection is the most compute intensive step in 3D image reconstruction amounting for 97% of execution time

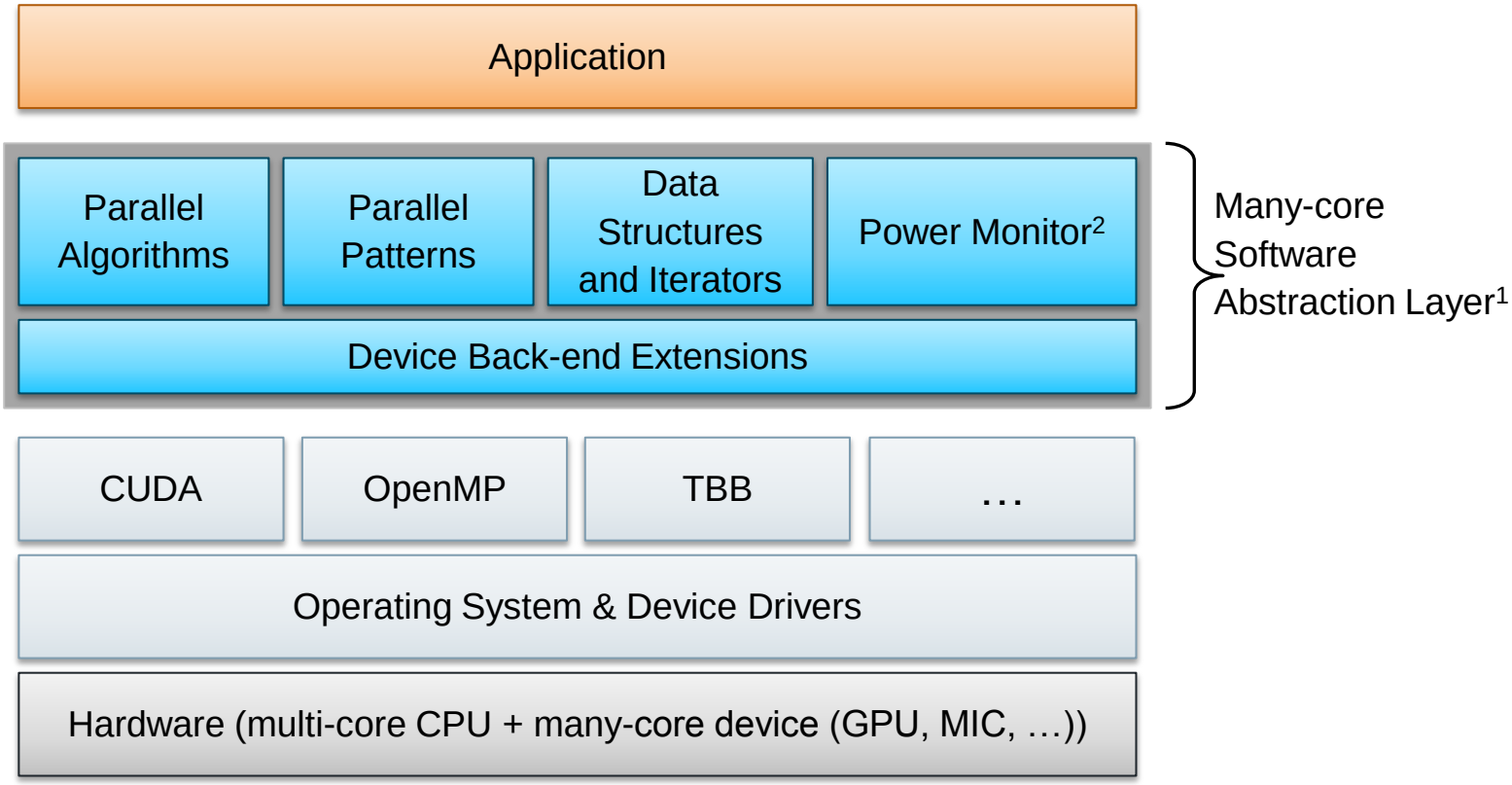
•For N*N*N volume, N*N threads are launched.

•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.


© Siemens AG 2015. All rights reserved

# Software Stack




[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


© Siemens AG 2015. All rights reserved