Radically Simplified GPU Parallelization: The Alea Dataflow Programming Model

Luc Bläser
Institute for Software, HSR Rapperswil

Daniel Egloff
QuantAlea, Zurich

Funded by Swiss Commission of Technology and Innovation,
Project No 16130.2 PFES-ES

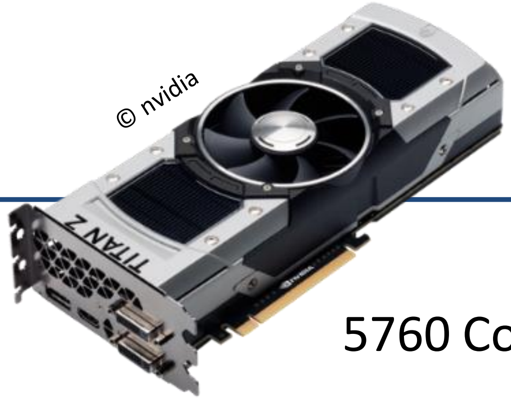
HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
FHO Fachhochschule Ostschweiz

INSTITUT FÜR SOFTWARE

QuantAlea

GTC 2015
20 Mar 2015

GPU Parallelization Requires Effort



5760 Cores

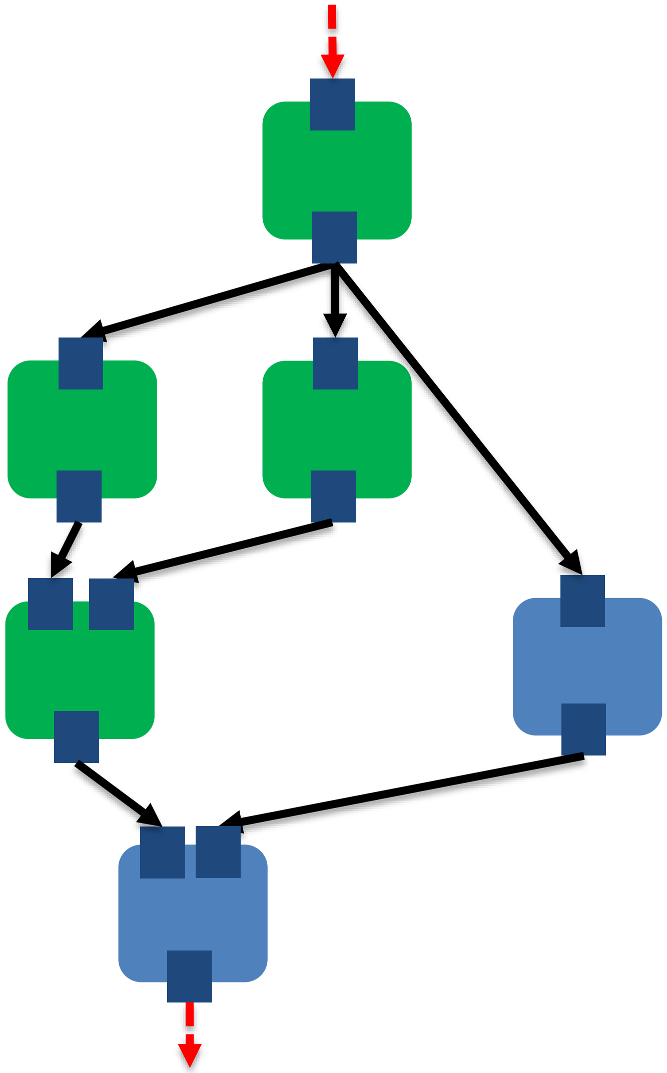
- Massive Parallel Power
  - □ Thousands of cores
  - □ But specific pattern: vector-parallelism
- High obstacles
  - □ Particular algorithms needed
  - □ Machine-centric programming models
  - □ Limited language and runtime integration
- Good excuses against it - unfortunately
  - □ Too difficult, costly, error-prone, marginal benefit

Our Goal: A Radical Simplification

- GPU parallel programming for (almost) everyone
  - □ No GPU experience required
  - □ Fast development
  - □ Good performance
- On the basis of .NET
  - □ Available for C#, F#, VB etc.

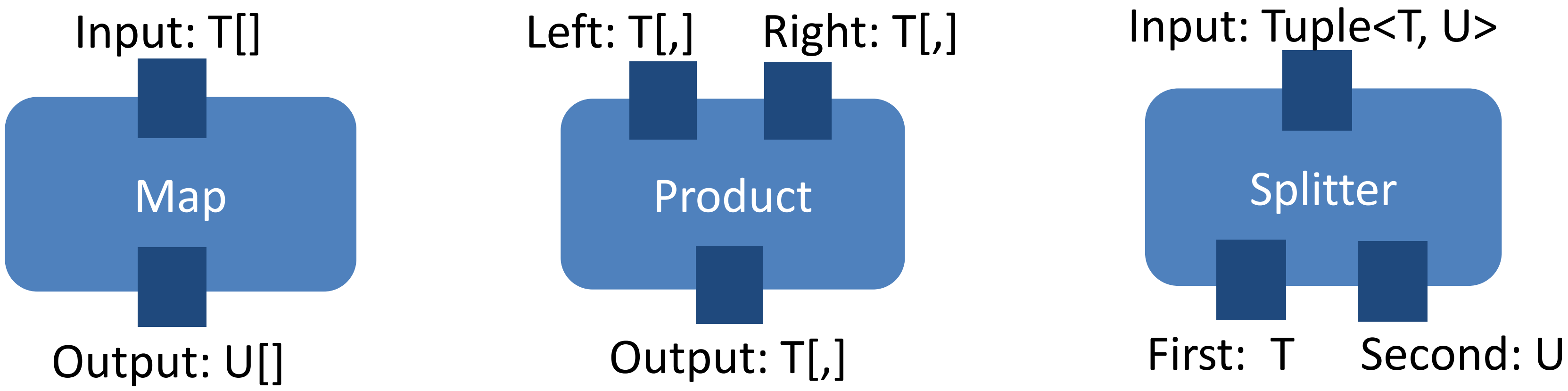
Alea Dataflow Programming Model

- Dataflow
  - □ Graph of operations
  - □ Data propagated through graph
- Reactive
  - □ Feed input in arbitrary intervals
  - □ Listen for asynchronous output

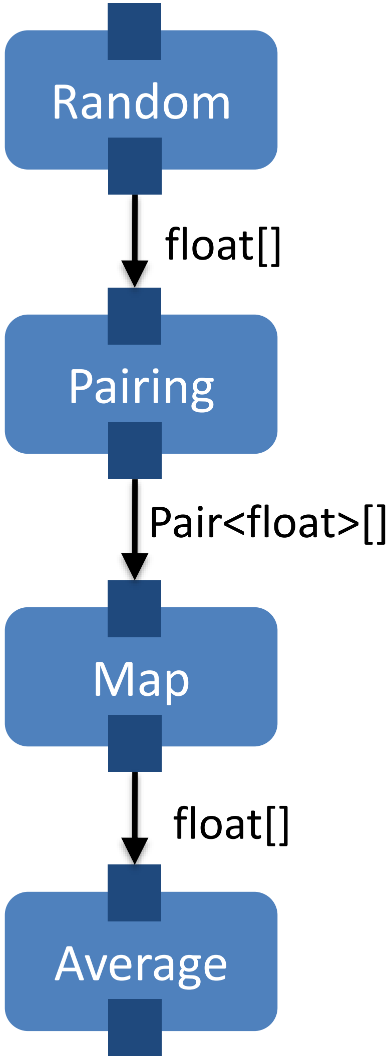


Operation

- Unit of (vector-parallel) calculation
- Input and output ports
- Port = stream of typed data
- Consumes input, produces output



Graph

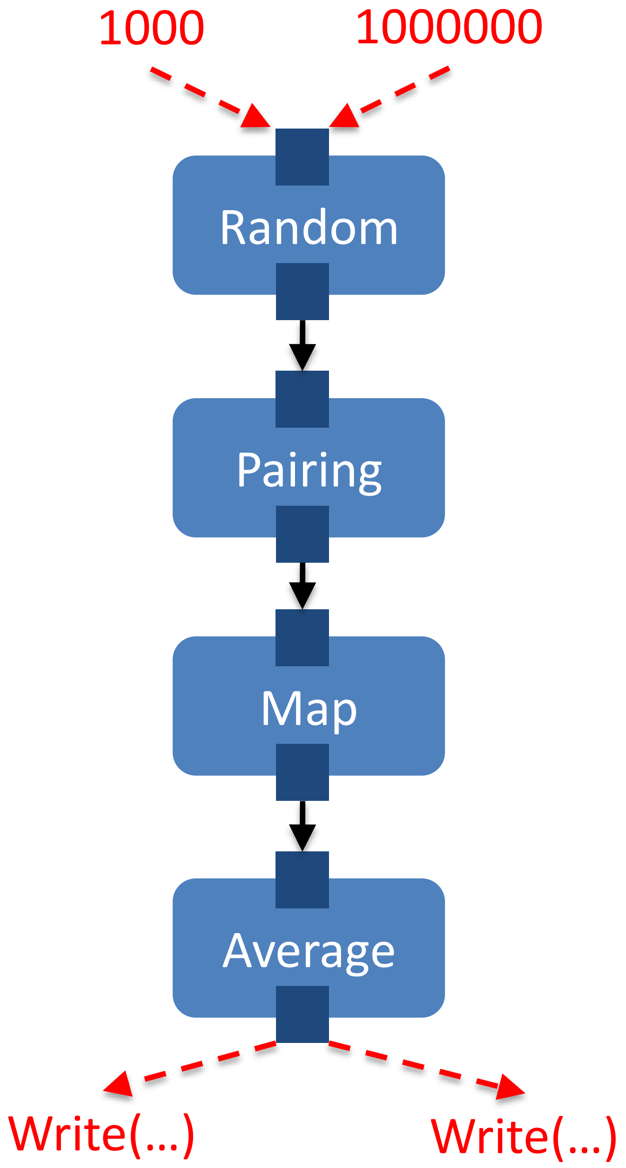


```
var randoms = new Random<float>(0, 1);
var coordinates = new Pairing<float>();
var inUnitCircle = new Map<Pair<float>, float>
  (p => p.Left * p.Left +
        p.Right * p.Right <= 1
        ? 1f : 0f);
var average = new Average<float>();

randoms.Output.ConnectTo(coordinates.Input);
coordinates.Output.ConnectTo(inUnitCircle.Input);
inUnitCircle.Output.ConnectTo(average.Input);
```

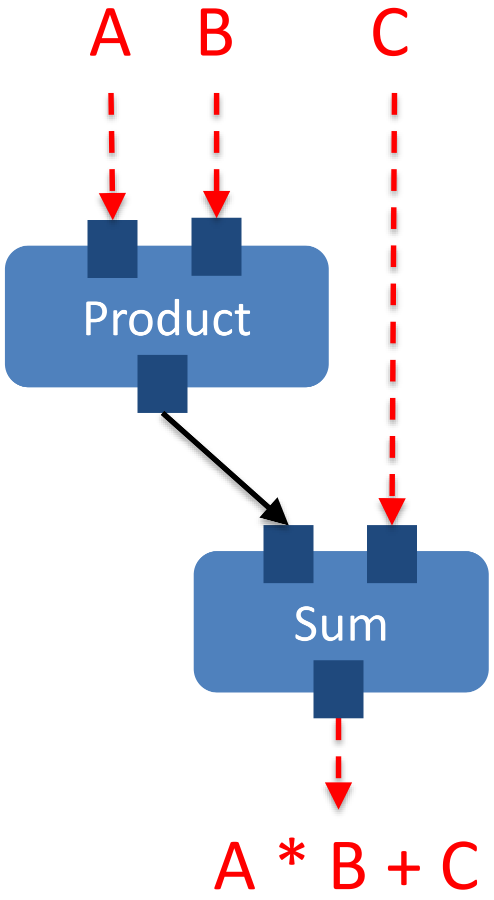
Dataflow

- Send data to input port
- Receive from output port
- All asynchronous

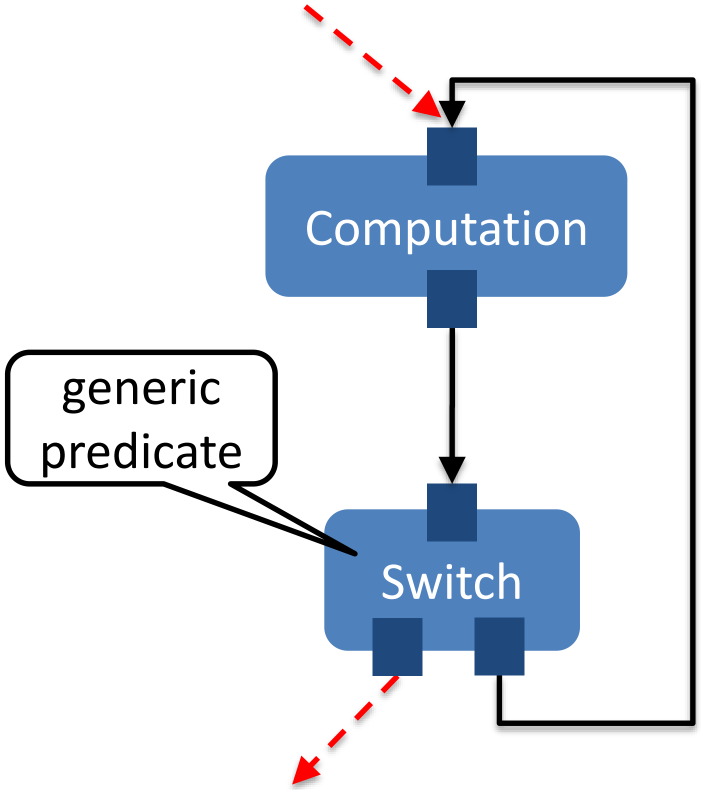
```
average.Output.OnReceive(x ->
        Console.WriteLine(4 * x));

random.Input.Send(1000);
random.Input.Send(1000000);
```



More Patterns



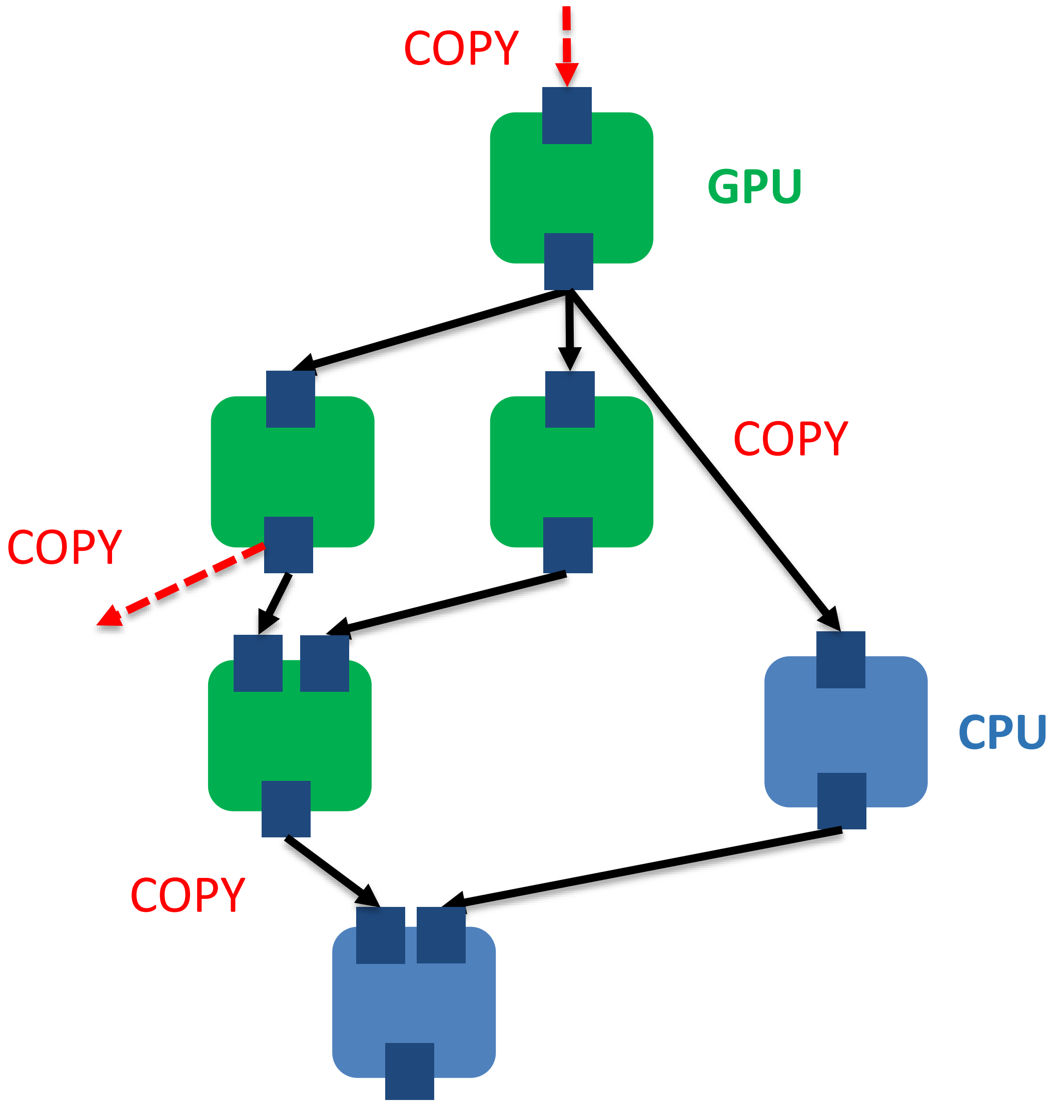
Algebraic Computation



Iterative Computation

Scheduler Behind the Scenes

- Operation implement GPU and/or CPU
- GPU operations script their launch configurations
- Automatic memory copying – only when needed
- Automatic free memory management
- Host scheduling with .NET TPL



Custom Operation (1)

Single-input/output operation

```
public sealed class Reduce<T> : Operation<T[], T> {
  private Implementation _cudaImpl;
  private Implementation _cpuImpl;

  public Reduce(Func<T, T, T> aggregator) {
   _cudaImpl = new CudaReduceImplementation<T>(aggregator);
   _cpuImpl  = new CpuReduceImplementation<T>(aggregator);
  }

  protected override Implementation[] GetImplementations() {
    return new[] { _cudaImpl, _cpuImpl };
  }
}
```

generic delegate

supported HW implementations

Custom Operation (2)

```
class CudaReduceImplementation<T> : CudaImplementation<Reduce<T>> {
  [Kernel] private void ReduceKernel(…)  { … calls aggregator … }

  protected override void Plan(Reduce<T> operation, CudaScript script) {
    var input = script.Consume(operation.Input);
    int n = input.Length;
    DeviceArray1D<T> output = null;
    while (n > 1) {
      int blocks = …;
      output = script.Malloc<T>(blocks);
      script.Launch(new LaunchParam(…), ReduceKernel);
      input = output; n = blocks;
    }
    script.Produce(operation.Output, script.SingleValue(output));
  }
}
```

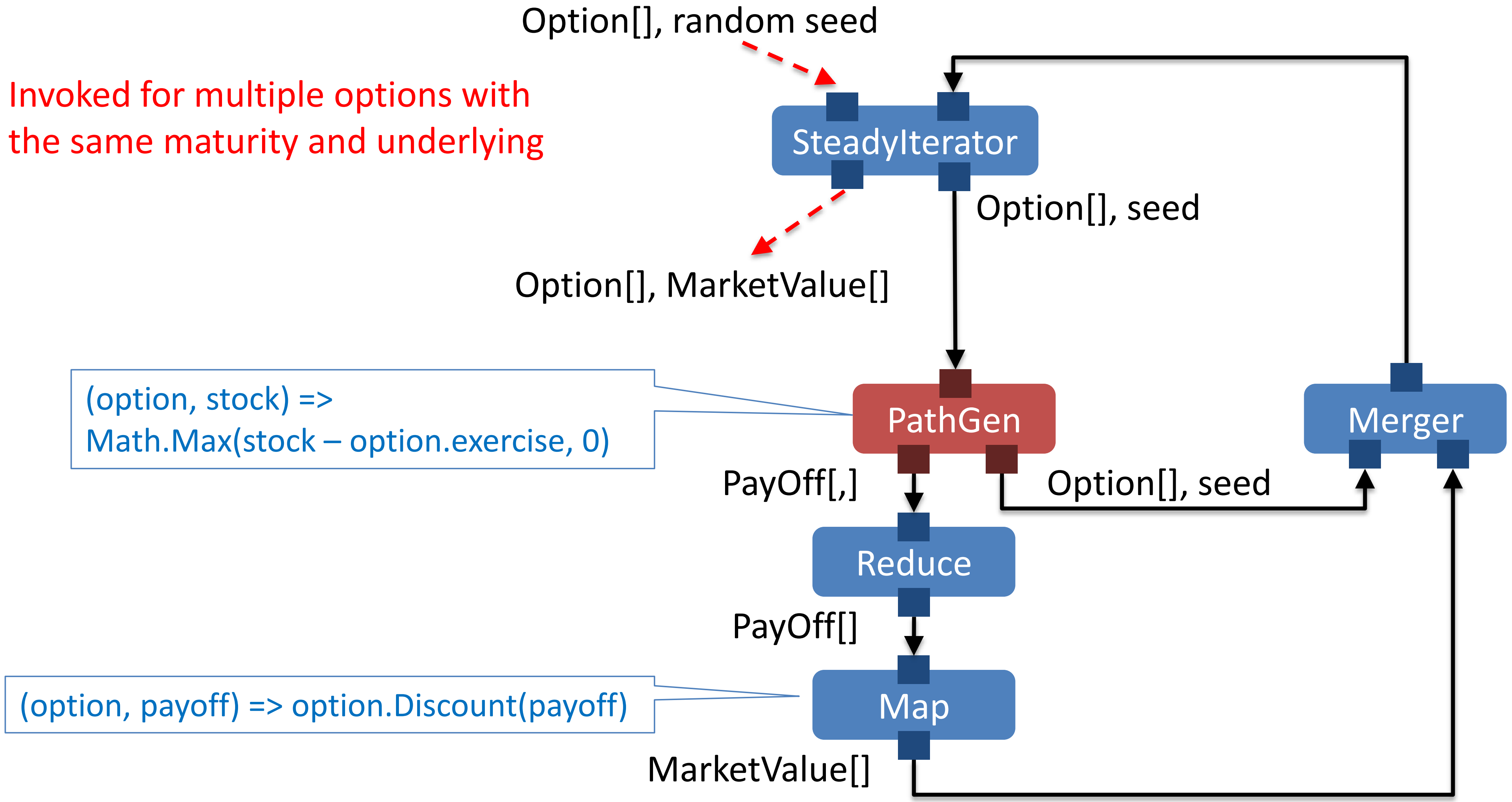
execution script for the future

Practical Application Cases

- Monte Carlo Option Pricing Engine
  - □ Evaluating options with random stock price paths
- Neural Network Training Kernel
  - □ Recognizing handwritings

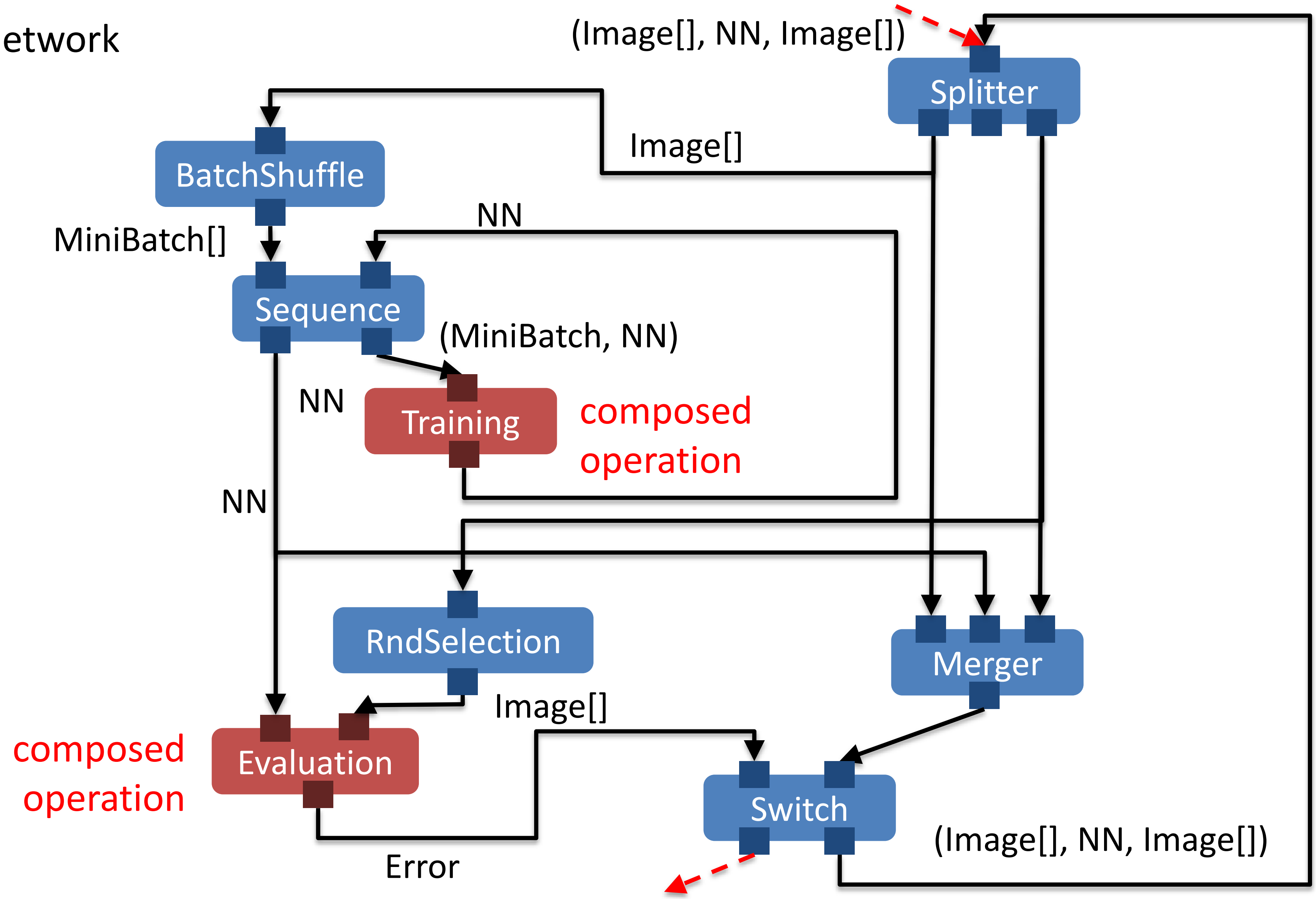
Monte-Carlo Option Pricing

Invoked for multiple options with the same maturity and underlying



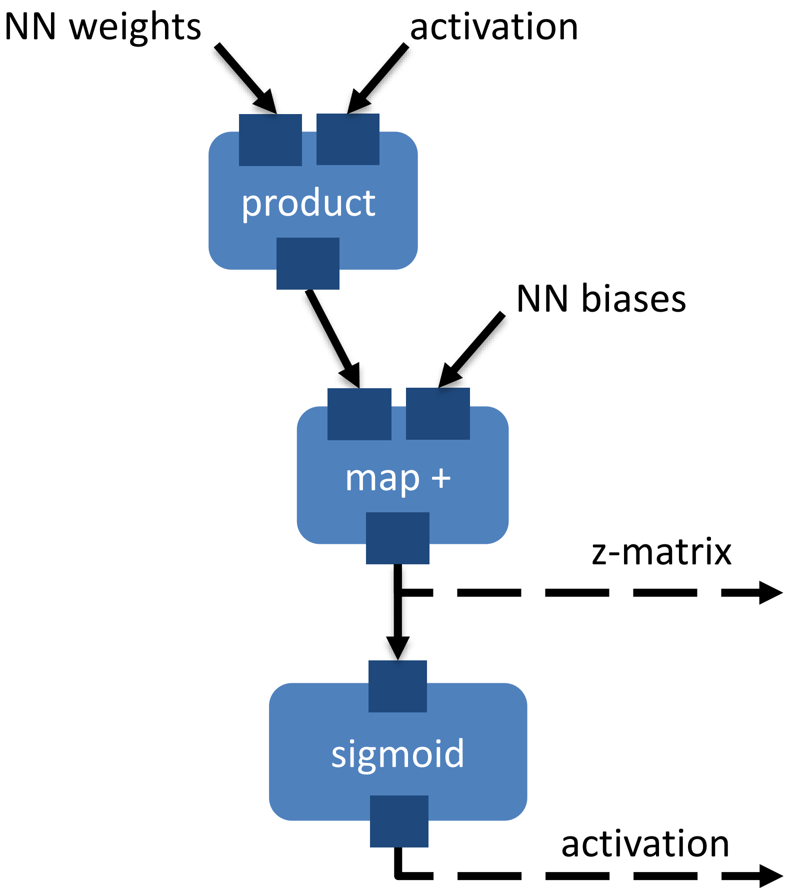
Neural Network Training

NN = Neural Network

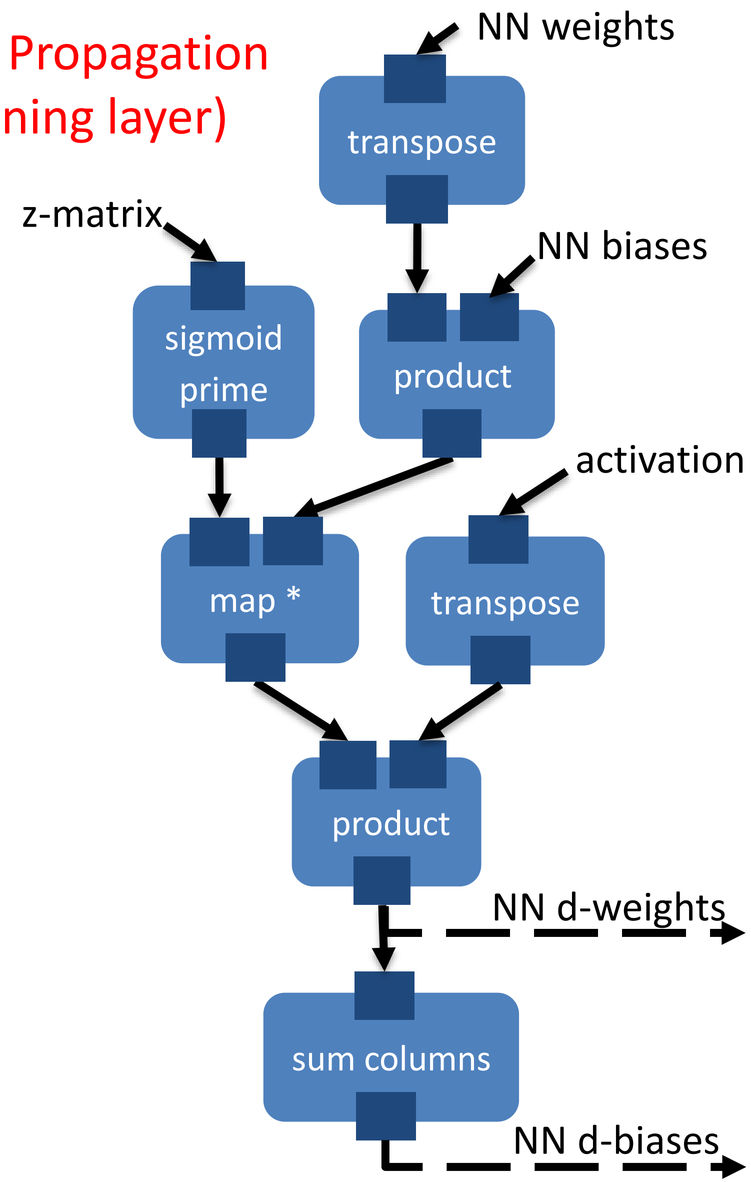


Digging Into Composed Operations

Feed Forward
(per training & evaluation layer)



Backward Propagation
(per training layer)



Performance

- GeForce GTX Titan Black (2880 cores) vs.
  CPU (4 Core Intel Xeon E5-2609 2.4GHz)
- Option Pricing Case (1 year)

Configuration	Speedup
32 options, 16k paths/it, 30 days	6
32 options, 32k paths/it, 90 days	18
32 options, 128k paths/it, 360 days	30

- Training Phase of Neural Network Case (MNIST data)

Configuration	Speedup
60k images, 750 size, 30 hidden neurons	1
60k images, 3k size, 200 hidden neurons	20
60k images, 3k size, 600 hidden neurons	30

Conclusions

- Simple GPU parallelization in .NET
  - □ Fast development without GPU knowledge …
  - □ … when using prefabricated operations
- Efficient runtime system
  - □ Automatic memory management
  - □ <20% overhead compared to native C CUDA code
- Expressive and extendible
  - □ Asynchrony allows cycles, infinite streams
  - □ Generic operations, custom operations

Further Information

- www.quantalea.com

Prof. Dr. Luc Bläser
HSR Hochschule für Technik Rapperswil
Institute for Software
lblaeser@hsr.ch
http://concurrency.ch

Dr. Daniel Egloff
QuantAlea GmbH

Switzerland
daniel.egloff@quantalea.net
www.quantalea.com