The Alea Reactive Dataflow System for GPU Parallelization

Philipp Kramer, 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

HLPGPU 2016
19 Jan 2016

GPU Parallelization Is Tough

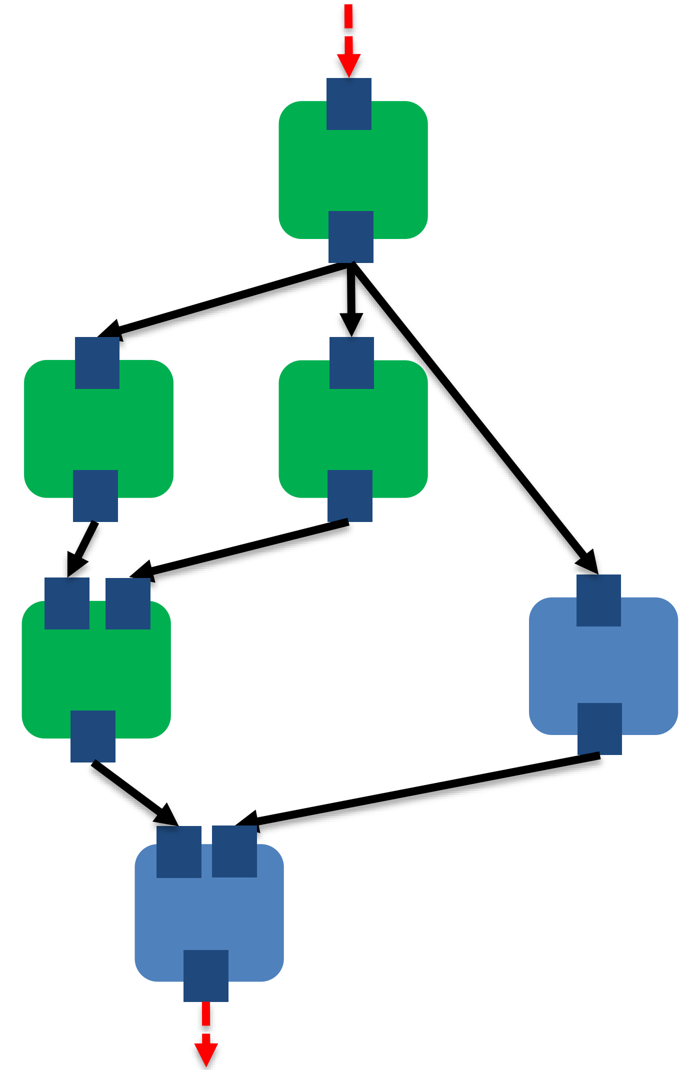
- Massive Parallel Power
- Not easy to use
  - Vector parallel and machine-centric programming models
  - Invocation and data management logic

Simplify GPU Programming

- 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 (pre-fabricated)
  - Data propagated through graph
- Reactive
  - Feed input in arbitrary intervals
  - Listen for asynchronous output



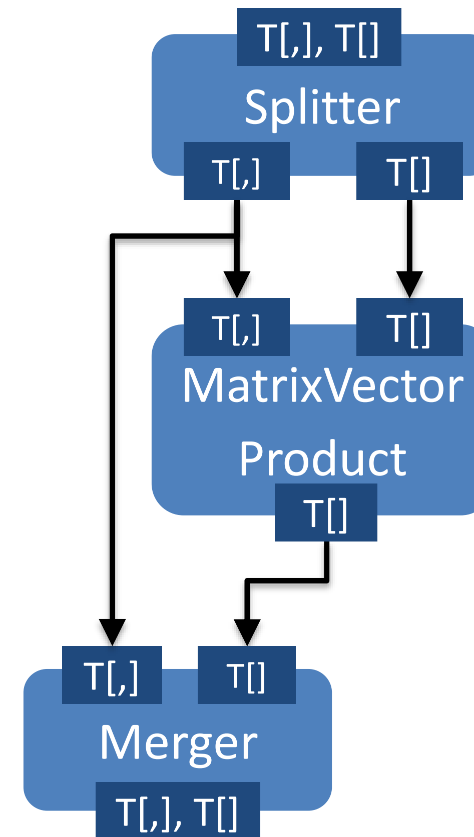
Graph Construction

e.g. Markov-Chain-Step:

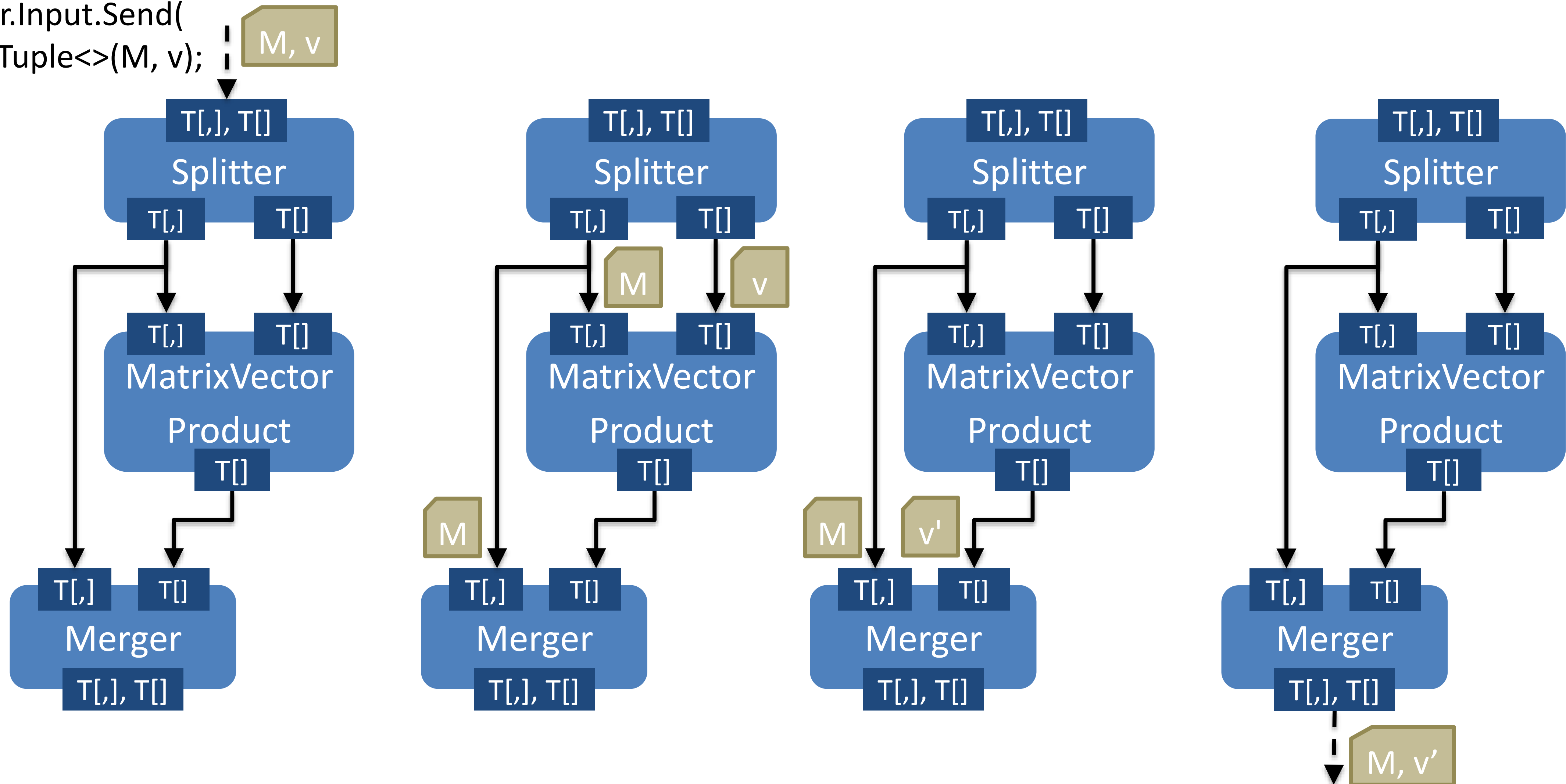
```
var splitter = new Splitter<float[,], float[]>();
var mvp = new MatrixVectorProduct<float>();
var merger = new Merger<float[,], float[]>();

splitter.First.ConnectTo(mvp.Left);
splitter.First.ConnectTo(merger.First);
splitter.Second.ConnectTo(mvp.Right);
mvp.ConnectTo(merger.Second);

splitter.Second.OnReceive(Console.WriteLine);
```



Dataflow Execution

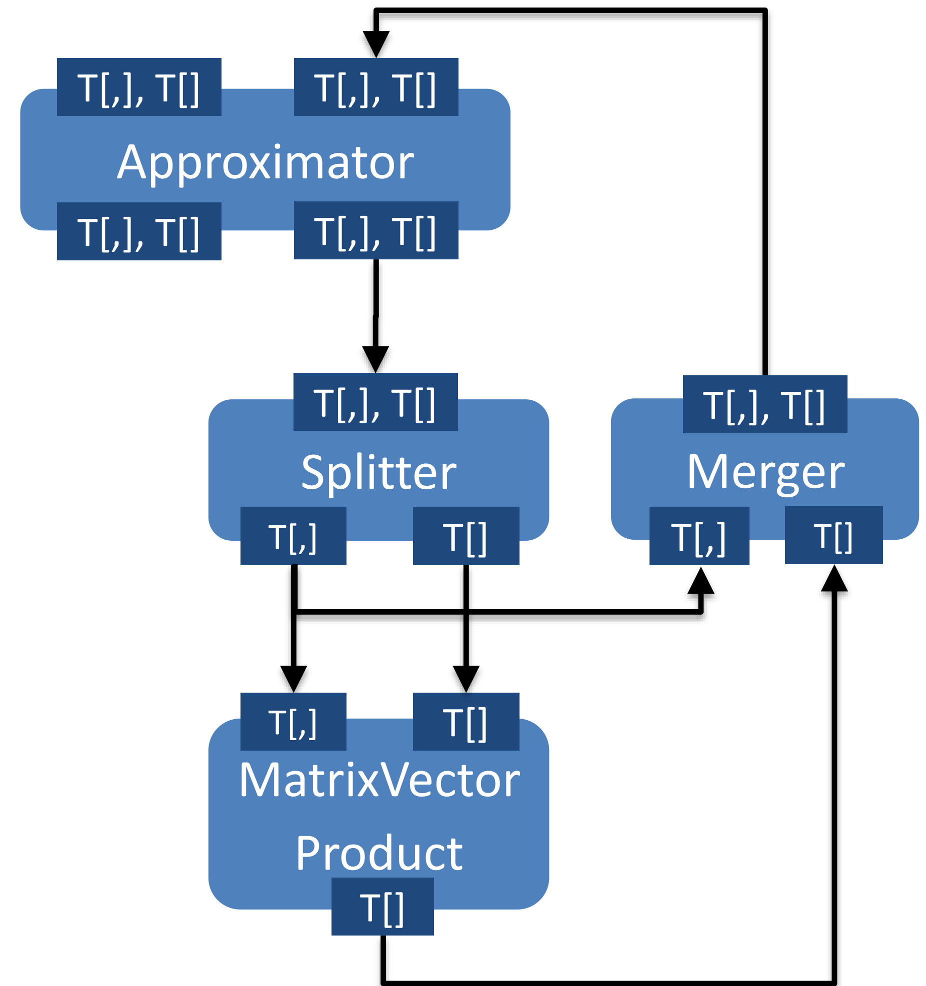


Cyclic Graph

Corresponds to iterative calculation

e.g. Markov-Chain steady-state:

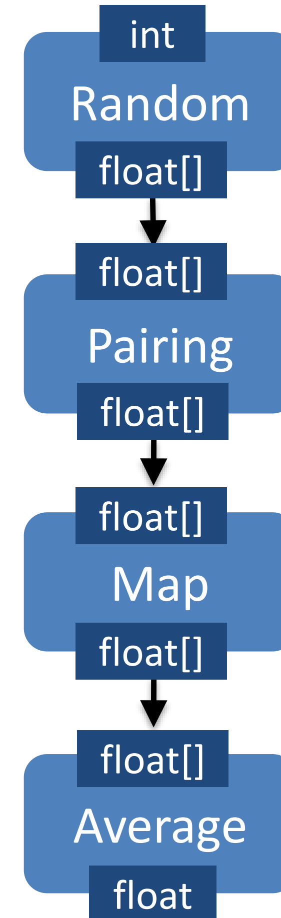
```
Func<int, float[], float[], bool> steadyCondition = …;
Func<float[], float[], float[]> aggregation = …;
var approx = new Approximator<float[,], float[]>(
    steadyCondition, aggregation);
approx.Iterate.ConnectTo(splitter);
merger.ConnectTo(approx.Continue);
```



Fluent Graph Construction

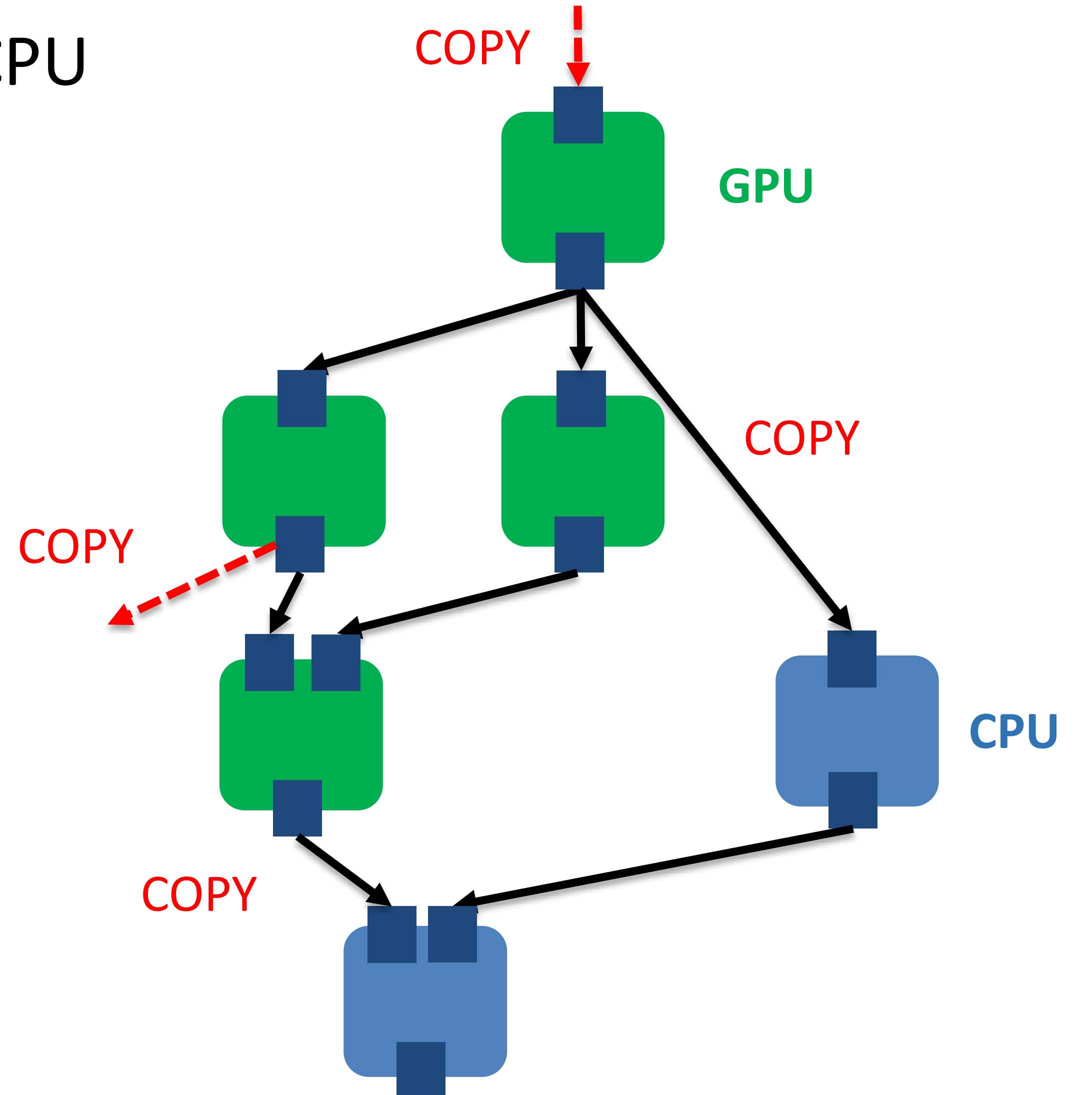
e.g. Monte Carlo Pi

```
var randoms = new RandomMersenne<float>(0, 1);
randoms
    .Pairing()
    .Map(p => p.Left * p.Left + p.Right * p.Right <= 1 ? 1f : 0f)
    .Average()
    .OnReceive(x => Console.WriteLine(4 * x));
```



Runtime System behind the Scenes

- Operations implement GPU and/or CPU
- Automatic memory copying
  - only when needed
- GPU garbage collection
- Multi-GPU support
- Scheduling with .NET TPL



GPU Parallelization can be Easy

- Pre-fabricated and custom operations
  - Vector parallel and machine-centric programming models
- Automatic orchestration
  - Invocation and data management logic
  - Garbage collection
  - Synchronization

5760 Cores

Performance Comparison to CUDA C

- GPU Kernels written in C# in Cuda C style

- Comparison of GPU-kernel-performance to CUDA C
  - GeForce GTX Titan Black (2880 cores)

Benchmark	Speedup
Average	1.12
Matrix Convolution	1.03
Matrix $AB + BC + CA$	0.81
Monte Carlo π	1.06

Applications and Performance

- GeForce GTX Titan Black (2880 cores) vs.
  CPU (4 Core Intel Xeon E5-2609 2.4GHz)
- Option Pricing Case (single GPU, 32 options)

Configuration	Speedup
16k paths/it, 30 days	6
32k paths/it, 90 days	18
128k paths/it, 360 days	31

- Training Phase of Neural Network Case (MNIST data)

Configuration	Unopt. Speedup	Opt. Speedup
30 hidden neurons	2.4	3.3
480 hidden neurons	24	33
7680 hidden neurons	93	124

Conclusions

- Simple GPU parallelization in .NET
  - Fast development with generic prefabricated operations
  - Cleaner and slimmer code by separation of concerns
  - Possibility to define custom operations
- Efficient runtime system
  - Automatic data movement and GPU garbage collection
  - Low overhead compared to CUDA C

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