Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

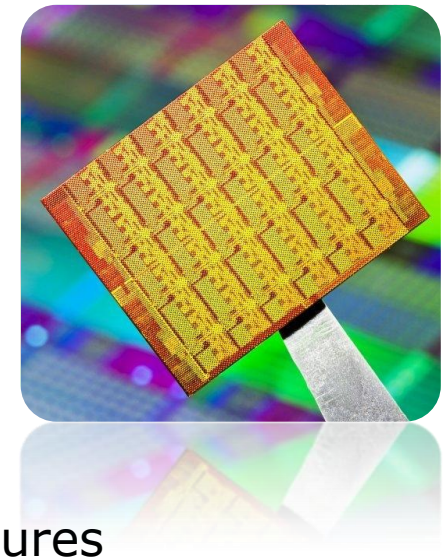
Programming Models for Parallel ~~Het~~ Hybrid ~~ous~~ Computing

Frank Feinbube

HPI Research School
Operating Systems and Middleware
Prof. Dr. Andreas Polze

Motivation

- A lot of big computational problems still to solve
- Processors will not get much faster, but will get more cores instead
  - Harder to utilize by programmers
  - New challenges for hardware designers



- Approaches towards future hardware architectures
  - New architectures are evaluated (Intel SCC)
  - Accelerators that accompany common general purpose CPUs (Hybrid Systems)

Domains of Hybrid Systems

High Performance Computing (HPC)

- Huge mathematical workloads
- More floating point operations per second (FLOPS)
- As energy efficient as possible
- ➢ Heterogeneous computing with GPU compute devices

Business Servers

- Huge amounts of business data → complex analysis
- ➢ Special Accelerators:
  - Additional blades that accompany reliable mainframes
  - (de)compression, XML parsing, [en|de]cryption, regular expression matching

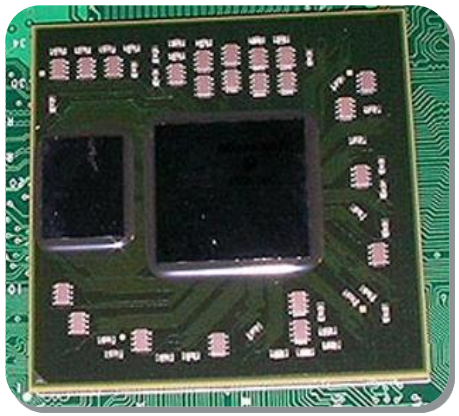
Domains of Hybrid Systems

Home / Desktop Computers

- Netbooks
  - Always-on, mobile → need to be energy efficient
  - Entertainment device → need for good performance
  - ➢ On-Chip GPUs like NVIDIA ION GPU
- Medium-class & high-end computers
  - Realistic simulations with BOINC → GPU accelerated

Mobile and Embedded Systems

- Severe restrictions on power consumption
- Demanding applications
- ➢ Co-processors, powerful GPU-compute devices



Research Focus

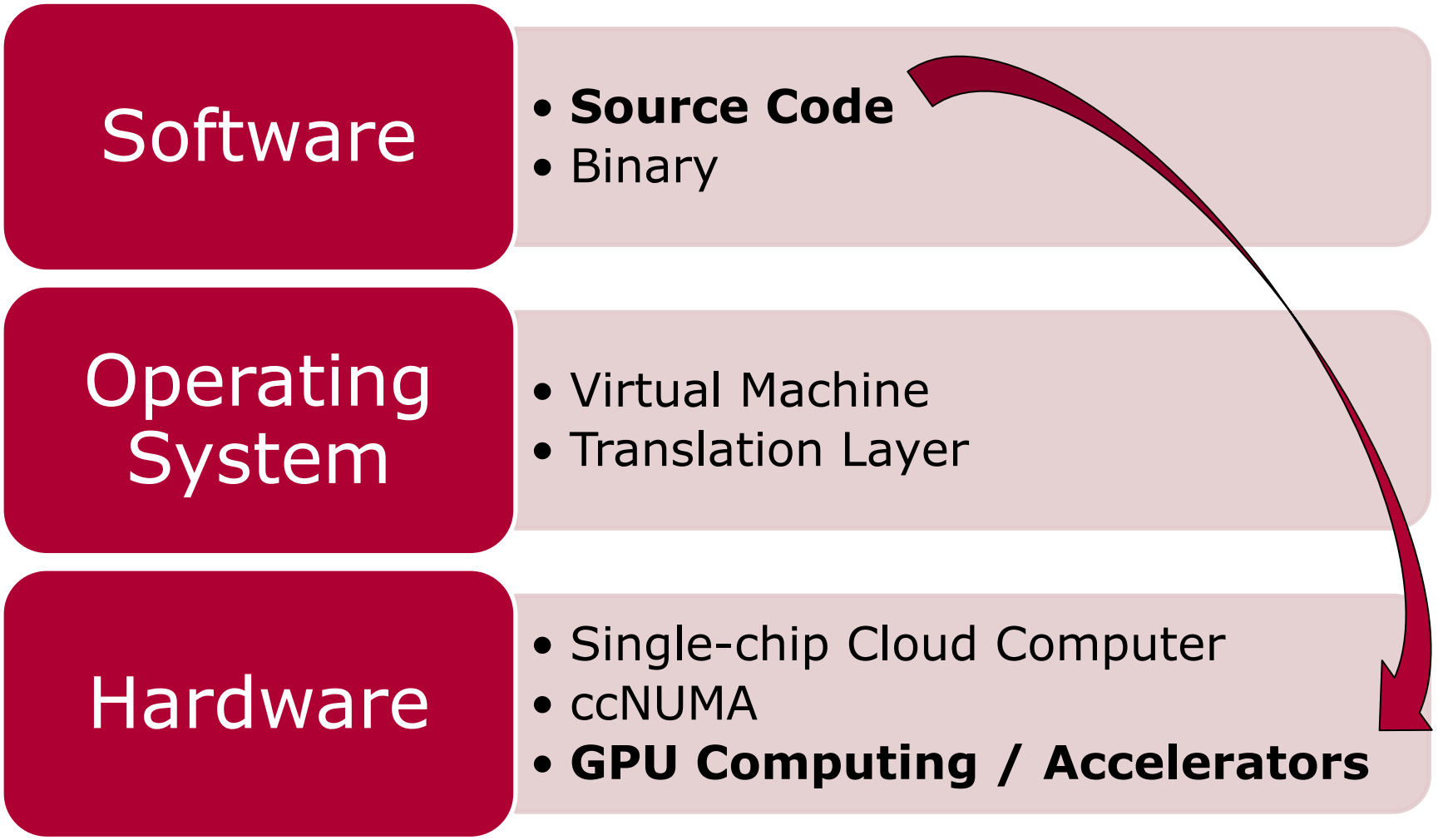
Context

- Parallel Hybrid Systems and Accelerator Technologies
- Parallel Language Extensions, Parallel Libraries, Parallel Toolkits

Problem Statement

- How can we help developers to handle the complexity of parallel hybrid architectures w.r.t. coding experience and resulting application performance?

Research Context



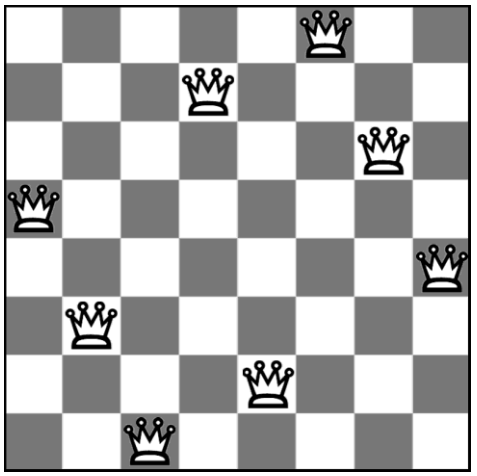
Approach

1. Identify hardware capabilities and restrictions of hybrid architectures.
2. Identify best practices and patterns for multi-core development and hybrid computing.
3. Identify common use cases and algorithms for hybrid computing.
4. Reduce the complexity for developers using high-level languages by introducing abstractions and exploiting runtime reflection mechanisms.
5. Demonstrate the solution with representative example uses cases and algorithms (on various platforms).

Paper presentation: NQueens on CUDA: Optimization Issues

Frank Feinbube, Bernhard Rabe, Martin von Löwis, Andreas Polze

- Experience in applying the NQueens puzzle solution on GPUs using Nvidia's Compute Unified Device Architecture
- Demonstrate that optimizations of CUDA programs may have contrary results on different CUDA architectures
- Evaluation results → it is not sufficient to use new programming languages or compilers to achieve best results with emerging GPU computing



Approach

- ✓ Identify hardware capabilities and restrictions of hybrid architectures.
2. Identify best practices and patterns for multi-core development and hybrid computing.
3. Identify common use cases and algorithms for hybrid computing.
4. Reduce the complexity for developers using high-level languages by introducing abstractions and exploiting runtime reflection mechanisms.
5. Demonstrate the solution with representative example uses cases and algorithms (on various platforms).

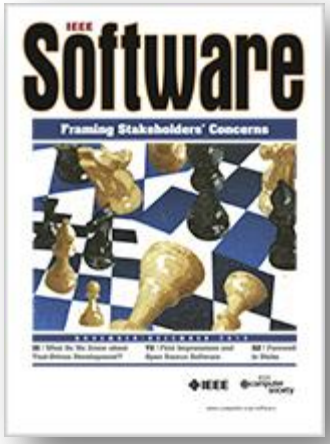
Journal Paper for IEEE Software: Survey on Best Practices for Optimizations in GPU Computing

Publication in IEEE Software's SI: Jan/Feb 2011

„Joint Forces - From Multithreaded Programming to GPU Computing"

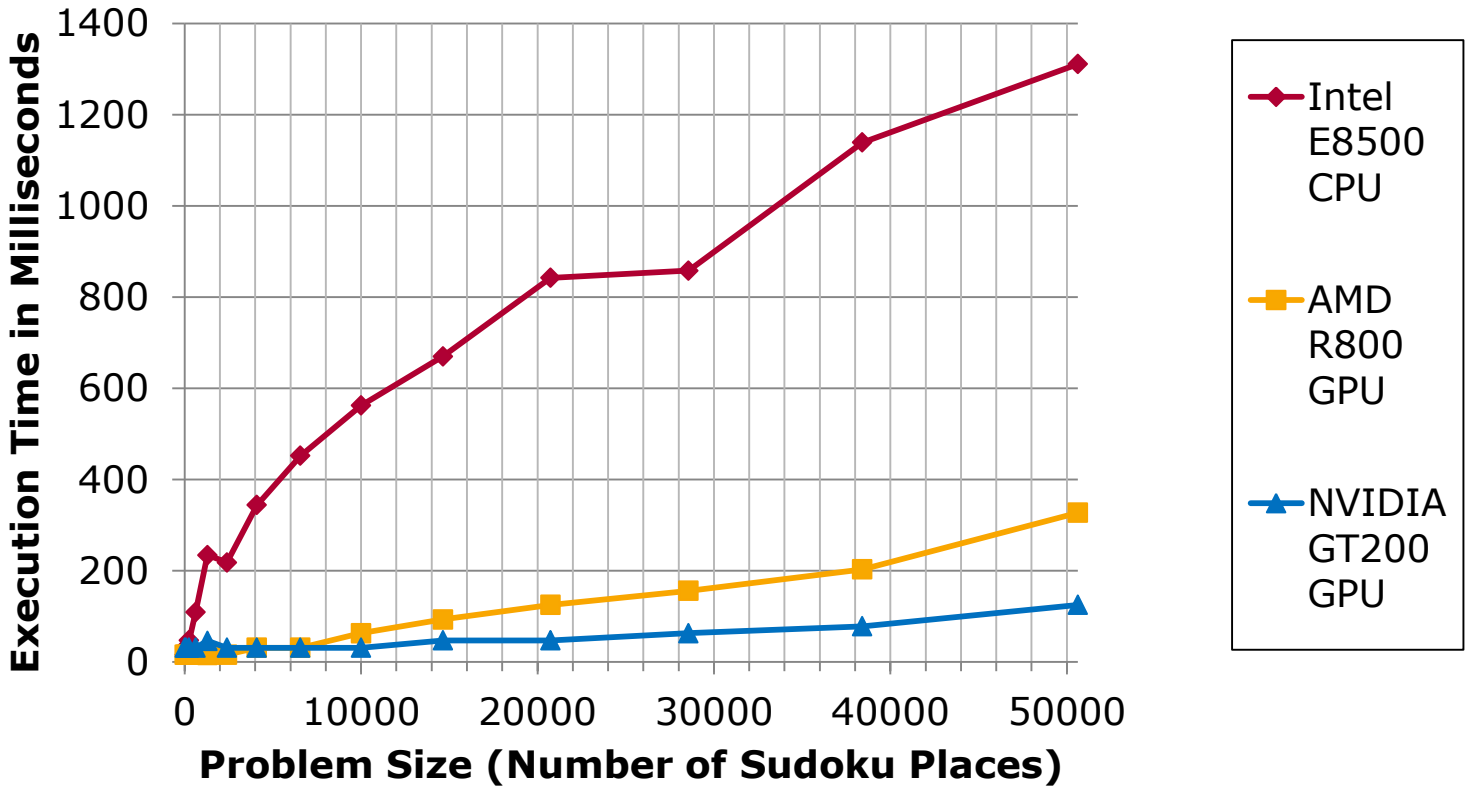
Frank Feinbube, Peter Tröger, Andreas Polze

- How to utilize GPU compute power in the best possible way.
- Explain the primary concepts of GPU hardware and the according programming principles.
- Discuss a collection of commonly agreed critical performance optimization strategies.
- These optimizations are the key factor for getting true scalability and performance improvements when moving from a multithreaded to a GPU-enhanced version.



Journal Paper for IEEE Software: Survey on Best Practices for Optimizations in GPU Computing

big performance gains for small problem sizes

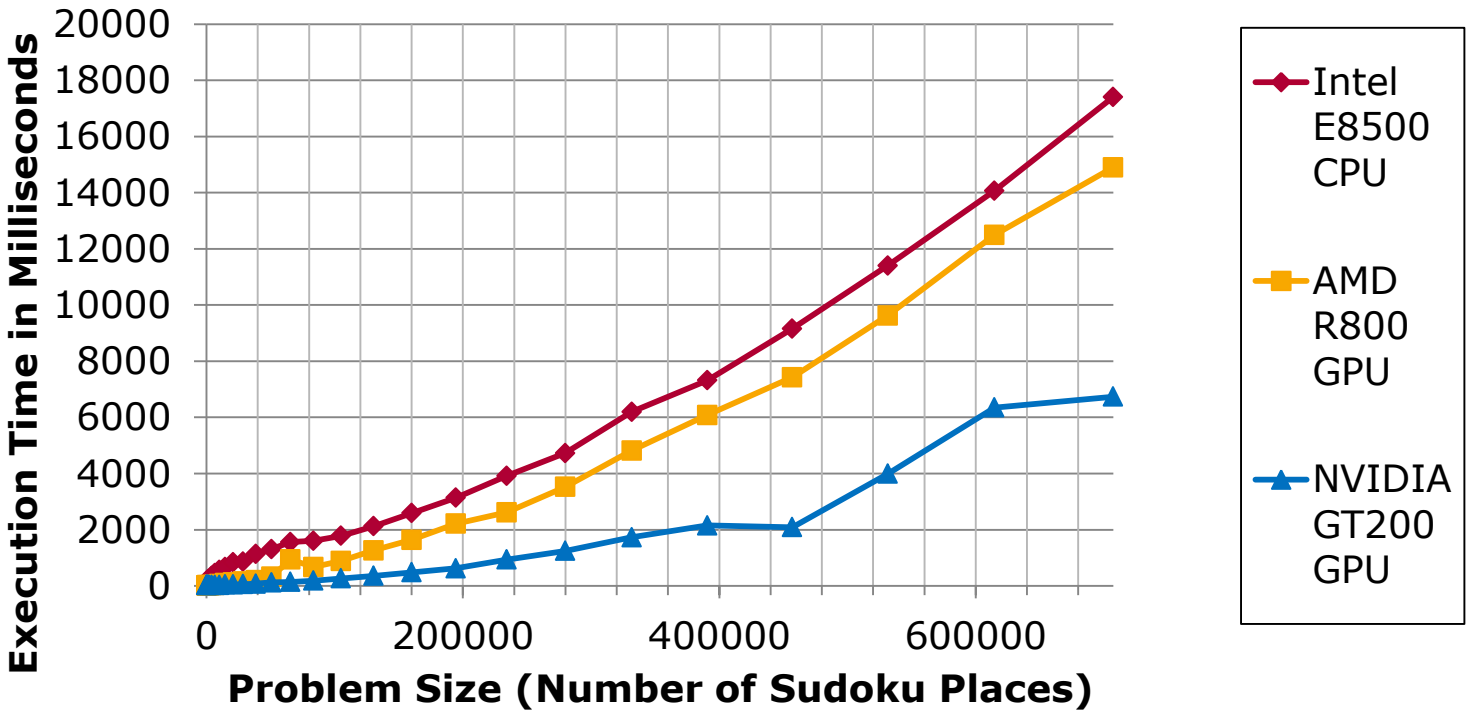


* less is better

Journal Paper for IEEE Software: Survey on Best Practices for Optimizations in GPU Computing

small/moderate performance gains for large problem sizes

→ further optimizations needed



* less is better

Journal Paper for IEEE Software: Survey on Best Practices for Optimizations in GPU Computing

Strategy	Importance	
	ATI	NVIDIA
Algorithm Design		
Use host asynchronously	Low	Low
Recompute instead of transferring	**High**	**High**
Use simple work items	Medium	Medium
Memory Transfer		
Reduce input data size	**High**	**High**
Chain work items	Medium	Medium
Move more operations to GPU	Medium	Medium
Overlap transfer and compute time	Medium	Medium
Pass memory directly to the work item	Low	Low
Control Flow		
Avoid divergent branching	**High**	**High**
Support predication	Medium	Low
Memory Types		
Reduce per thread memory usage	**High**	**High**
Store arguments in constant memory	Low	Low
Use OpenCL images for data structures with hard-to-predict access patterns	Low	Low
Use local as a cache for global memory	**High**	**High**
Avoid global arrays on work item stacks	Medium	Medium
Collaboratively write to local memory	Medium	Medium

Journal Paper for IEEE Software: Survey on Best Practices for Optimizations in GPU Computing

Strategy (cont.)	Importance	
	ATI	NVIDIA
Memory Access		
Consider trade-off between work item count and memory usage	**High**	**High**
Ensure device memory accesses are coalesced	**High**	**High**
Avoid memory bank conflicts	**High**	**High**
Access texture memory with appropriate work-group aspect ratio	Low	Low
Sizing		
Local work group size should be a multiple of the processing element's execution size	**High**	**High**
Ensure device memory accesses are coalesced	**High**	**High**
Evaluate different work group sizes	Medium	Medium
Instructions		
Use shift operations instead of division and modulo	Low	Low
Use fused multiply add when appropriate	Low	Low
Use vector types and operations	**High**	-
Precision		
Avoid automatic conversion of doubles to floats	Low	Low
Use the native math library for low precision tasks	-	Medium
Use build options that trade precision for speed	-	**High**

Approach

- ✓ Identify hardware capabilities and restrictions of hybrid architectures.
- ✓ Identify best practices and patterns for multi-core development and hybrid computing.
3. Identify common use cases and algorithms for hybrid computing.
4. Reduce the complexity for developers using high-level languages by introducing abstractions and exploiting runtime reflection mechanisms.
5. Demonstrate the solution with representative example uses cases and algorithms (on various platforms).

Knowledge Sink for Use Cases, Tools and Libraries

- Readings collection for GPU computing (44 Publications)
- GPU Compute Use Cases (35 Publications)
- GPU Computing Tools and Libraries (33 Publications)

http://www.dcl.hpi.uni-potsdam.de/research/gpureadings/

UPCRC Illinois Summer School on Multicore Programming 2010

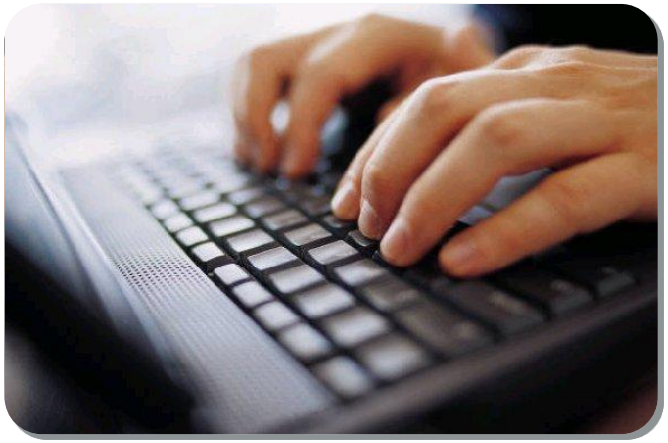
- Shared Memory & Parallel Programming Models
- Control Parallelism in OO Languages, Refactoring for Parallelism
- Shared Memory Control Parallelism: OpenMP, Intel TBB
- Intel Parallel Inspector, Intel Parallel Amplifier, Intel Parallel Advisor
- Visual Studio 2010: C++ Concurrency Runtime, .NET Parallel Extensions
- Vectorization
- The OpenCL Programming Model

- Hands-On-Labs:
  - Java, OpenMP, TBB, .NET TPL, Vectorization, OpenCL

Example Implementations of Representative Use Case

- Based on UPCRC hands-on-labs
- Extended based on „CUDA by Example“ by Sanders et al., „Programming Massively Parallel Processors“ by Kirt et al., …

- Examples:
  - Matrix-Matrix Multiplication, Convolution, Prefix Scan, Quicksort, Minimum Spanning Tree
  - Dot Product, Heat Transfer, Histogram, Julia Set, Ray Tracing, Ripple, Summing Vectors
  - …



Approach

- ✓ Identify hardware capabilities and restrictions of hybrid architectures.
- ✓ Identify best practices and patterns for multi-core development and hybrid computing.
- ✓ Identify common use cases and algorithms for hybrid computing.
4. Reduce the complexity for developers using high-level languages by introducing abstractions and exploiting runtime reflection mechanisms.
5. Demonstrate the solution with representative example uses cases and algorithms (on various platforms).

Approach

- ✓ Identify hardware capabilities and restrictions of hybrid architectures.
- ✓ Identify best practices and patterns for multi-core development and hybrid computing.
- ✓ Identify common use cases and algorithms for hybrid computing.
- ➢ Reduce the complexity for developers using high-level languages by introducing abstractions and exploiting runtime reflection mechanisms.
5. Demonstrate the solution with representative example uses cases and algorithms (on various platforms).

Prototype to Run OpenCL-Code from the .NET Framework

- Library for .NET
- Executed on OpenCL-enabled GPUs

- Matrix-Matrix Multiplication using a simple For-loop approach
  - known to developers, because parallel-for is part of the .NET Framework 4.0

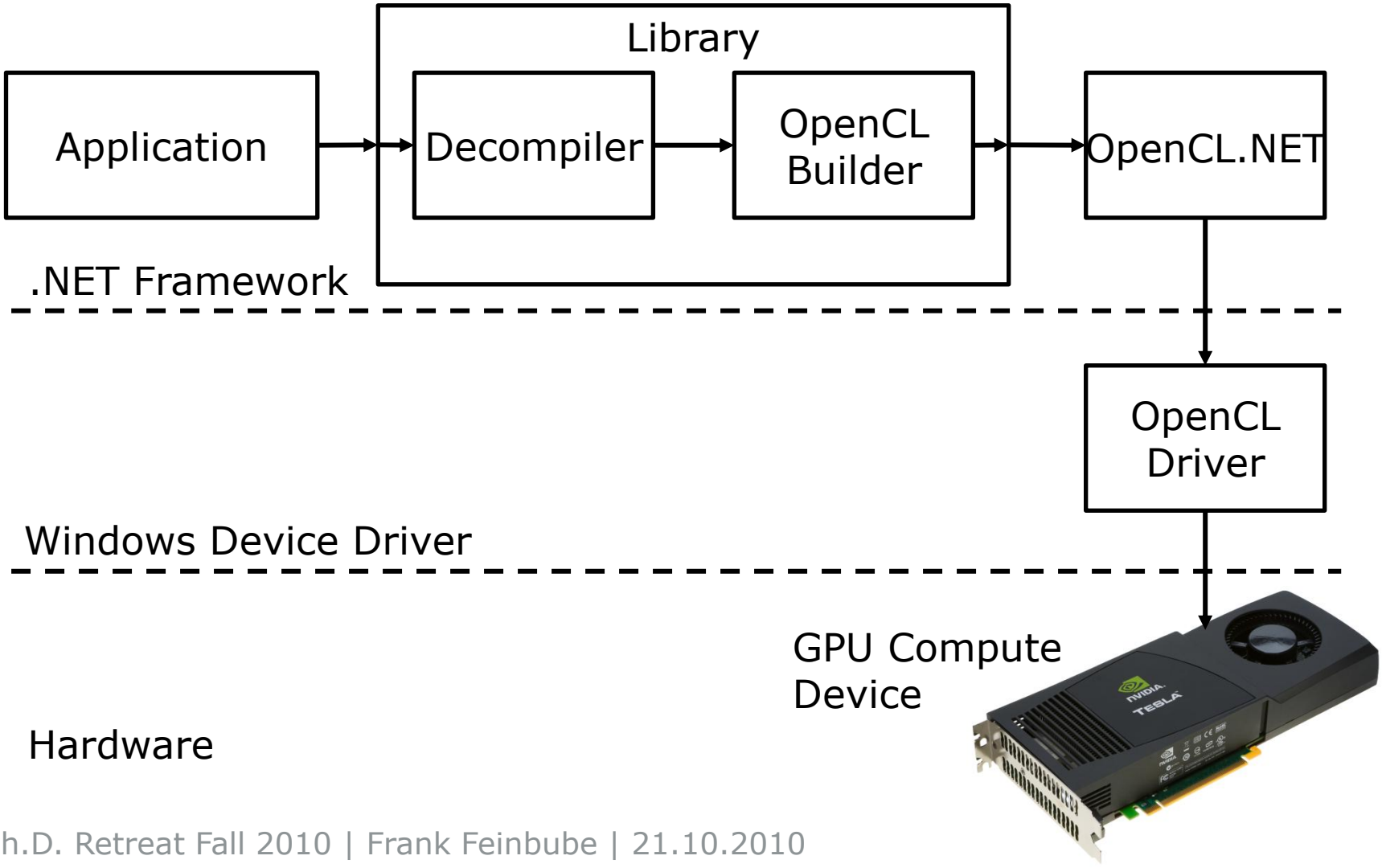
Microsofts Solution →

```
// Task Parallel Library
System.Threading.Tasks.Parallel.For(0, 100, action);
```

Our Solution →

```
// Our Library
GpuParallel.ForGpu(0, 100, action);
GpuParallel.ForGpu2D(0, 100, 0, 100, action);
```

Architecture



Next Steps

- Using the library for the other examples as well.
  → Demonstrate usefulness and performance gains.
- Provide additional constructs. (parallel Invoke, Barriers, Atomics, …)
- Apply best practice optimizations automatically.
  - Static: Memory Access Coalescing
  - Dynamic: Sizing
- Identify code parts that are adequate to be executed on an accelerator.

- Cooperations for further Use Cases?