Lift: Primitives for Hybrid CPU + GPU Computing

Nuno Subtil
nuno.subtil@roche.com

Lift in One Slide

https://github.com/nsubtil/lift

- **Lean and mean C++ parallel programming library**
  - Pass-by-value memory containers
  - Parallel primitive interface
  - Abstractions for GPU-specific features
  - GPU-aware test harness available for client code
  - Open-source (BSD license), runs on x86 and NVIDIA GPUs

- Foundation for Firepony (https://github.com/broadinstitute/firepony)
- **Actively used, developed and maintained by Genia Technologies**
  - Signal processing pipeline written entirely on top of Lift

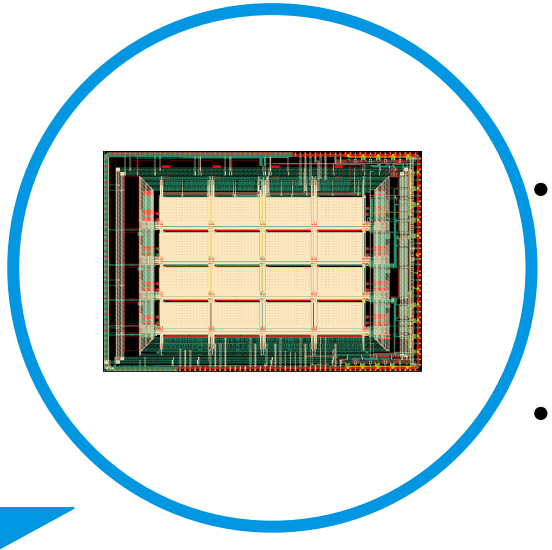
Genia's Next Generation Sequencing (NGS)

A powerful combination of electronics and molecular biology

Nanopore

Integrated Circuit (IC)

- Single Molecule Sequencing
- Long Read capabilities

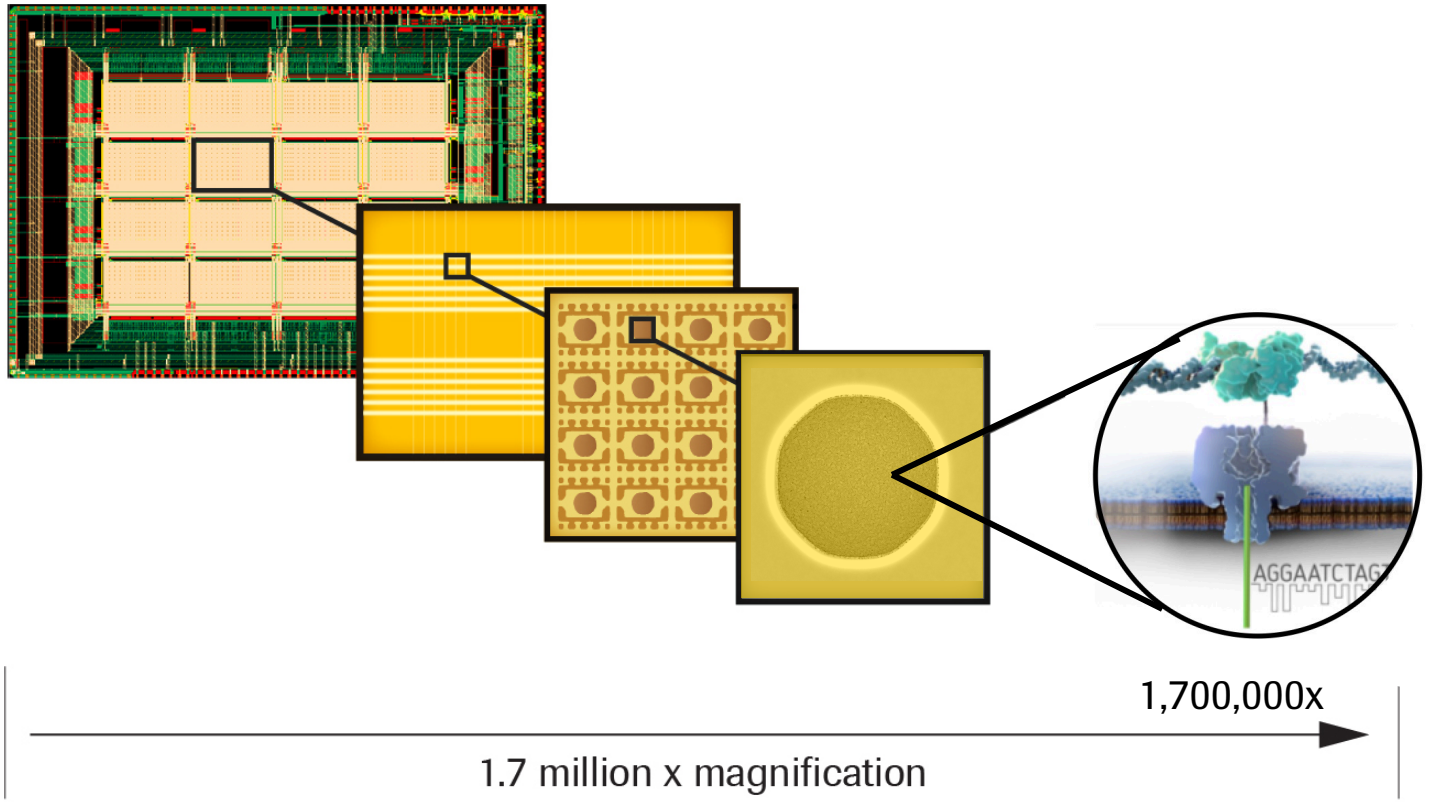


- Scalable, Electrical Detection
- Low Cost components

All the benefits of single molecule capability with semiconductor scalability

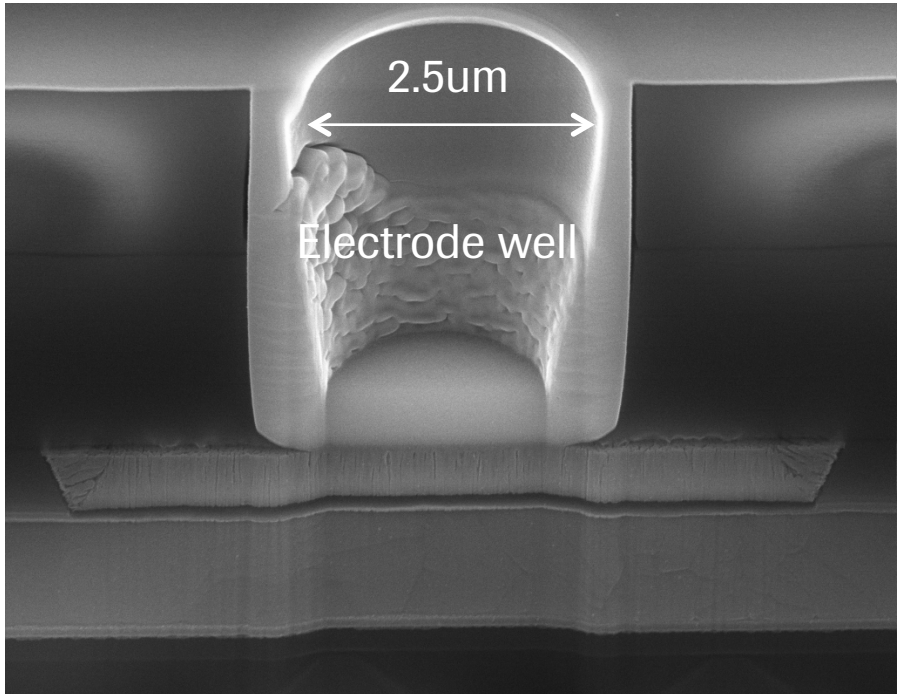
Integrated Circuits – A Scalable Solution

Sequencing Machines across the IC provide massively paralleled sequencing



Genia's Integrated Circuit (IC) Technology

Cost reductions driven by semiconductor scalability

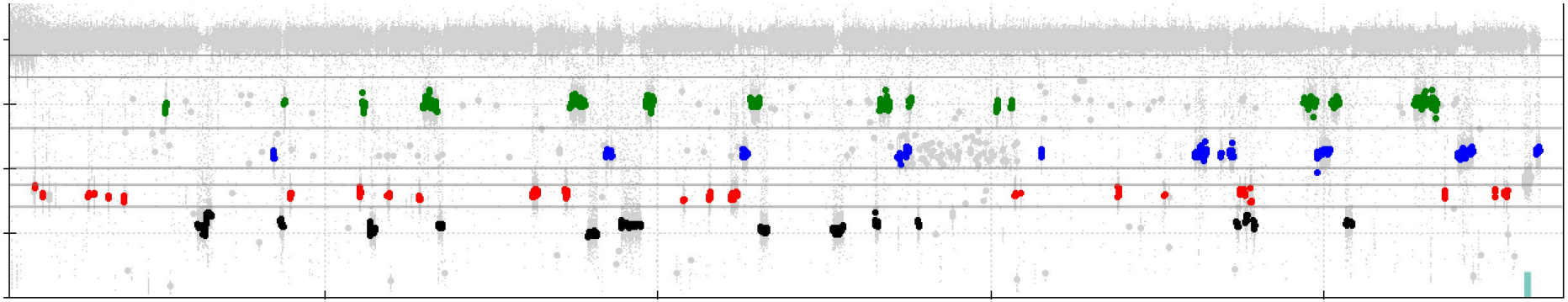


- Potential to drive sequencing costs downward
- Leverages standard IC manufacturing process
- Allows for scalable production and throughput

Single Molecule Nanopore Sequencing Data

Nucleotide Tags are readily distinguished in the Nanopore

Each tag signal represents a DNA nucleotide base of sequence

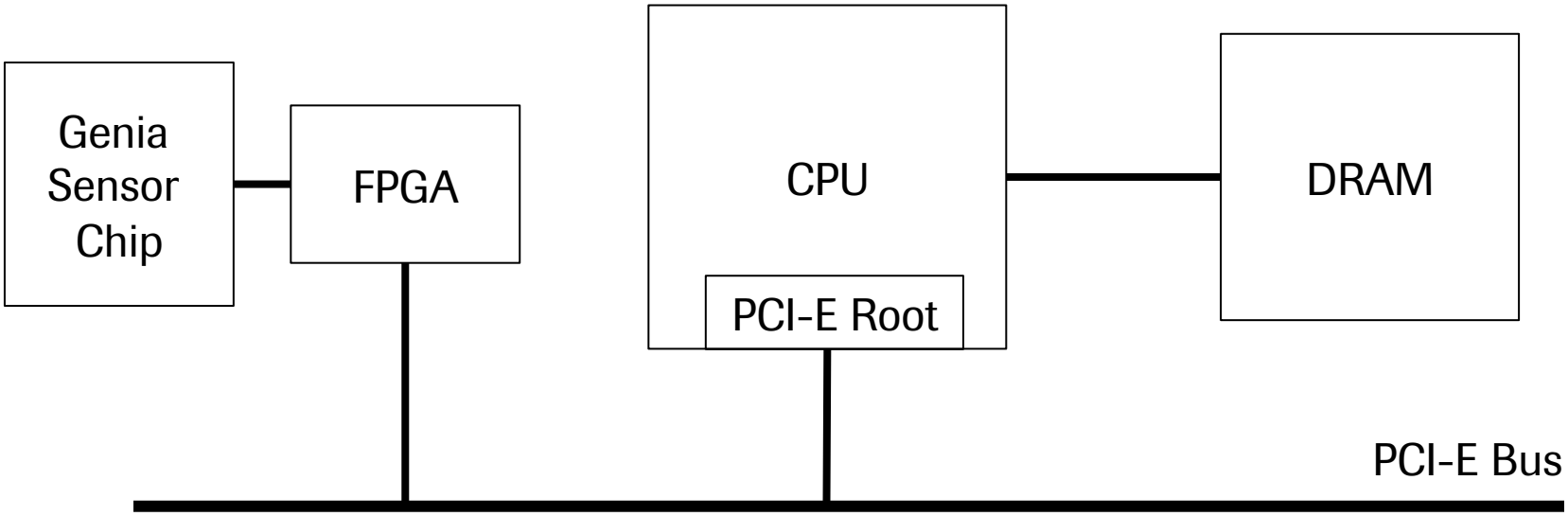


Tag Legend: **A C T G**

- Electrical detection of four tag levels

Data on file

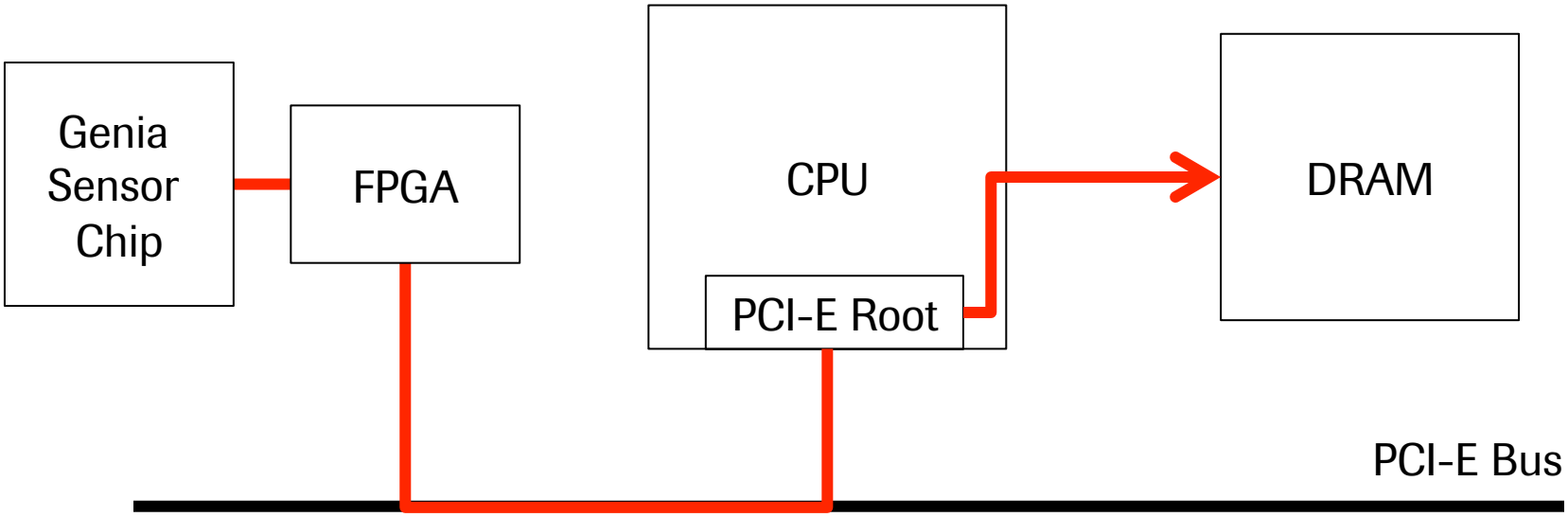
Genia System Overview



“Sequencing by Signal Processing”

- Sensor measures and outputs electrical signals
- Integrated sensor chip contains millons of individual sensors
- Base calls generated with a signal processing pipeline implemented in software

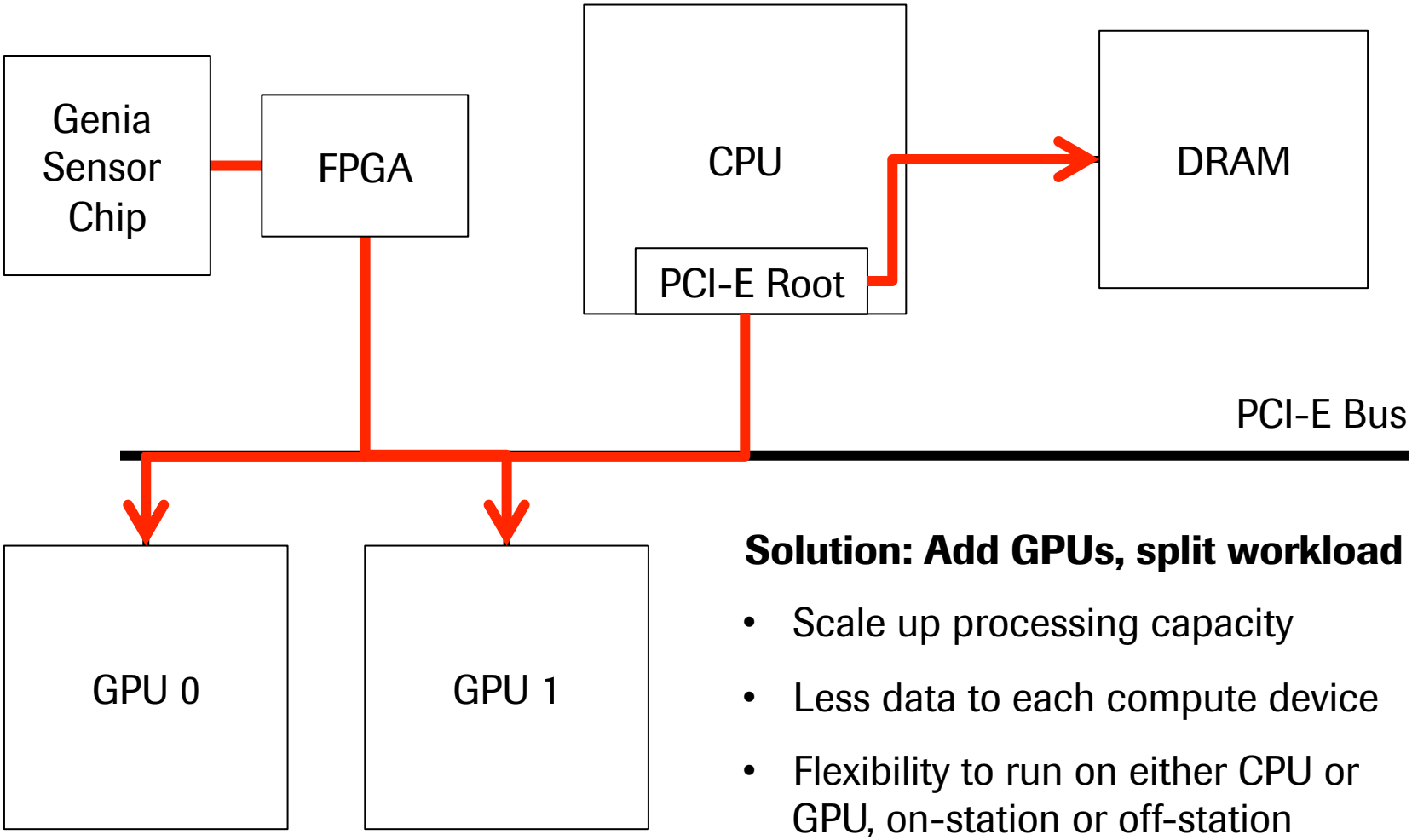
Genia System Overview



Projected Data Rate: 8GB/s

- Hard real-time requirement: SSDs can't keep up with raw signal data, can only store base calls
- CPU memory bus too slow for real-time processing

Genia System Overview



Solution: Add GPUs, split workload

- Scale up processing capacity
- Less data to each compute device
- Flexibility to run on either CPU or GPU, on-station or off-station

State of the art: CUDA + Thrust

Roche

- Memory containers: host_vector and device_vector
  - Implement familiar std::vector semantics

- “Smart” host/device pointers
  - Can dereference device pointers on host

- Parallel primitives: for_each, sort, scan, the works
  - Can take arbitrary Thrust pointers and schedule on CPU or GPU

genia

Motivation for Lift

Roche

- Memory containers: host_vector and device_vector
  - Implement familiar std::vector semantics
  - **Can not pass by value: containers are not valid on GPU code**
    - Code reuse across architectures becomes complex
- "Smart" host/device pointers
  - Can dereference device pointers on host
  - **Not a performance path**
    - Significant added complexity in the library codebase
- Parallel primitives: for_each, sort, scan, the works
  - Can take arbitrary Thrust pointers and schedule on CPU or GPU
  - **CPU path exists, but requires work to be effective**

genia

Design Goals for Lift

- **Simplify usage of GPU memory containers**
  - Allow pass-by-value containers
  - Implement a model that fits, not an existing model that doesn't
- **Be simple and obvious**
  - Any user should be able to debug problems
- **Enable kernel code sharing between CPU and GPU paths**
  - Abstractions for GPU-specific features…
    - … turn into no-ops on CPU
    - … or provide equivalent implementations

Engineering Goals for Lift

- **Compatibility with existing libraries**
  - Fully compatible with Thrust, CUB, C++ STL, …
  - Makes use of existing libraries to implement parallel primitives
- **Focal point for tracking and testing 3rd party parallel backends**
  - Lift tracks **specific commits** in the CUB, TBB and Thrust trees
  - Test suite aims to validate both Lift as well as the libraries it relies on
  - Focal point for tracking bugs in 3rd party libraries and implementing workarounds
- **Facilitate integrating these libraries in existing source trees**
  - One-line CMake build system brings all this into your project

Example: “context” pattern

Single structure holds working data set

```
struct context {
    vector<int> buffer_a;
    vector<int> buffer_b;
};
```

```
__host__ __device__ void work(context c)
{
    c.buffer_a[threadIdx.x] *= c.buffer_b[5];
}
```

“Context” pattern in Thrust

```
struct context {
    thrust::device_vector<int> buffer_a;
    thrust::device_vector<int> buffer_b;
};
```

```
__host__ __device__ void work(context c)
{
    c.buffer_a[threadIdx.x] *= c.buffer_b[5];
}
```

“Context” pattern in Thrust

```
struct context {
    thrust::device_vector<int> buffer_a;
    thrust::device_vector<int> buffer_b;
};
```

This can only hold GPU data, no CPU path.

context is non-POD! Can’t do this…

```
__host__ __device__ void work(context c)
{
    c.buffer_a[threadIdx.x] *= c.buffer_b[5];
}
```

The NVBIO solution, part 1

```
struct context {
    thrust::device_vector<int> buffer_a;
    thrust::device_vector<int> buffer_b;

    struct view {
        int *buffer_a; size_t buffer_a_len;
        int *buffer_b; size_t buffer_b_len;
    };

    operator view() { … };
};
__host__ __device__ void work(context::view c)
{
    c.buffer_a[threadIdx.x] *= c.buffer_b[5];
}
```

The NVBIO solution, part 1

Still tied to GPU only…

```
struct context {
    thrust::device_vector<int> buffer_a;
    thrust::device_vector<int> buffer_b;

    struct view {
        int *buffer_a; size_t buffer_a_len;
        int *buffer_b; size_t buffer_b_len;
    };

    operator view() { … };
};
__host__ __device__ void work(context::view c)
{
    c.buffer_a[threadIdx.x] *= c.buffer_b[5];
}
```

The NVBIO solution, part 2

```
template <typename target> struct context {
    nvbio_vector<target, int> buffer_a;
    nvbio_vector<target, int> buffer_b;

    struct view {
     typename nvbio_vector<target, int>::view buffer_a;
     typename nvbio_vector<target, int>::view buffer_b;
    };
    operator view() { … };
};
template <typename tgt>
__host__ __device__ void w(typename context<tgt>::view c)
{
    c.buffer_a[threadIdx.x] *= c.buffer_b[5];
}
```

The NVBIO solution, part 2

Forced to reimplement container anyway!

```
template <typename target> struct context {
    nvbio_vector<target, int> buffer_a;
    nvbio_vector<target, int> buffer_b;

    struct view {
     typename nvbio_vector<target, int>::view buffer_a;
     typename nvbio_vector<target, int>::view buffer_b;
    };
    operator view() { … };
};
template <typename tgt>
__host__ __device__ void w(typename context<tgt>::view c)
{
    c.buffer_a[threadIdx.x] *= c.buffer_b[5];
}
```

Need to maintain view code…

Methods from context not available in view and vice versa

“Context” pattern in Lift

```
template <target_system system> struct context {
    allocation<system, int> buffer_a;
    allocation<system, int> buffer_b;
};
```

```
template <target_system system>
__host__ __device__ void work(context<system> c)
{
    c.buffer_a[threadIdx.x] *= c.buffer_b[5];
}
```

“Context” pattern in Lift

```
template <target_system system> struct context {
    allocation<system, int> buffer_a;
    allocation<system, int> buffer_b;
};
```

Containers are handles to memory, guaranteed to be POD

Call-by-value works!

```
template <target_system system>
__host__ __device__ void work(context<system> c)
{
    c.buffer_a[threadIdx.x] *= c.buffer_b[5];
}
```

Closer to C++ object model

Same C++ class holds memory containers and compute code

Memory management identical across CPU and GPU

```
template <target_system system>
struct fasta_database {
    allocation<system, uint8> sequences;
    allocation<system, uint32> sequence_index;

    void resize(int num_reads, int bps_per_read) {
        sequences.resize(num_reads * num_bps);
        sequence_index.resize(num_reads);
    }

    __host__ __device__ uint8 *get_read(int idx) {
      uint32 start = sequence_index[idx];
      return &sequences[start];
    }
    …
};
```

Tying it all together

```
__host__ __device__
uint8 smith_waterman(uint8 *candidate, uint8 *ref)
{ … }

template <target_system system>
struct aligner
{
  fasta_database<system> candidates;
  const pointer<system, uint8> template;
  pointer<system, uint8> out_scores;

  __host__ __device__ void operator(int idx)
  {
    auto read = candidates.get_read(idx);
    out_scores[idx] = smith_waterman(read, template);
  }
};
```

Tying it all together

```
__host__ __device__
uint8 smith_waterman(uint8 *candidate, uint8 *ref)
{ … }

template <target_system system>
struct aligner
{ … }

template <target_system system>
void score(fasta_database<system> candidates,
           pointer<system, uint8> template,
           pointer<system, uint8> output)
{
    parallel<system>::for_each(candidates.size(),
                               aligner<system>(…));
}
`
```

Tying it all together --- Lambda version

```
__host__ __device__
uint8 smith_waterman(uint8 *candidate, uint8 *ref)
{ … }

template <target_system system>
void score(fasta_database<system> candidates,
           pointer<system, uint8> ref,
           pointer<system, uint8> output)
{
    parallel<system>::for_each(
        candidates.size(),
        [=] __host__ __device__ (int idx)
        {
            auto read = candidates.get_read(idx);
            output[idx] = smith_waterman(read, ref);
        }
    );
}
```

Lift Memory Containers

`lift::pointer`: base class for all memory containers

- Similar to a fixed-size `std::vector`
- Adds `peek()`/`poke()` for cross-device reads and writes

`lift::allocation`: “pointer that can reallocate itself”

- Implements `resize()`

`lift::persistent_allocation`: avoids touching the heap on resize

- Implements `reserve()` / `capacity()` + `push_back()`

`lift::scoped_allocation`: special case

- Deallocates when going out of scope (like `std::vector`)
- Can not be copied, but is implicitly convertible to `lift::allocation`

Lift Memory Containers

Lift provides memory containers that pull from a per-GPU allocator:

```
lift::suballocation

lift::persistent_suballocation

lift::scoped_suballocation
```

- Same semantics and API as non-suballocated counterparts
  - GPU memory sourced from a per-GPU suballocator (avoid slow cudaMalloc calls when possible)
  - CPU version relies on malloc
  - Suballocator tuning API still evolving

Lift Parallel Primitives

Roche

Array of "standard" parallel primitives

- Sort, scan, stream compaction, … based on existing libraries
- Optional automatic temporary storage management
- CPU path is considered 1st class citizen

Specialized implementation of for-each

- Iteration pattern tuned for target device
  - Grid stride loop on GPU, contiguous block per thread on CPU
- Can specify launch configuration manually
  - Optional; Lift can compute a suitable configuration
  - No-op for CPU path
    - Plan to refactor API to specify device-specific parameters

genia

Lift Compute Device Abstractions

Roche

Utility functions to abstract GPU-specific features

- `lift::ldg_pointer`: turns any pointer into a read-only LDG version
  - No-op for CPU
- `lift::atomics`: atomic operations
  - Maps directly to CUDA builtins on GPU
  - Provides same feature set on CPU

Compute Device enumeration functions

- Filter GPUs on the system based on requirements
- Detect CPU cache topology and extension flags

genia

Lift Test Harness

Roche

GPU-aware test harness

- One line of code to add a test
- One line of CMake code to build test suite
- "Smart" runtime: skips CUDA tests if no hardware found
  - Suitable for running on Travis CI (or any other automated build)

```
template <target_system system>
void my_test(void) {
  scoped_allocation<system, int> a = { 1, 1 };
  LIFT_TEST_CHECK(parallel<system>::sum(a) == 2);
}
LIFT_TEST_FUNC(my_test_name, my_test);
```

Generates CPU and GPU test!

genia

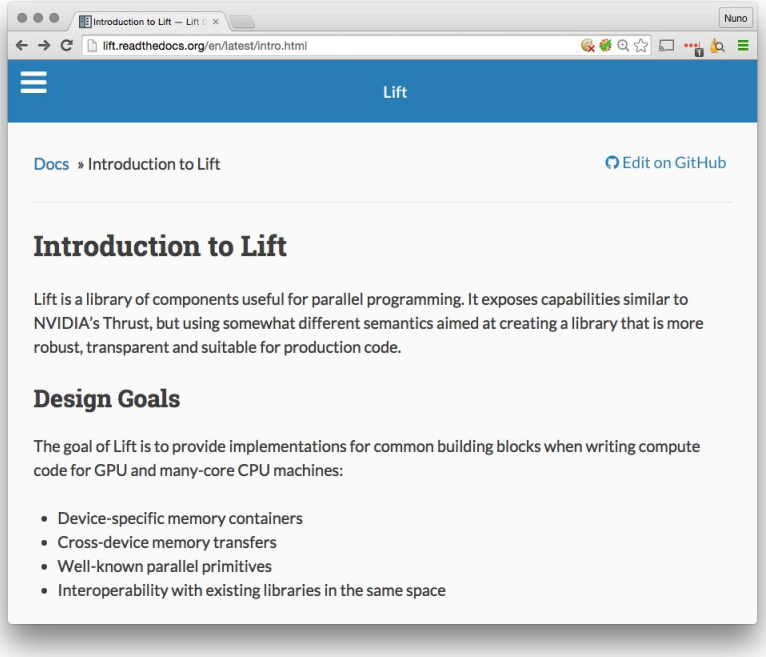
Roadmap for Lift

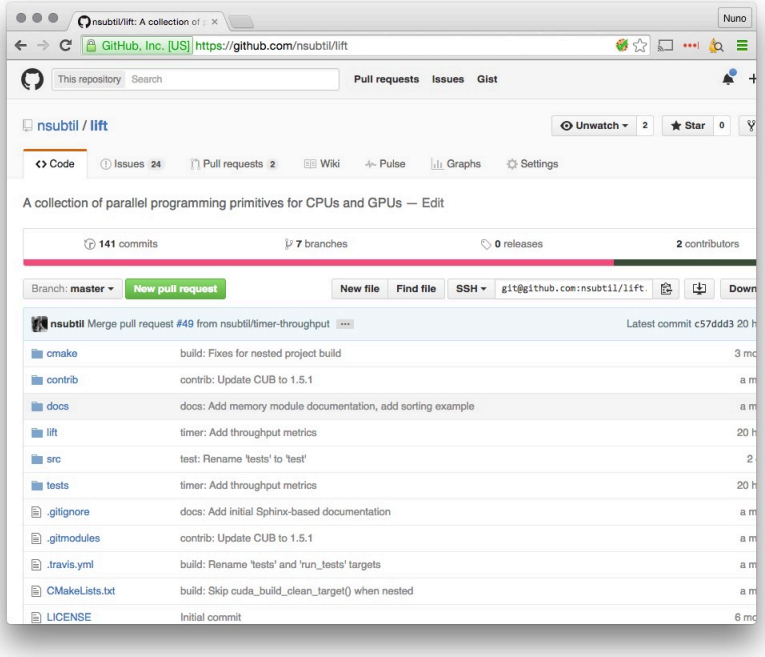
- Abstractions for shared memory, constant memory
- Pinned memory support
- Suballocator hinting / tuning API
- Multi-GPU support
  - Cross-GPU communication primitives
- Asynchronous GPU parallel primitives
  - std::future-like interface: launch work, collect results later
- IBM POWER support
- PCI-E topology detection
- Profiling annotations (NVTX, Intel VTune Tasks, …)

Where to get Lift?

GitHub: https://github.com/nsubtil/lift

Documentation: https://lift.readthedocs.org/





Roche Sequencing

Changing Science. Changing Lives.

Roche Sequencing

Doing now what patients need next