INTRODUCING NVBIO: HIGH PERFORMANCE PRIMITIVES FOR COMPUTATIONAL GENOMICS

Jonathan Cohen, NVIDIA
Nuno Subtil, NVIDIA
Jacopo Pantaleoni, NVIDIA
SEQUENCING AND MOORE’S LAW

- Consistently innovating microarrays and next-generation sequencing at a much faster rate than Moore’s Law
- Elasticity unlocks huge markets like agriculture, cancer, newborn screening and consumer genomics

1. Based on company estimates
GPUS AND MOORE’S LAW
ARE GPUS A FIT FOR BIOINFORMATICS?

Con
- Branchy code, integer-based
- IO intensive
- Many bottlenecks, many pipelines

Pro
- High throughput
- Inner-loops are hungry for bandwidth & compute
- Few common computational patterns (Dynamic programming, text index traversal, branch-and-bound search)
WHY NVBIO?

- 2 year internal NVIDIA effort to answer conclusively yes or no

- **Goals:**
  - Map core computational building blocks to CUDA
  - Find programming idioms to express parallelism appropriately
  - Understand performance at socket, node, and system level
  - Deliver high-quality open source library for community to build on
  - Produce usable tools for non-developers, comparable with current state-of-the-art
OVERVIEW

- What is NVBIO
- Core concepts
- NVBowtie2
- Tutorial
INTRODUCING NVBIO

- **Open source** (GPL v2) C++ framework for sequence analysis
- Designed for **heterogeneous** computing: CPU + GPU
- **Scalable** computational building blocks for **large** datasets
- Allow programmer to **express and exploit parallelism** easily
- **Ready-to-use** tools (nvBowtie2, nvSetBwt)
HETEROGENEOUS LIBRARY

1k-10k

Text Indices
- FM-index
- Suffix Trie
- Radix Tree
- Sorted Dictionary

DP Alignment
- Edit Distance
- Smith-Waterman
- Needleman-Wunsch
- Gotoh
- Banded/Full DP

Text Search
- Exact Search
- Backtracking

Sequence I/O
- FASTQ
- FASTA
- BWT Index

Alignment I/O
- SAM
- BAM
- CRAM (wip)

Support Tools
- HTML report generators

10-100

GPU

CPU
BUILDING BLOCK: FM-INDEX

Original text
- CAACATGTGAC
- AGATTTAGACCA
- CAGATTTAGGTT
- GTGTACAAACCA
- GATTAGACACAA
- AGTTGTGA$

Burrows-Wheeler Transform
- GACCTAGCGAA
- GCTTTCCCC$G
- CGGACAAACAAC
- AAATAAATAATAA
- TATTGGTTTTTGA
- GGGTAGGA

FM-Index

Q=CTTAGAAC
Found at location 213321

Fast O(|Q|) query

O(1) Incremental Search:
Given set of substrings S,
find subset which start with X
BWT CONSTRUCTION BENCHMARK (hg19 - GRCh37)

CPU: i7 3930K 3.2Ghz
GPU: K40
STRING SEARCH BENCHMARK

- **Exact Search**
  - i7 3930K 3.2Ghz: 19.80 M queries/s
  - K40: 123.30 M queries/s

- **Approximate Search**
  - i7 3930K 3.2Ghz: 1.45 M queries/s
  - K40: 6.33 M queries/s

Reads: SRR493095
Genome: hg19
BUILDING BLOCK: DYNAMIC PROGRAMMING (SMITH WATERMAN, ETC.)

<table>
<thead>
<tr>
<th></th>
<th>_</th>
<th>C</th>
<th>C</th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>_</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
<td>-6</td>
</tr>
<tr>
<td>C</td>
<td>-1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
</tr>
<tr>
<td>A</td>
<td>-2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>-3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>T</td>
<td>-4</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>G</td>
<td>-5</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>G</td>
<td>-6</td>
<td>-3</td>
<td>-3</td>
<td>-1</td>
<td>2</td>
<td>5</td>
<td>8</td>
</tr>
</tbody>
</table>

Match = +2  
Mismatch = -1  
Delete = -1  
Insert = -1

CCATTG_
C_ATTGG
SMITH-WATERMAN BENCHMARK

GCUPS

Smith-Waterman

Edit Distance

* SSW library – 12 threads
void process(int i)
{
    State state(i);
    while (state.is_done() == false) // taken 30% of the times
    {
        if (state.A_flag) // taken 50% of the times
        {
            if (state.B_flag) // taken 33% of the times
                AB( state ); // => 5% utilization!
            else
                A( state ); // => 10% utilization!
        }
        else if (state.B_flag) // taken 50% of the times
            B( state ); // => 15% utilization!
        if (state.C_flag) // taken 33% of the times
            C( state ); // => 10% utilization!
    }
}
while (...)
BUILDING BLOCK: PIPELINE PARALLELISM

__host__ bool pipeline() {
    // pipeline scheduler - CPU
    if (while_q.size() > thresh) while_stage<<<while_q.size()>>>();
    if (AB_q.size() > thresh) AB_stage<<<AB_q.size()>>>() // etc.
    return while_q.empty() && AB.empty() && ...;
}

__global__ void while_stage() {
    // primary stage - GPU
    const int tid = thread_id(); // thread id
    if (tid >= in_queue.size()) return;

    const State state = in_queue[tid]; // fetch work from input queue
    if (state.A_flag)
    {
        if (state.B_flag) AB_queue.push( state );
        else A_queue.push( state );
    }
    else if (state.B_flag) B_queue.push( state );
    else if (state.C_flag) C_queue.push( state );
}
NVBOWTIE2: RE-ENGINEERING BOWTIE2 ON NVBIO

- Implement Bowtie2 algorithm from scratch on NVBIO

- Supports full spectrum of features

- First achieve same accuracy as Bowtie2, then worry about speed
NVBOWTIE2 - Architecture Overview

- CPU background input thread
- Prefetch read batches
- Decompress / reformat
  - CPU
  - CPU

Ran on GPU

- FM INDEX
- Seed map
- DP ALIGN
- Score
  - Yes
    - Finished?
      - Yes
        - Reseed?
          - Yes
          - Yes
          - Yes
          - No
        - No
      - No
    - No
  - No

- Traceback

CPU background output thread

- Consume alignment batches
  - CPU
  - CPU

- Reformat / compress
  - CPU
  - CPU

- SAM / BAM

- FASTQ (GZIP)

Runs on GPU

- CPU
  - CPU
  - CPU
  - CPU

Yes

No
### NVBOWTIE2 - RESULTS ON REAL DATASETS

<table>
<thead>
<tr>
<th>Dataset Description</th>
<th>Speedup</th>
<th>Alignment Rate</th>
<th>Disagreement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ion Proton 100M x 175bp (8-350) local</strong></td>
<td>8.4x</td>
<td>-0.6%</td>
<td>0.03%</td>
</tr>
<tr>
<td><strong>Illumina HiSeq 2000 10M x 100bp x 2 local</strong></td>
<td>2.8x</td>
<td>=</td>
<td>0.022%</td>
</tr>
<tr>
<td><strong>Illumina HiSeq 2000 10M x 100bp x 2 end-to-end</strong></td>
<td>3.4x</td>
<td>=</td>
<td>0.006%</td>
</tr>
</tbody>
</table>

**CPU:** Core i7 3930K 3.2Ghz  
**GPU:** K40  
**Alignment rate:** % difference in number of aligned reads (MAPQ >=32)  
**Disagreement:** % reads aligned to different location (MAPQ>=32)
NVBIO TUTORIAL
Nuno Subtil
OUTLINE

- Problem statement

- Architecting an aligner for the GPU
  - Batching
  - Lessons learned from nvBowtie

- How to implement using nvbio
  - I/O
  - FM index lookups
  - DP alignment
PROBLEM STATEMENT

- Implement a simple proto-alignment pipeline
  - Input: human genome, set of short (~300bp) reads
  - Output: a single genome coordinate per read

- Two-stage approach:
  - Seed using the initial 20bps via FM-index lookups
  - Use dynamic programming to extend the first seed we find
BATCHING FOR GPUS

- Classic CPU approach: drive one read through the entire pipeline, move on to the next

```c
#pragma omp parallel for
for(c = 0; c < num_reads; c++) {
    uint32 bwt_loc = fmi(fmindex, reads[c]);
    uint32 genome_loc = locate(fmindex, bwt_loc);
    uint32 score = sw(genome_loc, width, reads[c]);
}
```
GPUs benefit from a batched approach

```c
for(c = 0; c < num_reads; c++)
    bwt_loc[c] = fmi(fmindex, reads[c]);

for(c = 0; c < num_reads; c++)
    genome_loc[c] = locate(fmindex, bwt_loc[c]);

for(c = 0; c < num_reads; c++)
    score = sw(genome_loc[c], width, reads[c]);
```
BATCHING FOR GPUS

- GPUs benefit from a batched approach

```c
for(c = 0; c < num_reads; c++)
    bwt_loc[c] = fmi(fmindex, reads[c]);

for(c = 0; c < num_reads; c++)
    genome_loc[c] = locate(fmindex, bwt_loc[c]);

for(c = 0; c < num_reads; c++)
    score = sw(genome_loc[c], width, reads[c]);
```
GPUs benefit from a batched approach

\[ \text{fmi} \langle\langle T, B \rangle\rangle(\text{bwt\_loc, fmindex, reads}); \]

\[ \text{genome} \langle\langle T, B \rangle\rangle(\text{genome\_loc, fmindex, bwt\_loc}); \]

\[ \text{sw} \langle\langle T, B \rangle\rangle(\text{score, genome\_loc, width, reads}); \]
ARCHITECTURE

- Consume reads in large batches

- Keep track of a “pipeline context”
  - Intermediate buffers go here
  - Memory requirements determined by batch size
  - Avoid malloc/free during alignment (allocate memory up front)

- Isolate each stage clearly
  - Input: well-defined subset of pipeline state (+ read data from fastq)
  - Well-defined output
  - Allow for memory to be reused between stages
Intermediate buffers

FM index

Genome

context

Seeding

Select + Coordinate Xform

Extension

in.fastq

Parser

Output

GPU

CPU
struct pipeline_state {
    // results of the seeding round
    uint2 *seeds;
    // results of the locate round
    uint32 *locations;
    // results of the scoring round
    alignment_sink *sinks;
    // temporary storage for the scoring round
    void *temp;
};

io::ReadData *file_data;
while((file_data = next_batch_host()) != NULL) {
    // copy the file data to the GPU
    io::ReadDataCUDA *file_data_device = new io::ReadDataCUDA(*file_data);
    // run the seeding round on the GPU
    seed_gpu(&pipeline_ctx_gpu, file_data_device);
    locate_gpu(&pipeline_ctx_gpu, file_data_device);
    extend_gpu(&pipeline_ctx_gpu, file_data_device);
    delete file_data_device;
    delete file_data;
}
static nvio::io::ReadStream *read_stream = NULL;

void open_input_file(const char *fname)
{
    read_stream = nvio::io::open_read_file(fname, nvio::io::Phred33);
    if (read_stream == NULL)
    {
        fprintf(stderr, "error opening input file \%s\n", fname);
        exit(1);
    }
}

nvbio::io::ReadStream *next_batch(void)
{
    return read_stream->next(READ_BATCH_SIZE);
}
ACCESSING READ DATA

```c
// our packed data stream type for reads:
// packs uint8 symbols into 4 bits (i.e., a 16-letter alphabet)
typedef nvbio::io::ReadData::read_string packed_read;

nvbio::io::ReadData *data = next_batch();
for(int c = 0; c < data->m_n_reads; c++)
{
    packed_read r = data->get_read(c);
    for(int bp = 0; bp < r.size(); bp++)
    {
        if (r[bp] == 0) printf("A");
        if (r[bp] == 1) printf("C");
        if (r[bp] == 2) printf("G");
        if (r[bp] == 3) printf("T");
    }
}
```
CREATING AND LOADING THE FM-INDEX

- NVbio includes offline utilities to generate the FM-index
  - Same code is included in the library if you want to do it online
  - FM-index also encapsulates packed genome data and SSA

- FM-index interface can load the FM-index data from disk

- Also supported: placing the index in system shared memory
  - Index stays in memory across runs of your application
  - Eliminates most startup costs
  - Different class, same interface --- easy!
NVBIO: LOADING FM-INDEX DATA

// the data for our FM index, host-side
nvbio::io::FMIndexData *fmindex_data_host;

nvbio::io::FMIndexDataMMAP *mmap_loader = new nvbio::io::FMIndexDataMMAP();
ret = mmap_loader->load("human.NCBI36");
if (ret != 0) { // load succeeded
    fmindex_data_host = mmap_loader;
} else {
    nvbio::io::FMIndexDataRAM *file_loader = new nvbio::io::FMIndexDataRAM();
    ret = file_loader->load("/home/nsubtil/ncbi36/human.NCBI36");
    if (ret != 0) { // load succeeded
        fmindex_data_host = file_loader;
    } else {
        fprintf(stderr, "failed to load genome!\n");
        exit(1);
    }
}

// create the device copy of the FM index
// this will copy all of the FM index data to the GPU
nvbio::io::FMIndexDataCUDA *fmindex_data_device = new nvbio::io::FMIndexDataCUDA(*fmindex_data_host);
QUERYING THE FM-INDEX

- `uint2 nvgio::match(fmindex, pattern, length)`
  - Queries `fmindex` for pattern
  - Returns a BWT range (NOT genome coordinates)

```c
void seed_cpu(struct pipeline_state *context, nvgio::io::ReadData *data) {
    nvgio::io::FMIndexData::fm_index_type fmindex = fmindex_data_host->index();
    for(int c = 0; c < data->m_n_reads; c++) {
        packed_read r = data->get_read(c);
        context->seeds[c] = nvgio::match(fmindex, r, 20);
    }
}
```
__global__
void seed_kernel(struct pipeline_state context,
                 nvbio::io::FMIndexDataCUDA::fm_index_type fmindex,
                 nvbio::io::ReadData::ReadDataBase data)
{
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    while(tid < data.m_n_reads)
    {
        packed_read r = data.get_read(tid);
        context.seeds[tid] = nvbio::match(fmindex, r, 20);
        tid += blockDim.x * gridDim.x;
    }
}

void seed_gpu(struct pipeline_state *context, nvbio::io::ReadDataCUDA *data)
{
    nvbio::io::FMIndexDataCUDA::fm_index_type fmindex = fmindex_data_device->index();
    seed_kernel<<<256,256>>>(*context, fmindex, *data);
}
GENOME COORDINATE TRANSFORM

- FM-index returns BWT coordinates
- nvbio::locate(fmindex, bwt_coord)
  - Input: FM-index, BWT coordinate
  - Output: corresponding genome coordinate

```c
void locate_cpu(struct pipeline_state *context, nvbio::io::ReadData *data)
{
    nvbio::io::FMIndexData::fm_index_type fmindex = fmindex_data_host->index();
    for(int c = 0; c < data->m_n_reads; c++)
    {
        context->locations[c] = nvbio::locate(fmindex, context->seeds[c].x);
    }
}
```
GENOME COORDINATE TRANSFORM

```c
__global__
void locate_kernel(struct pipeline_state context,
                   nvbio::io::FMIndexDataCUDA::fm_index_type fmindex,
                   nvbio::io::ReadData::ReadDataBase data)
{
    int tid = threadIdx.x + blockIdx.x * blockDim.x;

    while(tid < data.m_n_reads)
    {
        context.locations[tid] = nvbio::locate(fmindex, context.seeds[tid].x);
        tid += blockDim.x * gridDim.x;
    }
}

void locate_gpu(struct pipeline_state *context, nvbio::io::ReadDataCUDA *data)
{
    nvbio::io::FMIndexDataCUDA::fm_index_type fmindex = fmindex_data_device->index();
    locate_kernel<<<256,256>>>(*context, fmindex, *data);
}
```
nvbio supports several different DP aligners
- \{ \text{Edit-distance, Smith-Waterman, Gotoh} \} \times \{ \text{local, semi-global, global} \} \times \{ \text{banded, full matrix} \} \times \{ \text{score, backtrack} \}

Template library for DP alignment
- Configured entirely at compile time
WORKFLOW FOR DP ALIGNMENT

- Configure alignment via typedefs
  - Alignment parameters are template parameters

- Create temporary storage

- Call nvbio::aln::alignment_score()
SETTING UP LOCAL GOTOH ALIGNMENT

Accumulates alignment results (score, position, …)
Can be implemented by client application

```cpp
// alignment "sinks" keep track of the best scores for DP alignment in some (user-modifiable) way
// this particular sink just keeps track of the absolute best score in the DP matrix
// the template parameter contains the data type for the score
typedef nvbio::aln::BestSink<int32> alignment_sink;

// we're going to use Gotoh local alignment with a constant scoring scheme
typedef nvbio::aln::GotohAligner<nvbio::aln::LOCAL, nvbio::aln::SimpleGotohScheme> aligner_type;
```

Alignment object type configured for local Gotoh alignment
STRING ALIGNMENT FUNCTION

```c
__device__ __host__
void align_string(alignment_sink *output,
                 packed_genome_window reference, packed_read query, cell_type *temp)
{
    // set up our scoring object
    // (parameters are match bonus + mismatch, gap open and gap extension penalties)
    const nvhbio::aln::SimpleGotohScheme scoring(2, -1, -1, -1);

    // align!
    alignment_sink sink;
    bool ret = nvhbio::aln::alignment_score(aligner_type(scoring),
                                           query,
                                           aln::trivial_quality_string(),
                                           reference,
                                           -255,  // minimum accepted score
                                           *output,
                                           temp);

    if (!ret)
        output->sink.x = (unsigned int) -1;
}
```
IT WORKS!!
WHAT ELSE?

- DP back-tracing
- Fast BWT builder

- Data movement between CPU and GPU
  - Views vs containers
- GPU work queues
  - Flexible scheduling: per thread, per warp, per block
- Many useful primitives
  - Queues, heaps, string sets, ...
WHERE TO GET

- Google Group: nvbio-users