Algorithms and Tools for Bioinformatics on GPUs

Bertil SCHMIDT
Contents

• Motivation
• Pairwise Sequence Alignment
• Multiple Sequence Alignment
• Short Read Error Correction using CUDA
• Some other CUDA-enabled Bioinformatics tools
Data Explosion

Growth Rate of GenBank and UniProtKB/TrEMBL

- GenBank Base Pairs
- GenBank Entries
- UniProtKB/TrEMBL Amino Acids
- UniProtKB/TrEMBL Entries

Typical Bioinformatics Applications

• Database Search
  – BLAST, Smith-Waterman
• Multiple Sequence Alignment
  – ClustalW
• Hidden Markov Models
  – HMMer
• Motive Finding
  – MEME
• De-novo Genome Assembly
  – Velvet
• Phylogenetic Trees
  – RAxML
• Protein Structure Prediction
Local Pairwise Sequence Alignment

Align \( S_1 = \text{ATCTCGTATGATG} \quad S_2 = \text{GTCTATCAC} \)

\[
Sbt(x, y) = \begin{cases} 
2 & \text{if } (x = y) \\
-1 & \text{else}
\end{cases}
\]

\( \alpha = 1 \), \( \beta = 1 \)

\[
H(i, j) = \max \begin{cases} 
0 \\
H(i-1, j) - 1 \\
H(i, j-1) - 1 \\
H(i-1, j-1) + Sbt(S_1_i, S_2_j)
\end{cases}
\]
Extraction of Parallelism


- Liu, Maskell, Schmidt: "CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units", **BMC Research Notes, 2:73, 2009**
Multiple Sequence Alignment (MSA)

- Exact DP solution has exponential complexity
- Heuristic optimization methods used in practice; e.g. progressive alignment method (*ClustalW*)

Profiling the three stages of ClustalW

- Stage 1 (*Distance matrix*) requires more than 90% of overall runtime
**MSA with ClustalW**

<table>
<thead>
<tr>
<th></th>
<th>HAHU</th>
<th>HBU</th>
<th>HAHO</th>
<th>HBUO</th>
<th>MYWH</th>
<th>PI LH</th>
<th>LGHB</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAHU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HBHU</td>
<td>21.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HAHO</td>
<td>32.9</td>
<td>19.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HBUO</td>
<td>20.7</td>
<td>39.0</td>
<td>20.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MYWH</td>
<td>11.0</td>
<td>9.8</td>
<td>10.3</td>
<td>9.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PI LH</td>
<td>9.3</td>
<td>8.6</td>
<td>9.6</td>
<td>8.4</td>
<td>7.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LGHB</td>
<td>7.1</td>
<td>7.3</td>
<td>7.5</td>
<td>7.4</td>
<td>7.3</td>
<td>4.3</td>
<td></td>
</tr>
</tbody>
</table>

**Dist Matrix:**

\[ O(n^2 \cdot l^2) \]

**NJ-Tree:**

\[ O(n^3) \]

**Dynamic Programming:**

- Alignments:
  - HBHU
  - HBUO

**Progressive:**

\[ O(n \cdot l^2) \]

- Alignments:
  - HBHU
  - HBUO
  - HAHU
  - HAHO

---

**NANYANG TECHNOLOGICAL UNIVERSITY**
## Distance Computation

- \(sbt(x,y) = +2 \) if \(x=y\); \(-1\) otherwise
- **Linear Gap Penalty**: \(g = -1\)

\[
H(i, j) = \max\begin{cases} 
0 \\
H(i-1, j) - g \\
H(i, j-1) - g \\
H(i-1, j-1) + sbt(S_1, S_2) 
\end{cases}
\]

<table>
<thead>
<tr>
<th></th>
<th>(\emptyset)</th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>T</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\emptyset)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

- \(nid(S_1, S_2) = 6 \Rightarrow d(S_1, S_2) = 1/3\)
- Disadvantage of \(nid\)-Implementation with trace-back: *Quadratic Memory or Divide-and-Conquer*
Our Approach: Distance Computation with DP in linear space

<table>
<thead>
<tr>
<th></th>
<th>(\emptyset)</th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>T</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>G</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(\emptyset)</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
</tr>
<tr>
<td>G</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>2/1</td>
<td>1/1</td>
<td>0/0</td>
<td>0/0</td>
<td>2/1</td>
<td>1/1</td>
<td>0/0</td>
</tr>
<tr>
<td>T</td>
<td>0/0</td>
<td>0/0</td>
<td>2/1</td>
<td>1/1</td>
<td>2/1</td>
<td>1/1</td>
<td>1/1</td>
<td>4/2</td>
<td>3/2</td>
<td>2/1</td>
<td>1/1</td>
<td>1/1</td>
</tr>
<tr>
<td>C</td>
<td>0/0</td>
<td>0/0</td>
<td>1/1</td>
<td>4/2</td>
<td>3/2</td>
<td>4/2</td>
<td>3/2</td>
<td>3/2</td>
<td>3/2</td>
<td>2/2</td>
<td>1/1</td>
<td>0/0</td>
</tr>
<tr>
<td>T</td>
<td>0/0</td>
<td>0/0</td>
<td>2/1</td>
<td>3/2</td>
<td>6/3</td>
<td>5/3</td>
<td>4/3</td>
<td>5/3</td>
<td>4/3</td>
<td>5/3</td>
<td>4/3</td>
<td>3/3</td>
</tr>
<tr>
<td>A</td>
<td>0/0</td>
<td>2/1</td>
<td>2/1</td>
<td>2/2</td>
<td>5/3</td>
<td>5/3</td>
<td>4/3</td>
<td>4/3</td>
<td>7/4</td>
<td>6/4</td>
<td>5/4</td>
<td>6/4</td>
</tr>
<tr>
<td>T</td>
<td>0/0</td>
<td>1/1</td>
<td>4/2</td>
<td>3/2</td>
<td>4/3</td>
<td>4/3</td>
<td>4/3</td>
<td>6/4</td>
<td>5/4</td>
<td>9/5</td>
<td>8/5</td>
<td>7/5</td>
</tr>
<tr>
<td>T</td>
<td>0/0</td>
<td>0/0</td>
<td>3/2</td>
<td>6/3</td>
<td>5/3</td>
<td>6/4</td>
<td>5/4</td>
<td>5/4</td>
<td>5/4</td>
<td>8/5</td>
<td>8/5</td>
<td>7/5</td>
</tr>
<tr>
<td>C</td>
<td>0/0</td>
<td>2/1</td>
<td>2/2</td>
<td>5/3</td>
<td>5/3</td>
<td>5/4</td>
<td>5/4</td>
<td>4/4</td>
<td>7/5</td>
<td>7/5</td>
<td>7/5</td>
<td>7/5</td>
</tr>
<tr>
<td>C</td>
<td>0/0</td>
<td>1/1</td>
<td>1/1</td>
<td>4/3</td>
<td>4/3</td>
<td>7/4</td>
<td>6/4</td>
<td>5/4</td>
<td>6/5</td>
<td>6/5</td>
<td>6/5</td>
<td>6/5</td>
</tr>
</tbody>
</table>
Recurrence Relations for Affine Gap Penalties

\[ H_A(i, j) = \max\{0, E(i, j), F(i, j), H_A(i - 1, j - 1) + sbt(S_1[i], S_2[j])\} \]

\[ E(i, j) = \max\{H_A(i, j - 1) - \alpha, E(i, j - 1) - \beta\} \]

\[ F(i, j) = \max\{H_A(i - 1, j) - \alpha, F(i - 1, j) - \beta\} \]

\[
N_A(i, j) = \begin{cases} 
0, & \text{if } H_A(i, j) = 0 \\
N_A(i - 1, j - 1) + m(i, j), & \text{if } H_A(i, j) = H_A(i - 1, j - 1) + sbt(S_1[i], S_2[j]) \\
N_E(i, j), & \text{if } H_A(i, j) = E(i, j) \\
N_F(i, j); & \text{if } H_A(i, j) = F(i, j)
\end{cases}
\]

\[
N_E(i, j) = \begin{cases} 
0, & \text{if } j = 1 \\
N_A(i, j - 1), & \text{if } E(i, j) = H_A(i, j - 1) - \alpha \\
N_E(i, j - 1); & \text{if } E(i, j) = E(i, j - 1) - \beta
\end{cases}
\]

\[
N_F(i, j) = \begin{cases} 
0, & \text{if } i = 1 \\
N_A(i - 1, j), & \text{if } F(i, j) = H_A(i - 1, j) - \alpha \\
N_F(i - 1, j); & \text{if } F(i, j) = F(i - 1, j) - \beta
\end{cases}
\]
CUDA Parallelization

- **Inter-task parallelization**
  - Each alignment (task) is assigned to exactly one thread
  - $dimBlock$ alignments are performed in parallel within a thread block.

- **Load balancing**
  - Sequences sorted by lengths $\Rightarrow$ all threads within a thread block have similar workload

- **Memory access**
  - $O(\min\{l_a,l_b\})$ storage for intermediate results per thread
  - Stored in global memory using coalesced memory access pattern
  - Partitioning of DP-matrix into blocks $\Rightarrow$ reduces global memory access by using shared memory and registers
  - Substitution matrix stored in shared memory
  - Sequences stored in texture memory
Overview: Parallelization Approaches

- Sequential ClustalW Algorithm
  - DP-Modification to allow more efficient parallelization

- FPGA:
  - Systolization with Verilog HDL
- GPU:
  - SIMT Parallelization with CUDA
- FPGA:
  - Load Balancing, Partitioning (FIFO, Multi-Lanes)
- GPU:
  - Load Balancing, Optimization of memory accesses (coalesced, shared)
- Cell/BE:
  - MIMD Parallelization with Cell/BE SDK
- Cell/BE:
  - Load Balancing, SIMD Vectorization
**Comparison: Speedup**

- **FPGA:** Xilinx XC5VLX330
  - 416PEs
  - 65MHz
- **GPU:** GeForce GTX 280
  - 240SPs
  - 1.3GHz
- **Cell/BE:** PlayStation3
  - 6SPEs
  - SIMD vector-length:8
  - 3.2GHz

![Speedup compared to ClustalW 2.0.9](chart.png)
Comparison: Productivity, Power

MCUPS per LOC

MCUPS per Watt
**MSA-CUDA: Performance Stage 2 + 3**

- **MSA-CUDA on a GPU (GeForce GTX280)**
  - Better Performance than ClustalW-MPI on a PC-cluster with 32 Cores for all tested datasets
Profile Hidden Markov Models

- Statistical model of a set of biological similar sequences
- HMMs can be created from MSAs
- General transition structure for a *global alignments*:

![Diagram of Profile Hidden Markov Models]

- **Viterbi Algorithm:**
  - Finds most likely path through the HMM generating the given sequence
  - Dynamic Programming: $O(\text{sequence length} \times \text{HMM length})$
- Efficient parallel implementation (similar to SW)
- Transition structure does not allow for *local or multi-hit alignments*
Improving MSA accuracy

- We have designed MSAProbs to significantly improve alignment accuracy
- Currently working on a CUDA version

### Table 3. Overall mean SPS and CS scores and runtime on BAliBASE 3.0

<table>
<thead>
<tr>
<th>Aligner</th>
<th>SPS</th>
<th>CS</th>
<th>Time (hh:mm:ss)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSAProbs</td>
<td>89.09</td>
<td>64.51</td>
<td>1:12:56</td>
</tr>
<tr>
<td>MUSCLE</td>
<td>84.33</td>
<td>53.17</td>
<td>0:16:11</td>
</tr>
<tr>
<td>MAFFT</td>
<td>87.50</td>
<td>61.07</td>
<td>0:41:05</td>
</tr>
<tr>
<td>Probalign</td>
<td>87.78</td>
<td>60.68</td>
<td>8:05:35</td>
</tr>
<tr>
<td>ProbCons</td>
<td>88.31</td>
<td>61.89</td>
<td>5:29:15</td>
</tr>
<tr>
<td>ClustalW</td>
<td>78.65</td>
<td>44.75</td>
<td>0:18:56</td>
</tr>
</tbody>
</table>
Short Read Error Correction

DNA → Sequencing machine → Read-sequences → DNA-sequence

May contain errors!
Next-Generation Sequencing (NGS)

- NGS technology has
  - ultra-high throughput
  - short read-length

<table>
<thead>
<tr>
<th>NGS Platform</th>
<th>Illumina (HiSeq2000)</th>
<th>SOLiD 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reads per run</td>
<td>1 billion</td>
<td>1.4 billion</td>
</tr>
<tr>
<td>Read length</td>
<td>35-100 bps</td>
<td>35-50 bps</td>
</tr>
<tr>
<td>Run time (single-end)</td>
<td>2-4 days</td>
<td>4-8 days</td>
</tr>
<tr>
<td>Run time (mate-pair)</td>
<td>4-8 days</td>
<td>8-16 days</td>
</tr>
</tbody>
</table>

- Example: Human Genome Sequencing (Li et al., 2010)
  - 4 billion reads, average read-length 53bp (71x coverage)

- NGS Bioinformatics Challenges
  - Scalability (to deal with huge amounts of reads)
  - Algorithm design (to deal with short reads)
**Error Correction**

- Illumina reads contain about 1%-2% sequencing errors (substitutions)
- Prior Error Correction can improve assembly quality as well as Graph size
  - N50-values produced by Edena for simulated reads (1.1M reads of length 35 generated from *Saccharomyces C Chr 5*)
  - Human Genome (Li et al. 2010): distinct 25-mers reduced from 14.6B to 5.0B through prior error correction
- Error Correction usually most time-consuming step in assembly
  - 50% of total runtime (24h on a high-end PC cluster) for human genome
- Error Correction also increases the amount of mapped reads to the reference genome for Re-sequencing applications
Spectral Alignment Problem (SAP)

Read:

\[
\begin{array}{cccccccc}
T & T & G & T & C & A & G & C & G & T & A \\
\end{array}
\]

\[L=11\]
\[l=4\]
error

\(l\)-mer Spectrum = \{…, TCA\_A, CA\_A\_C, A\_A\_CG, A\_CGT, …\}

- Changing the single error at position 6 in the given read from \textbf{G} to \textbf{A} results in \(l\) corresponding matches in the spectrum.
Spectral Alignment Problem (SAP)

- **Reads** $R=\{r_1, \ldots, r_k\}$ of length $L$
- **Length** $l<L$
- **Reference genome** $G$
- **$l$-mer spectrum** $T(G)$
- read error-free
  - $\Rightarrow$ all its $l$-mers have corresponding exact match in $T(G)$
- read has a single mutation error
  - overlapping $l$-mers have (in most cases) a low number of corresponding exact matches in $T(G)$
  - Changing error to correct base-pair, all overlapping $l$-mers have a corresponding match in $T(G)$
Spectral Alignment Problem (SAP)

- $G$ usually unknown $\Rightarrow T(G)$ approximated by $T(R,m)$
- $T(R,m)$: all $l$-mers that occur at least $m$ times (multiplicity) in $R$
- Possible complications
  1. Some $l$-mers in $T(G)$ not necessarily in $T(R,m)$ (false negatives)
  2. Some $l$-mers in $T(R,m)$ not necessarily in $T(G)$ (false positives)
- Quality of approximation can be reduced by using quality-scores associated with each read (i.e. Use only high-quality $l$-mers to build $T(m,R)$)
  - Q-score: value between 0 and 40
Bloom Filter Data Structure

- Membership test most important operation (test if an l-mer is in $T(m,R)$)
- Use of a space-efficient Bloom filter for probabilistic hashing stored in CUDA texture memory
SAP Voting Procedure

Read $r_i$: T T G T C A G C G T A

$l=4$ Error at position $pos$

$l$-mer Spectrum = {..., TCAA, CAAC, AACG, ACAT, ...}

Voting matrix $V(r_i)[[]][]$: 

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Corrected Read $r_i[6][A]$: T T G T C A A C G T A
DecGPU error correction algorithm

1. (Distributed) spectrum construction
2. filtering out error-free reads
3. fixing erroneous reads using a voting algorithm
4. trimming (or discarding entirely) the fixed reads that remain erroneous
5. optional iterative policy between the filtering and fixing stages for the correction of more than one base error in a single read
6. On a cluster DecGPU uses a one-to-one correspondence between an MPI process and one GPU
### Performance Evaluation – Datasets

**Table 1. Simulated and real short read datasets**

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Read length</th>
<th>Coverage</th>
<th>Error rate</th>
<th>No. of Reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECOLI30X1.5</td>
<td>36</td>
<td>30</td>
<td>1.5%</td>
<td>3866000</td>
</tr>
<tr>
<td>ECOLI30X3.0</td>
<td>36</td>
<td>30</td>
<td>3.0%</td>
<td>3860000</td>
</tr>
<tr>
<td>ECOLI75X1.5</td>
<td>36</td>
<td>75</td>
<td>1.5%</td>
<td>9666000</td>
</tr>
<tr>
<td>ECOLI75X3.0</td>
<td>36</td>
<td>75</td>
<td>3.0%</td>
<td>9666000</td>
</tr>
<tr>
<td>ECOLI150X1.5</td>
<td>72</td>
<td>150</td>
<td>1.5%</td>
<td>9666000</td>
</tr>
<tr>
<td>ECOLI150X3.0</td>
<td>72</td>
<td>150</td>
<td>3.0%</td>
<td>9666000</td>
</tr>
<tr>
<td>SRR006331</td>
<td>36</td>
<td>69</td>
<td>-</td>
<td>1693848</td>
</tr>
<tr>
<td>SRR016146</td>
<td>51</td>
<td>81</td>
<td>-</td>
<td>4438066</td>
</tr>
<tr>
<td>SRR001665</td>
<td>36</td>
<td>162</td>
<td>-</td>
<td>20816448</td>
</tr>
</tbody>
</table>
Runtime Comparison

- **Hardware Configurations**
  - DecGPU: single Tesla T10
  - DecGPU (CPU version): AMD Opteron 2378 quad-core (multi-threaded with OpenMP)
  - Euler-SR: Intel Core i7 (single threaded)

- **All times in seconds**

<table>
<thead>
<tr>
<th>Data set</th>
<th>Euler-SR</th>
<th>DecGPU (CPU)</th>
<th>DecGPU (GPU)</th>
<th>Overall Speedup</th>
<th>Overall Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SC</td>
<td>EC</td>
<td>SC</td>
<td>SC</td>
<td>Euler-SR</td>
</tr>
<tr>
<td>A</td>
<td>61</td>
<td>671</td>
<td>21</td>
<td>35</td>
<td>16</td>
</tr>
<tr>
<td>B</td>
<td>338</td>
<td>1524</td>
<td>119</td>
<td>196</td>
<td>93</td>
</tr>
<tr>
<td>C</td>
<td>746</td>
<td>7016</td>
<td>289</td>
<td>269</td>
<td>222</td>
</tr>
</tbody>
</table>
Accuracy and Runtime Comparison

• Comparison to hSHREC
  – Salmela (Bioinformatics, 2010) – extension of the suffix tree approach by Schroeder and Schmidt (Bioinformatics, 2009)
  – hSHREC and DecGPU both run a dual quad-core Intel Xeon E5506 (both multithreaded)

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Original Error Rate (%)</th>
<th>Corrected Error Rate (%)</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>DecGPU one fixing</td>
<td>DecGPU two fixing</td>
</tr>
<tr>
<td>ECOLI30X1.5</td>
<td>1.498</td>
<td>0.426</td>
<td>0.341</td>
</tr>
<tr>
<td>ECOLI30X3.0</td>
<td>3.003</td>
<td>1.773</td>
<td>1.625</td>
</tr>
<tr>
<td>ECOLI75X1.5</td>
<td>1.500</td>
<td>0.347</td>
<td>0.248</td>
</tr>
<tr>
<td>ECOLI75X3.0</td>
<td>3.000</td>
<td>1.262</td>
<td>0.988</td>
</tr>
<tr>
<td>ECOLI150X1.5</td>
<td>1.500</td>
<td>0.579</td>
<td>0.348</td>
</tr>
<tr>
<td>ECOLI150X3.0</td>
<td>3.001</td>
<td>1.781</td>
<td>1.241</td>
</tr>
</tbody>
</table>
Accuracy Evaluation for Re-sequencing
## Quality Evaluation for de-novo Assembly

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Assembler</th>
<th>N50</th>
<th>N90</th>
<th>MAX</th>
<th>No. of Contigs</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRR006331</td>
<td>Velvet</td>
<td>5315</td>
<td>1590</td>
<td>15137</td>
<td>310</td>
</tr>
<tr>
<td></td>
<td>DecGPU-Velvet</td>
<td>10396</td>
<td>2223</td>
<td>40784</td>
<td>213</td>
</tr>
<tr>
<td></td>
<td>ABySS</td>
<td>5706</td>
<td>1598</td>
<td>15951</td>
<td>334</td>
</tr>
<tr>
<td></td>
<td>DecGPU-ABySS</td>
<td>6130</td>
<td>1710</td>
<td>21981</td>
<td>308</td>
</tr>
<tr>
<td>SRR016146</td>
<td>Velvet</td>
<td>37221</td>
<td>10607</td>
<td>91592</td>
<td>259</td>
</tr>
<tr>
<td></td>
<td>DecGPU-Velvet</td>
<td>43499</td>
<td>11998</td>
<td>170070</td>
<td>237</td>
</tr>
<tr>
<td></td>
<td>ABySS</td>
<td>34116</td>
<td>9041</td>
<td>167883</td>
<td>342</td>
</tr>
<tr>
<td></td>
<td>DecGPU-ABySS</td>
<td>37522</td>
<td>10225</td>
<td>134316</td>
<td>298</td>
</tr>
<tr>
<td>SRR001665</td>
<td>Velvet</td>
<td>67285</td>
<td>16964</td>
<td>268020</td>
<td>255</td>
</tr>
<tr>
<td></td>
<td>DecGPU-Velvet</td>
<td>95457</td>
<td>30793</td>
<td>268205</td>
<td>176</td>
</tr>
<tr>
<td></td>
<td>ABySS</td>
<td>95754</td>
<td>25960</td>
<td>212280</td>
<td>136</td>
</tr>
<tr>
<td></td>
<td>DecGPU-ABySS</td>
<td>95760</td>
<td>26095</td>
<td>268359</td>
<td>124</td>
</tr>
</tbody>
</table>
Scalability

- **FPP of a Bloom filter** (where $N_B =$ number of buckets, $N_E =$ number of elements, $\alpha = hN_E / N_B$):
  - DecGPU: $h = 8$, max $N_B = 2^{32}$
  - 2GB memory
- **FPP and maximal $N_E$ for representative $\alpha$ values**

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>FPP</th>
<th>Maximal $N_E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2.5 \times 10^{-2}$</td>
<td>536870912</td>
</tr>
<tr>
<td>0.5</td>
<td>$5.7 \times 10^{-4}$</td>
<td>268435456</td>
</tr>
<tr>
<td>0.25</td>
<td>$5.7 \times 10^{-6}$</td>
<td>134217728</td>
</tr>
<tr>
<td>0.125</td>
<td>$3.6 \times 10^{-8}$</td>
<td>67108864</td>
</tr>
</tbody>
</table>

- **We estimate the maximal number of reads that can be processed by $N_{PE}$ MPI processes as**:

$$N_R = N_{PE} \cdot N_E \cdot \frac{E(N_{kmer})}{L - k + 1} = N_{PE} \cdot N_E \cdot \frac{C}{L}$$

- **Example**: $C=75$, $L=36$, $N_{PE}=8$: $N_R = 2.24$ billion for $\alpha = 0.25$
mCUDA-MEME: Motif finding

- Complications of identifying common motifs in DNA sequences:
  - We do not know the motif sequence
  - We do not know where it is located relative to the genes start
  - Motifs can differ slightly from one gene to the next
  - How to discern it from “random” motifs?
mCUDA-MEME: Motif finding

Table 2. Execution time (in seconds) and speedup comparison between mCUDA-MEME and parallel MEME

<table>
<thead>
<tr>
<th>Datasets</th>
<th>mCUDA-MEME</th>
<th>parallel MEME</th>
<th>Speedups</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8 GPUs</td>
<td>1 GPU</td>
<td>32 CPU cores</td>
</tr>
<tr>
<td></td>
<td>SPS</td>
<td>All</td>
<td>SPS</td>
</tr>
<tr>
<td>NRSF500</td>
<td>72</td>
<td>115</td>
<td>532</td>
</tr>
<tr>
<td>NRSF1000</td>
<td>279</td>
<td>392</td>
<td>2129</td>
</tr>
<tr>
<td>NRSF2000</td>
<td>1244</td>
<td>1663</td>
<td>9554</td>
</tr>
</tbody>
</table>

- CUDA-MEME is integrated in the CompleteMotifs pipeline (http://cmotifs.tchlab.org) for ChiP-Seq data analysis
CUDA-BLASTP

- **Filtration approach:**
  - Assumes good alignments contain short exact matches
  - Find such matches quickly using data structures such as lookup tables
  - Identified short matches are used as seeds for further detailed analysis
**CUDA-BLASTP**

<table>
<thead>
<tr>
<th>Query Sequence Length</th>
<th>SSEARCH Runtime</th>
<th>NCBI BLASTP Runtime</th>
<th>CUDA-BLASTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>127(P14144)</td>
<td>41.9</td>
<td>34.5</td>
<td>5.8</td>
</tr>
<tr>
<td>254(P42018)</td>
<td>65.6</td>
<td>36.0</td>
<td>5.9</td>
</tr>
<tr>
<td>517(Q52TG9)</td>
<td>189.4</td>
<td>68.3</td>
<td>11.9</td>
</tr>
<tr>
<td>1054(Q52KR2)</td>
<td>262.8</td>
<td>131.7</td>
<td>27.7</td>
</tr>
<tr>
<td>2026(P08678)</td>
<td>616.6</td>
<td>155.3</td>
<td>29.2</td>
</tr>
</tbody>
</table>

**TABLE II**

Runtime profiling (in seconds) of each stage for scanning the GenBank Non-Redundant Protein Database with CUDA-BLASTP on an NVIDIA GeForce GTX 280 for varying query lengths.

<table>
<thead>
<tr>
<th>Query Sequence Length</th>
<th>Pre-Processing Stage</th>
<th>Stage 1 and 2</th>
<th>Stage 3</th>
<th>Stage 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Kernel1</td>
<td>Kernel1</td>
<td>Total</td>
</tr>
<tr>
<td>127(P14144)</td>
<td>0.26</td>
<td>3.45</td>
<td>1.52</td>
<td>4.97</td>
</tr>
<tr>
<td>254(P42018)</td>
<td>0.29</td>
<td>3.70</td>
<td>1.55</td>
<td>5.25</td>
</tr>
<tr>
<td>517(Q52TG9)</td>
<td>0.30</td>
<td>6.07</td>
<td>1.57</td>
<td>7.64</td>
</tr>
<tr>
<td>1054(Q52KR2)</td>
<td>0.29</td>
<td>10.24</td>
<td>1.58</td>
<td>11.82</td>
</tr>
<tr>
<td>2026(P08678)</td>
<td>0.30</td>
<td>15.71</td>
<td>1.61</td>
<td>17.32</td>
</tr>
</tbody>
</table>

**TABLE I**

Run times (in seconds) for scanning the GenBank Non-Redundant Protein Database (contains 3,163,461,953 amino acids in 9,230,955 sequences) with SSEARCH35 (multi-threaded version with SSE2 optimization) and NCBI BLASTP 2.2.22 (single-threaded version) running on an Intel Quad-Core i7-920 2.66GHz CPU and CUDA-BLASTP on an NVIDIA GeForce GTX 280 for varying query lengths. The query sequences have the accession numbers P14144, P42018, Q52TG9, Q52KR2, and P08678. The corresponding speedups compared to SSEARCH35 (Speedup1) and NCBI BLASTP 2.2.22 (Speedup2) are also reported.
Summary and Current Work

- DecGPU
  - [http://decgpu.sourceforge.net](http://decgpu.sourceforge.net)
- CUDA-SW++
- mCUDA-MEME
  - [http://sites.google.com/site/yongchaosource/mcuda-meme](http://sites.google.com/site/yongchaosource/mcuda-meme)
- CUDA-BLASTP
  - [https://sites.google.com/site/liuweiguohome/software](https://sites.google.com/site/liuweiguohome/software)
- Tools currently under development
  - CRiSPy (Computing Species Richness in 16S rRNA Pyrosequencing Datasets with CUDA)
  - PASHA (denovo genome assembler)
  - MSAProbs (CUDA version)
  - More versions of BLAST