Algorithms and Tools for Bioinformatics on GPUs

Bertil Schmidt
Data Explosion

Growth Rate of GenBank and UniProtKB/TrEMBL

- Blue diamonds represent GenBank Base Pairs.
- Red squares represent GenBank Entries.
- Green triangles represent UniProtKB/TrEMBL Amino Acids.
- Purple circles represent UniProtKB/TrEMBL Entries.

Year:
- 1980
- 1985
- 1990
- 1995
- 2000
- 2005
- 2010
- 2015
Contents

• Overview
  – CUDA-enabled HPC Bioinformatics Software developed by my group
• Pairwise Sequence Alignment
• Multiple Sequence Alignment
• CUDA-BLASTP
• Short Read Error Correction using CUDA
• Short Read Alignment with CUDA
• Motif Finding with CUDA
• Metagenomics Sequencing Studies with CUDA
• Conclusion
CUDA-enabled HPC Bioinformatics Software developed by my group

• Sequence database searching
  – CUDASW++ (Smith-Waterman)
  – CUDA-BLASTP

• Multiple sequence alignment
  – MSA-CUDA

• Next-Generation Sequencing (NGS)
  – DecGPU (short-read error correction)
  – CUSHAW (short-read mapping)
  – CRiSPy-CUDA (short-read clustering)

• Motif finding
  – CUDA-MEME

• Accessible via: http://hpc.informatik.uni-mainz.de/
Local Pairwise Sequence Alignment

Align \( S_1 = \textcolor{#0000FF}{\text{ATCTCGTATGATG}} \) \( S_2 = \textcolor{#FF0000}{\text{GTCTATCAC}} \)

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

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

\[
H(i, j) = \max \begin{cases} 
0 \\
H(i-1, j) - 1 \\
H(i, j-1) - 1 \\
H(i-1, j-1) + Sbt(S_{1i}, S_{2j})
\end{cases}
\]
**Extraction of Parallelism**

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

- 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)

<table>
<thead>
<tr>
<th></th>
<th>FOS_RAT</th>
<th>FOS_MOUSE</th>
<th>FOS_CHICK</th>
<th>FOSB_MOUSE</th>
<th>FOSB_HUMAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protein</td>
<td>MMFSGFNADYEASSRSSCSSASPDGSLSYYHSAPDSFSSSMGSPVNTQDFCADLSVSSANF 60</td>
<td>MMFSGFNADYEASSRSSCSSASPDGSLSYYHSAPDSFSSSMGSPVNTQDFCADLSVSSANF 60</td>
<td>MMYQGFAGEYEAPSSCSSASSPAGDSLTYYPSAPDSFSSSMGSPVNSQDFCTDLAVSSANF 60</td>
<td>-MFQAFPGDYDS-GSRCSS-SPSAESQ--YLSSVDSFGSPPTAAASQE-CAGLGEMPGSF 54</td>
<td>-MFQAFPGDYDS-GSRCSS-SPSAESQ--YLSSVDSFGSPPTAAASQE-CAGLGEMPGSF 54</td>
</tr>
</tbody>
</table>

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

#### Diagrams
- **Distance Matrix Computation**
- **Guided Tree**
- **Progressive Alignment along the Guided Tree**

#### Profiling the three stages of ClustalW
- **Stage 1 (Distance matrix)** requires more than 90% of overall runtime
### Dist Matrix: $O(n^2 \cdot l^2)$

<table>
<thead>
<tr>
<th></th>
<th>HA HU</th>
<th>HB HU</th>
<th>HA HO</th>
<th>HB HO</th>
<th>MY WH</th>
<th>P1 LH</th>
<th>LG HB</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>HBHO</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>P1LH</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>

### NJ-Tree: $O(n^3)$

- **Alignment**:
  - HAHU
  - HBHU
  - HBHO

### Progressive: $O(n \cdot l^2)$

- **Alignment**:
  - HAHU
  - HBHU
  - HBHO

### Dynamic Programming

- HAHU
- HBHU
- HBHO
CUDA Parallelization: Stage 1

- 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
- **Cell/BE:** MIMD Parallelization with Cell/BE SDK

- **FPGA:** Load Balancing, Partitioning (FIFO, Multi-Lanes)
- **GPU:** Load Balancing, Optimization of memory accesses (coalesced, shared)
- **Cell/BE:** Load Balancing, SIMD Vectorization
Comparison: Speedup and Productivity (Stage 1)

- **FPGA**: Xilinx XC5VLX330
  - 416 PEs
  - 65 MHz

- **GPU**: GeForce GTX 280
  - 240 SPs
  - 1.3 GHz

- **Cell/BE**: PlayStation3
  - 6 SPEs
  - SIMD vector-length: 8
  - 3.2 GHz
**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
  - Best Paper Award at IEEE ASAP 2009 (Boston)
CUDA-BLASTP

- “Seed-and-extend” approach:
  - Assumes good alignments contain short exact matches
  - Find such matches quickly using lookup data structures
  - Identified short matches are used as seeds for further extension
Next-Generation Sequencing (NGS)

- ultra-high throughput
- short read-length

- 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)

<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>
Short Read Error Correction with SAP

Changing the single error at position 6 in the given read from G to A results in l corresponding matches in the spectrum.

l-mer Spectrum = {..., TCA\textcolor{blue}{A}, CA\textcolor{red}{A}C, A\textcolor{red}{A}CG, ACGT, ...}

\[
\begin{array}{c|cccccccccc}
    \text{A} & 0 & 0 & 0 & 0 & 0 & 0 & \textbf{4} & 0 & 0 & 0 & 0 \\
    \text{C} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
    \text{G} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
    \text{T} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

Corrected Read \( r_i[6][A] \): T T G T C A A C G T A
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
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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Euler-SR</th>
<th>DecGPU</th>
<th>Overall Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SC</td>
<td>EC</td>
<td>SC</td>
</tr>
<tr>
<td>A</td>
<td>61</td>
<td>671</td>
<td>16</td>
</tr>
<tr>
<td>B</td>
<td>338</td>
<td>1524</td>
<td>93</td>
</tr>
<tr>
<td>C</td>
<td>746</td>
<td>7016</td>
<td>222</td>
</tr>
</tbody>
</table>

<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></td>
<td></td>
</tr>
<tr>
<td>ECOLI30X1.5</td>
<td>1.498</td>
<td>0.426</td>
<td>101</td>
</tr>
<tr>
<td>ECOLI30X3.0</td>
<td>3.003</td>
<td>1.773</td>
<td>136</td>
</tr>
<tr>
<td>ECOLI75X1.5</td>
<td>1.500</td>
<td>0.347</td>
<td>253</td>
</tr>
<tr>
<td>ECOLI75X3.0</td>
<td>3.000</td>
<td>1.262</td>
<td>329</td>
</tr>
<tr>
<td>ECOLI150X1.5</td>
<td>1.500</td>
<td>0.579</td>
<td>834</td>
</tr>
<tr>
<td>ECOLI150X3.0</td>
<td>3.001</td>
<td>1.781</td>
<td>1062</td>
</tr>
</tbody>
</table>
Evaluation for Re-sequencing and de-novo Assembly

![Bar Chart](chart.png)

Maximum number of mismatches

<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>
Background on Short Read Mapping

Reference Genome

Index Data Structure (e.g. Hash Table)

Reads

Mapping of Reads to Reference Genome
**Background on Short Read Mapping**

<table>
<thead>
<tr>
<th>Subject (Reads)</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>... C C A T A G C T A G C A A A T ...</td>
<td>... C C A T A G C T A G C A A A T ...</td>
</tr>
<tr>
<td>... C C A T G G C T A A A A T ...</td>
<td>... C C A T G G C T A A A A T ...</td>
</tr>
<tr>
<td>... C C A G G C T G A A A A T ...</td>
<td>... C C A G G C T G A A A A T ...</td>
</tr>
<tr>
<td>... C C T A G G C A G A A A ...</td>
<td>... C C T A G G C A G A A A ...</td>
</tr>
<tr>
<td>... C A T A G G T A G A A ...</td>
<td>... C A T A G G T A G A A ...</td>
</tr>
<tr>
<td>... A T A G G A G A A A ...</td>
<td>... A T A G G A G A A A ...</td>
</tr>
</tbody>
</table>

**Input:** Given a reference genome and a set of short reads

**Output:** best/all significant alignment(s) for each read

- „Significance“: alignment with smallest number of mismatches/gaps

**Fundamental operation for many NGS applications**

- SNP discovery

**Challenges:** Throughput, Sensitivity, increasing read length and error rates
Background on Short Read Mapping

• Reference Genome and reads are too large for direct DP approach

• „Seed-and-Extend“
  – Use an index data structure to rapidly find short exact matches to seed longer in-exact alignments

• Indexing approaches (memory sizes for human genome)
  – Suffix tree (> 35 GB)
  – Suffix array (> 12 GB)
  – Hash tables (> 12 GB)

• CUSHAW: GPU-Approach
  – Index Reference Genome using BWT (Burrows Wheeler Transform)
  – Needs 2.2 GB memory for Human Genome ⇒ fits on Fermi C2050
  – optimized for Fermi architecture using CUDA
  – CUSHAW currently only supports a restrictive alignment models
    • allows up to 2 mismatches in seed
    • no indels
- **BWT(cattattagga$)**
- Backward search to calculate the SA interval for a substring “**tta**”

\[
\begin{align*}
I_a(i) &= C(S[i]) + \text{Occ}(S[i], I_a(i+1)-1) + 1, \quad 0 \leq i < |S| \\
I_b(i) &= C(S[i]) + \text{Occ}(S[i], I_b(i+1)), \quad 0 \leq i < |S|
\end{align*}
\]
Performance Comparison for \( K \)-mer Match Search with BWT

(Seed Generation)

<table>
<thead>
<tr>
<th>( K )</th>
<th>#Mismatches Allowed</th>
<th>Time (in seconds)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 CPU core</td>
<td>4 CPU cores</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>5.8</td>
<td>3.4</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>7.3</td>
<td>4.1</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>139.8</td>
<td>47.6</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>107.6</td>
<td>37.9</td>
</tr>
<tr>
<td>30</td>
<td>2</td>
<td>1067.6</td>
<td>309.5</td>
</tr>
</tbody>
</table>

**Reference**
CUSHAW: short-read aligner to human genome based on BWT

<table>
<thead>
<tr>
<th></th>
<th>Throughput (SRR002273, 2x36bp)</th>
<th>Sensitivity (SIM 2x120bp, 4% error rate)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSHAW</td>
<td>105M Reads/h</td>
<td>84.3%</td>
</tr>
<tr>
<td>BWA</td>
<td>7M Reads/h</td>
<td>70.9%</td>
</tr>
<tr>
<td>Bowtie</td>
<td>24M Reads/h</td>
<td>49.2%</td>
</tr>
<tr>
<td>SOAP2</td>
<td>4M Reads/h</td>
<td>10.2%</td>
</tr>
</tbody>
</table>

- **Performance comparison**
  - CUSHAW (running on a Fermi C2050 GPU)
  - BWA, Bowtie, SOAP2 running on an Intel i7 CPU (single-threaded)
  - **Throughput**: 8.55 million paired-end reads of length 36bps aligned to human genome
  - **Sensitivity**: 2 million simulated paired reads of length 120bps with 4% error rate aligned to human genome.

- **Reference**
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
Taxonomic clustering of large-scale short-read Metagnomic data

• Profiling of microbial communities
  – in water, human gut, etc.
  – *Example*: water-membrane profiling
• Based on sequencing of hyper-variable regions of 16S rRNA marker gene
• Typical dataset sizes (based on 454 sequencers)
  – average read length 200-600 bps
  – Number of reads 10K to 10M
• **Taxonomy-independent clustering approach**
  – Performs a hierarchical clustering and then bins the reads into **OTUs** (Operational Taxonomic Units) based on a distance threshold
  – Typically, Clustering is computed on a pairwise genetic distance matrix derived from an all-against-all read comparison
• **Advantage**
  – ability to characterize novel microbes
• **Disadvantages:**
  – highly compute/memory-intensive
Execution time of CRiSPy-CUDA

- \( R = \{R_1, \ldots, R_n\} \) input reads:
  1. Compute matrix \( D \) of size \( n \times n \), where \( D_{i,j} \) is the genetic distance between \( R_i \) and \( R_j \)
  2. Hierarchical clustering of \( D \)
  3. Using \( D \), group reads into OTUs at each given distance level \( d \) (e.g. for \( d = 0.02; 0.03; 0.05; 0.1 \))

- Techniques to reduce runtime and memory
  1. Filtration using k-mer distance
  2. Sparse Matrix Representation of distance matrix

Dataset | ESPRIT (in mins) | CRiSPy-CUDA (speedup)
---------|------------------|---------------------
SRR029122 | 167              | 68                  
SRR013437 | 477              | 78                  
SRR064911 | 4720             | 94                  

Fig. 1. The flowchart of the ESPRIT algorithm
Conclusion

• NGS technologies establish the need for scalable Bioinformatics tools that can process massive amounts of short reads

• CUDA is a highly suitable technology to address this need

• NGS algorithms need to be adapted since throughput and read length continues to increase

• Website
  – [http://hpc.informatik.uni-mainz.de/](http://hpc.informatik.uni-mainz.de/)