Parallel Algorithms and Tools for Bioinformatics on GPUs

Bertil Schmidt

JG|U
JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Contents

- Overview
  - HPC Bioinformatics Software developed by my group
- Pairwise Sequence Alignment
- Multiple Sequence Alignment
- Short Read Error Correction
- Short Read Mapping
- De-novo Assembly
- Motif Finding
- Short Read Clustering

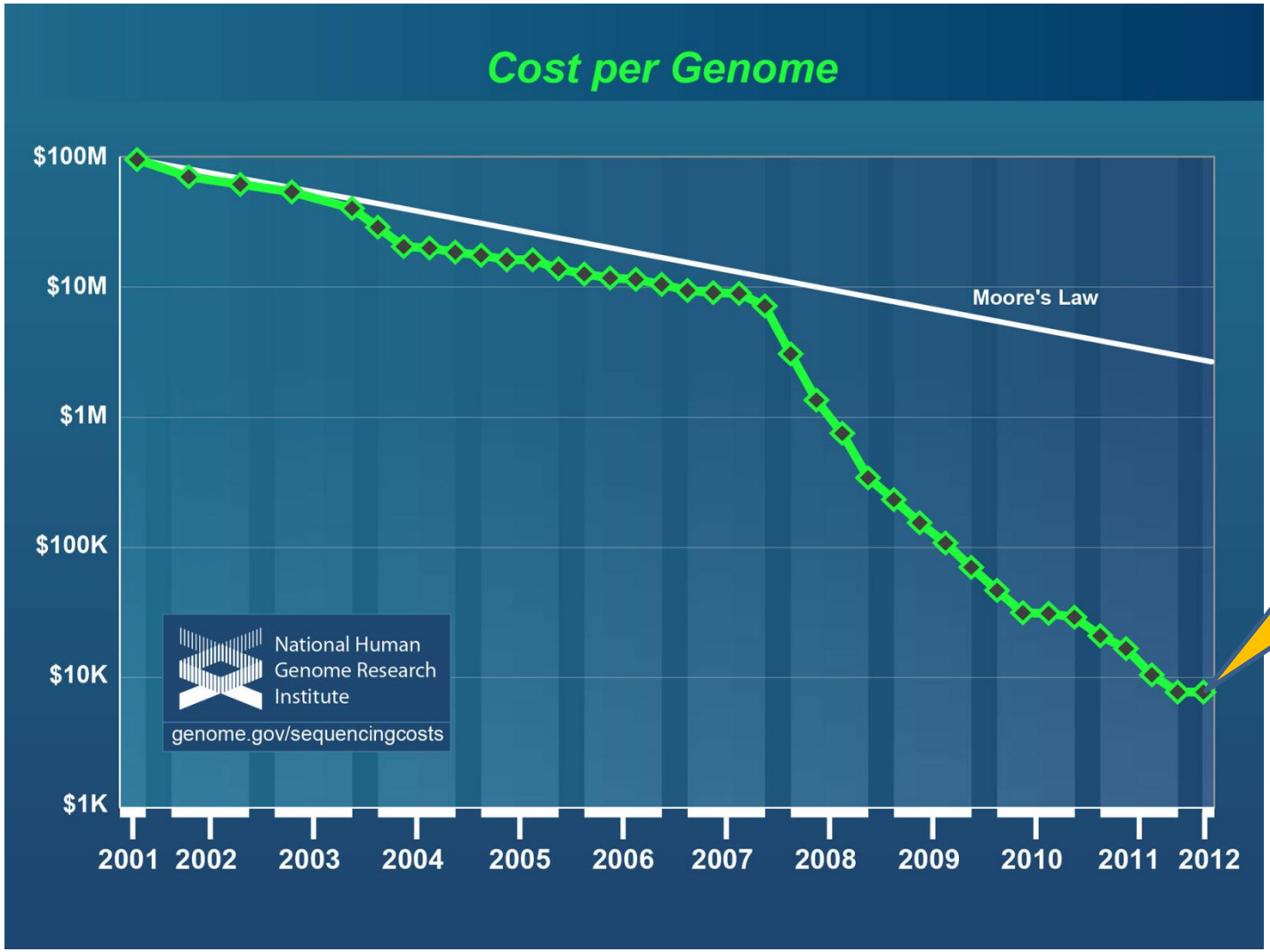
Software developed by my group

- Sequence database searching
  - CUDASW++ (Smith-Waterman)
  - CUDA-BLASTP
- Multiple sequence alignment
  - MSA-CUDA
  - MSAProbs
- Next-Generation Sequencing (NGS)
  - DecGPU, SHREC (short-read error correction)
  - CUSHAW (short-read mapping)
  - CRiSPy-CUDA, DySC (short-read clustering)
  - PASHA, Taipan (de-novo assemby)
- Motif finding
  - CUDA-MEME
- Accessible via: http://hpc.informatik.uni-mainz.de/

Utilized Parallel Architectures and Programming Languages

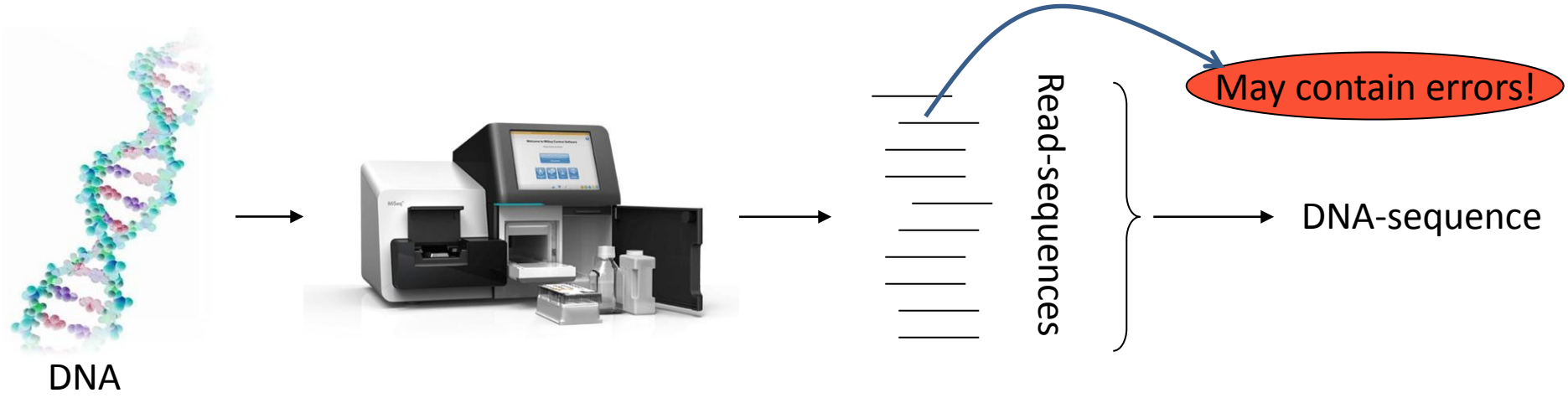
- Multi-core CPUs
  - Pthreads/OpenMP
  - SSE Vectorization
- Many-core GPUs
  - CUDA
- CPU/GPU Clusters
  - MPI
  - Pthreads/OpenMP
  - CUDA
- FPGAs
  - Verilog
- MOGON
  - 81st in latest Top500 list

Cost of DNA Sequencing



1000-fold drop from 2007 to 2012

Next-Generation Sequencing (NGS)



HiSeq2500	
Read length (typical)	100bps
Reads per run	1.2 Billion
Run Time (paired end)	~1 day

- Ultra-high throughput
- Short-read length
- Example: Human Genome Sequencing (Li et al., 2010)
  - 4 billion reads
  - Average length 53bps (71x Coverage)

Common Algorithmic Patterns in NGS Data Analysis

- Indexing and lookup
  - Hashing
  - BWT and FM-Index
  - Bloom filter
- Dynamic programming
  - Smith-Waterman, Needleman-Wunsch
- NGS Bioinformatics Challenges
  - Scalability
    - to deal with huge amounts of reads
  - Algorithm design
    - to deal with short reads
  - Parallelisation
    - Many Bioinformatics algorithms are irregular and therefore challenging to map to parallel architectures

Local Pairwise Sequence Alignment

Align *S1*=***ATCTCGTATGATG*** *S2*=***GTCTATCAC***

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

α=1, β=1

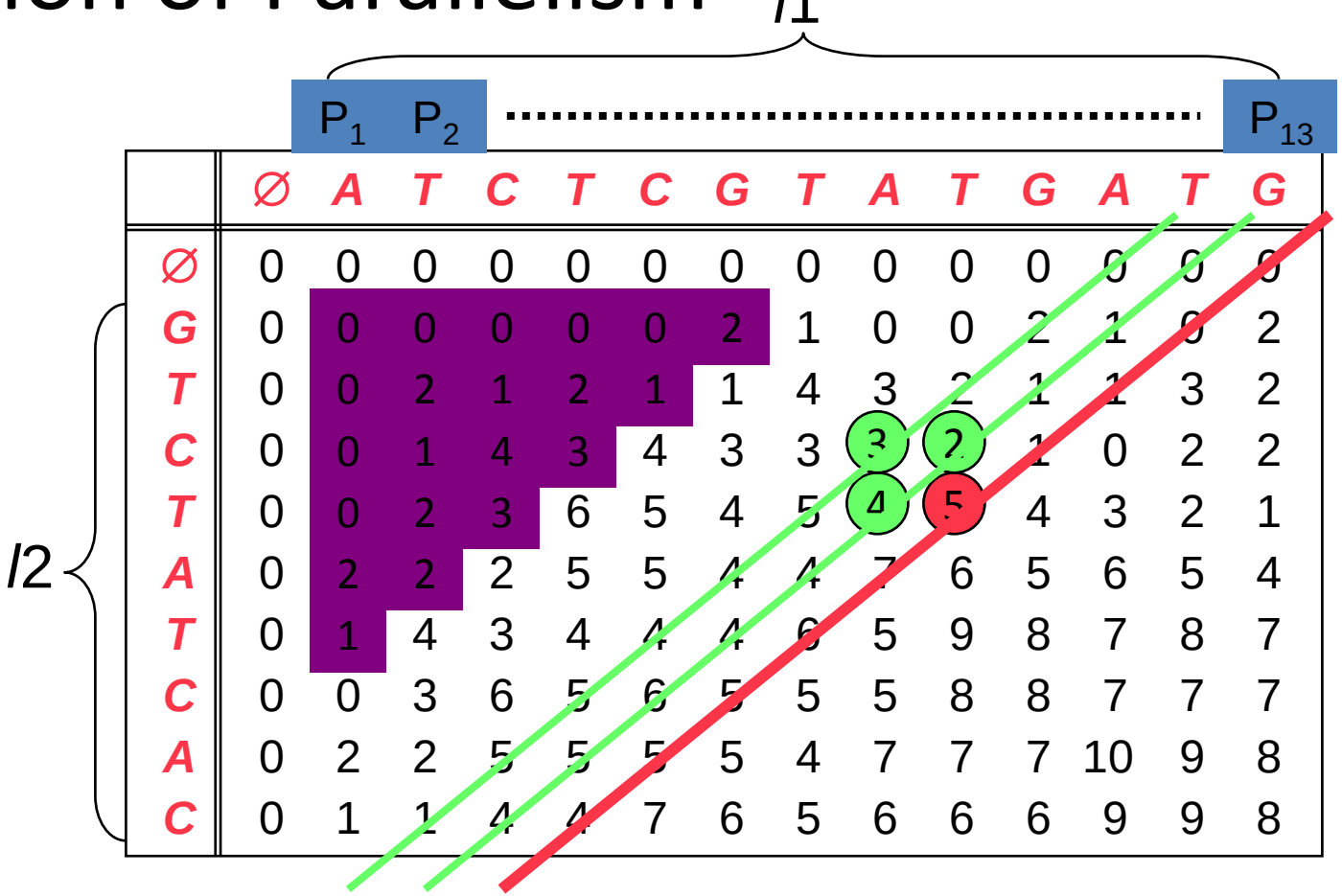
	∅	A	T	C	T	C	G	T	A	T	G	A	T	G
∅	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	2	1	0	0	2	1	0	2
T	0	0	2	1	2	1	1	4	3	2	1	1	3	2
C	0	0	1	4	3	4	3	3	3	2	1	0	2	2
T	0	0	2	3	6	5	4	5	4	5	4	3	2	1
A	0	2	2	2	5	5	4	4	7	6	5	6	5	4
T	0	1	4	3	4	4	4	6	5	9	8	7	8	7
C	0	0	3	6	5	6	5	5	5	8	8	7	7	7
A	0	2	2	5	5	5	5	4	7	7	7	10	9	8
C	0	1	1	4	4	7	6	5	6	6	6	9	9	8

$$H(i,j)=\max\begin{cases}0\\ H(i-1,j)-1\\ H(i,j-1)-1\\ H(i-1,j-1)+Sbt(S1_i,S2_j)\end{cases}$$

ATCTCGTATGATG

GTC–TATCAC

Extraction of Parallelism

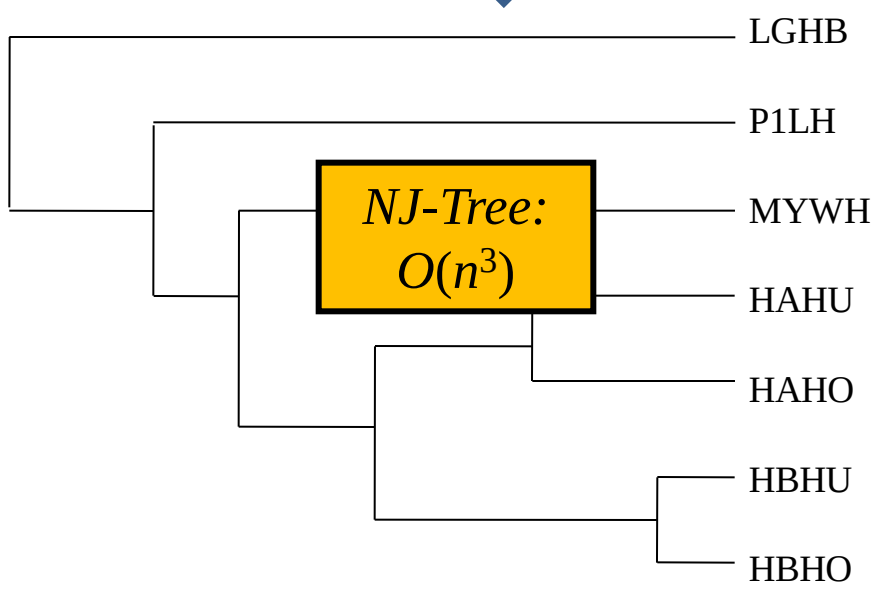


- Ligowski, Rudnicki, Liu, Schmidt: "*Accurate scanning of sequence databases with the Smith-Waterman algorithm*", **GPU Computing Gems, Vol. 1, Chapter 11, 2011**
- Liu, Maskell, Schmidt: "*CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units*", **BMC Research Notes, 2:73, 2009**

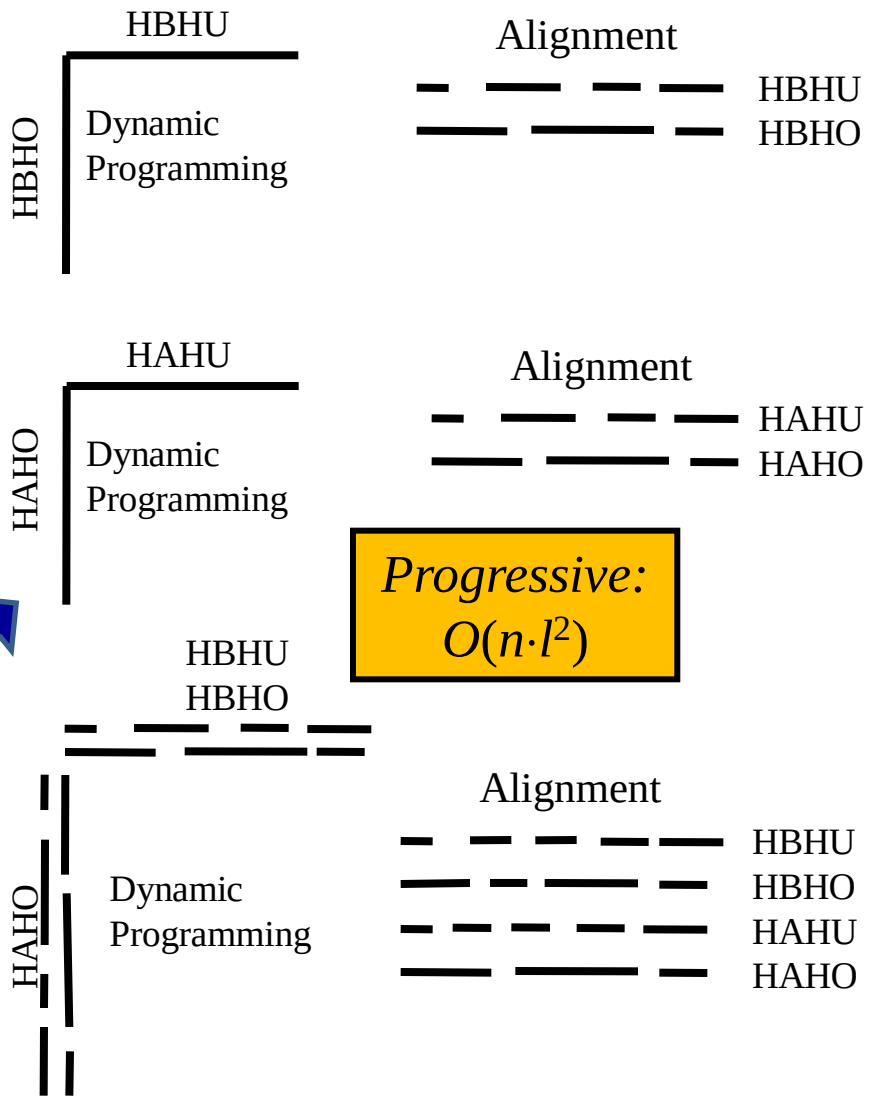
MSA with ClustalW

	HA HU	HB HU	HA HO	HB HO	MY WH	P1 LH	LG HB
HAHU							
HBHU	**21.1**						
HAHO	**32.9**	**19.7**					
HBHO	**20.7**	**39.0**	**20.4**				
MYWH	**11.0**	**9.8**	**10.3**	**9.7**			
P1LH	**9.3**	**8.6**	**9.6**	**8.4**	**7.0**		
LGHB	**7.1**	**7.3**	**7.5**	**7.4**	**7.3**	**4.3**	

Dist Matrix:
$O(n^2 \cdot l^2)$



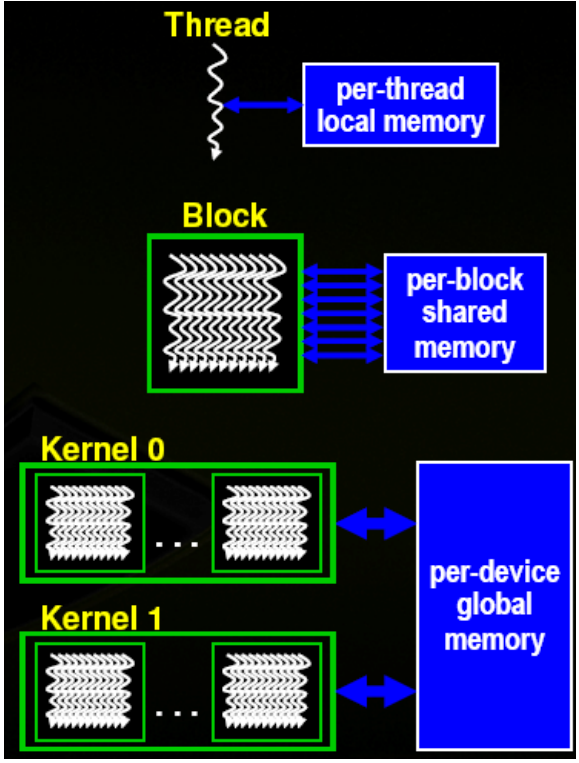
NJ-Tree:
$O(n^3)$

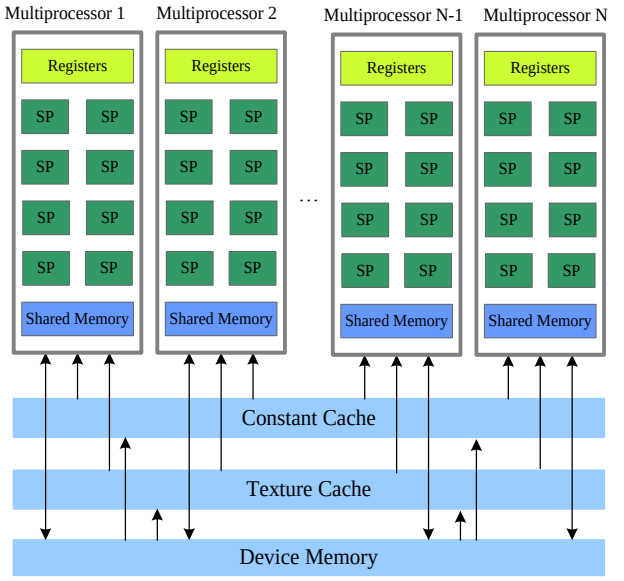


Progressive:
$O(n \cdot l^2)$

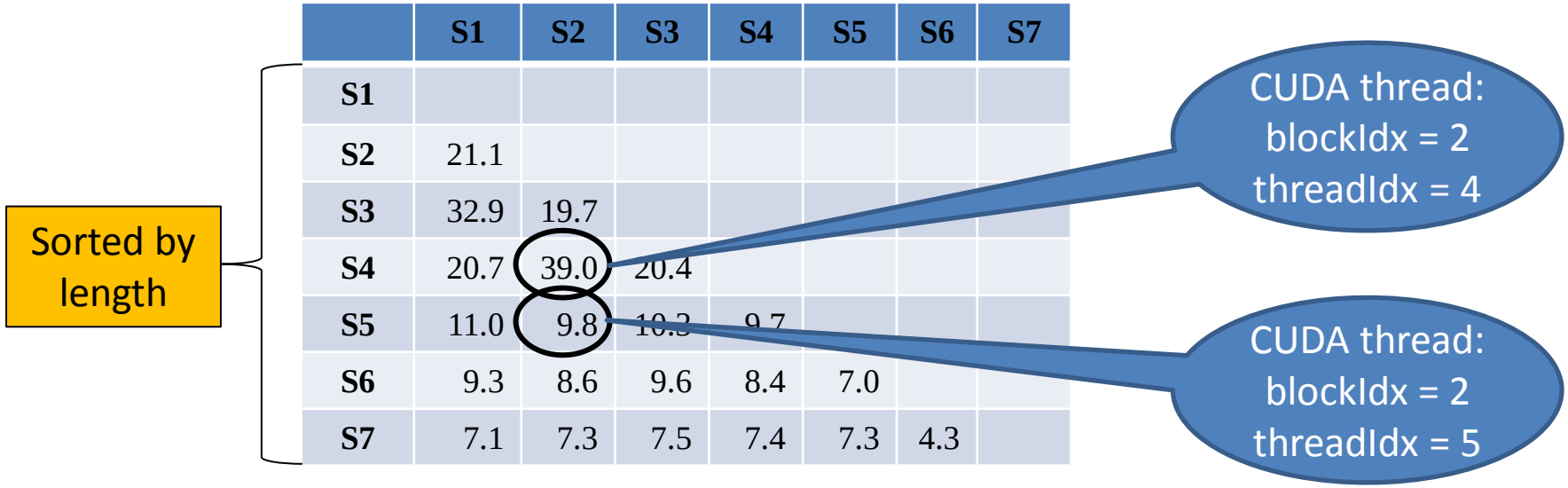
CUDA

- **Kernels** consist of threads
  - All threads execute the same program
  - Local memory per thread
- Threads grouped into **thread blocks**
  - Threads within a block can cooperate through **Per Block Shared Memory** (PBSM)
  - SIMT Model and Local synchronization
  - Threads run in groups of 32 called **warps**
- Threads/blocks have IDs
- GPU global memory
  - **3GB or 6GB RAM** for Fermi-based Tesla
- Texture Memory
  - Cached (texture cache, L2)
  - Texture cache optimized for spatial locality





CUDA Parallelization: Stage 1



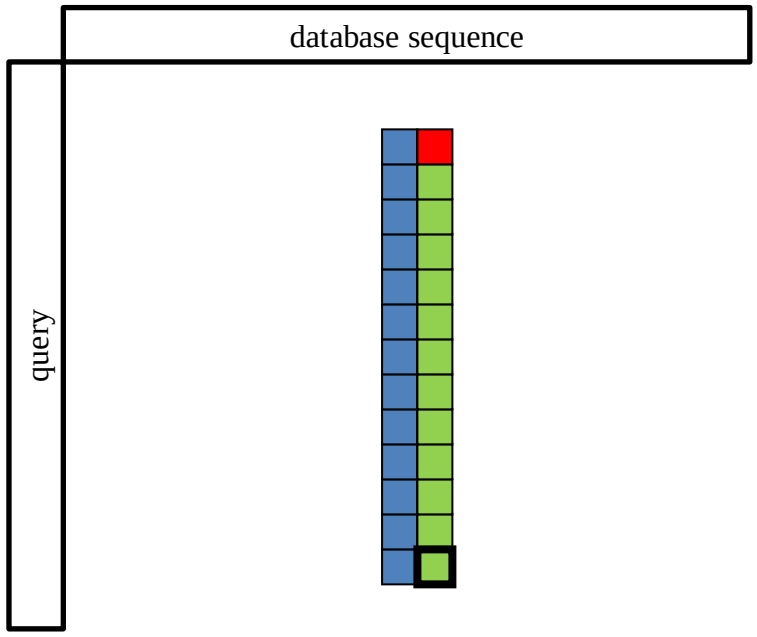
	S1	S2	S3	S4	S5	S6	S7
S1							
S2	21.1						
S3	32.9	19.7					
S4	20.7	39.0	20.4				
S5	11.0	9.8	10.3	9.7			
S6	9.3	8.6	9.6	8.4	7.0		
S7	7.1	7.3	7.5	7.4	7.3	4.3	

- 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

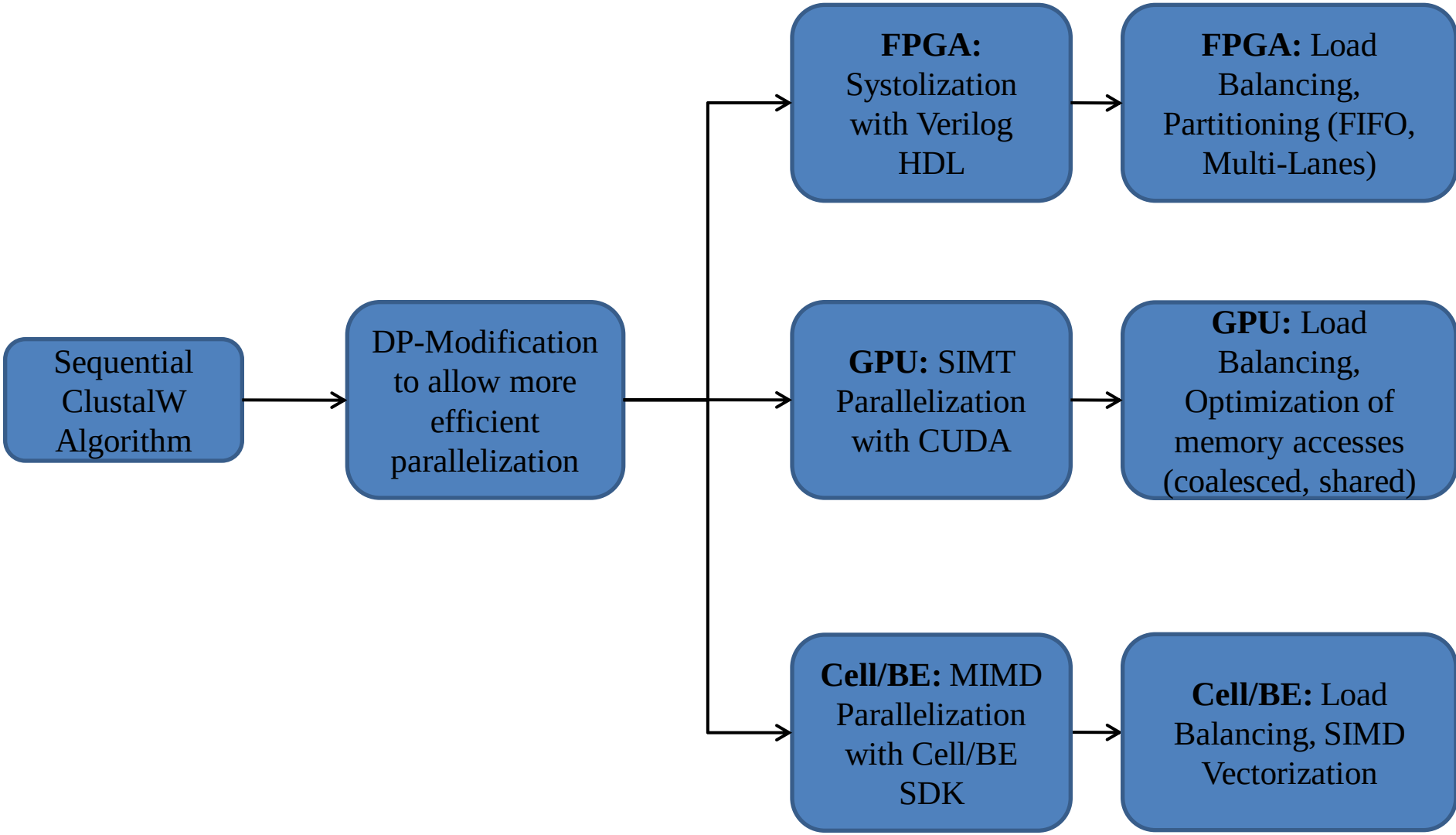
CUDA Parallelization: Stage 1

- Memory access
  - $O(min\{l_a,l_b\})$ storage for intermediate results per thread
  - Stored in global memory using coalesced memory access pattern
  - Partition of DP-matrix into blocks $\Rightarrow$ reduce global memory access by using **shared memory** and **registers**

- Illustration of the use of shared memory
  - Each thread sequentially computes $k = 12$ matrix cells in column j (green) in shared memory.
  - The $k+1$ required cells of column $j-1$ (blue) have been computed in the previous iteration and are therefore already stored in shared memory.
  - However, the upper neighboring cell (red) needs to be read from global memory.
  - At the end of the computation only the bottom cell (fat, green) needs to be written to global memory

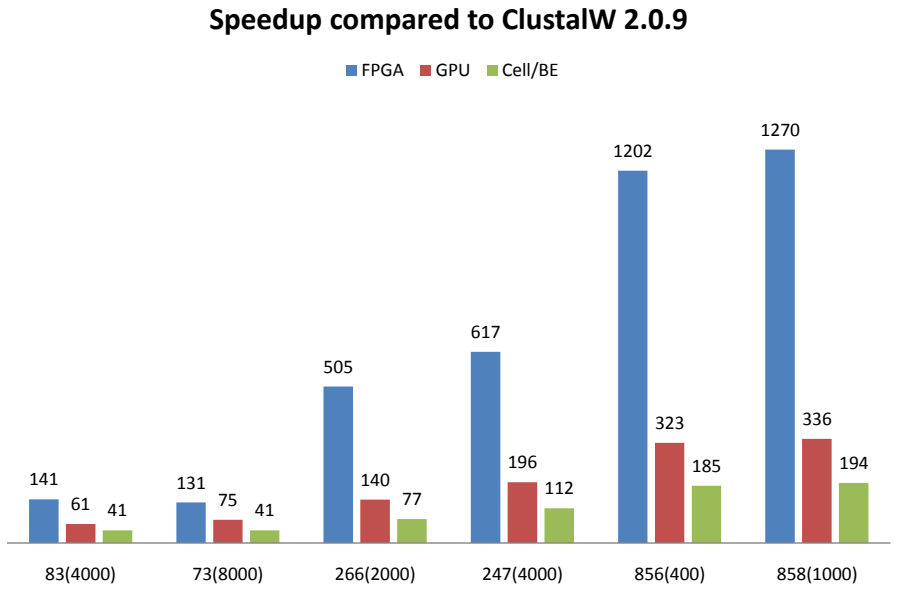


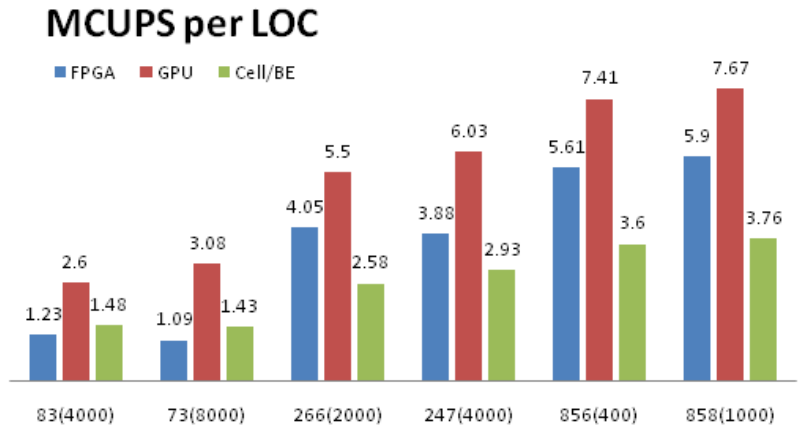
Overview: *Parallelization Approaches*



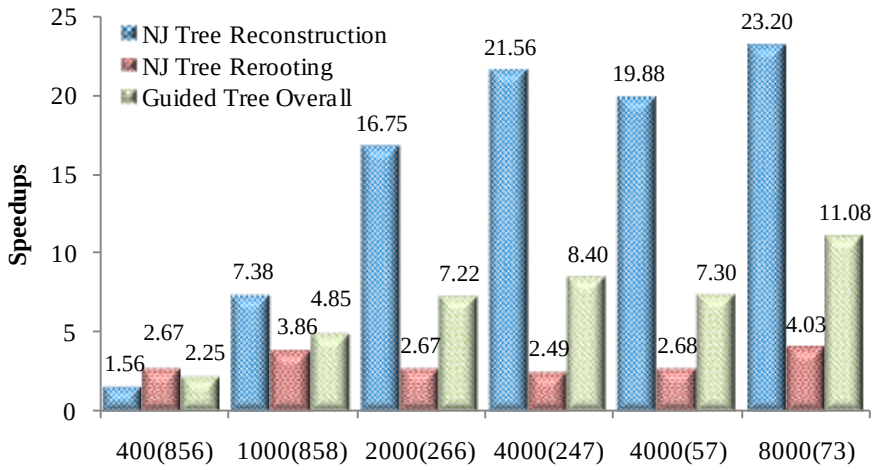
Performance Comparison: Speedup and Productivity (Stage 1)

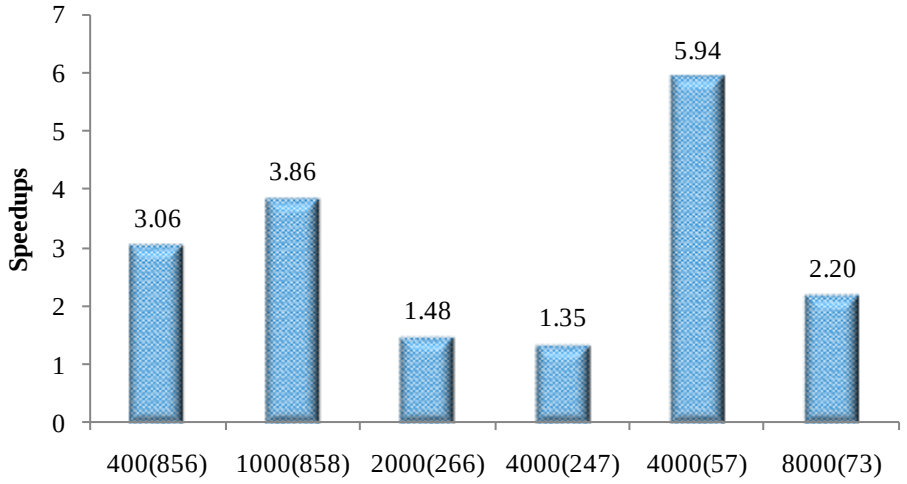
- **FPGA:** Xilinx XC5VLX330
  - 416PEs
  - 65MHz
- **GPU:** GeForce GTX 280
  - 240SPs
  - 1.3GHz
- **Cell/BE:** PlayStation3
  - 6SPEs
  - SIMD vector-length:8
  - 3.2GHz





MSA-CUDA: Performance Stage 2 + 3

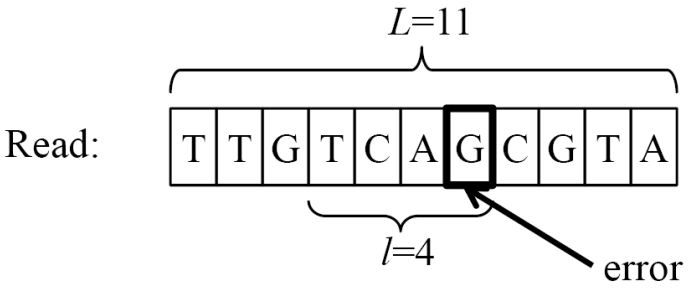




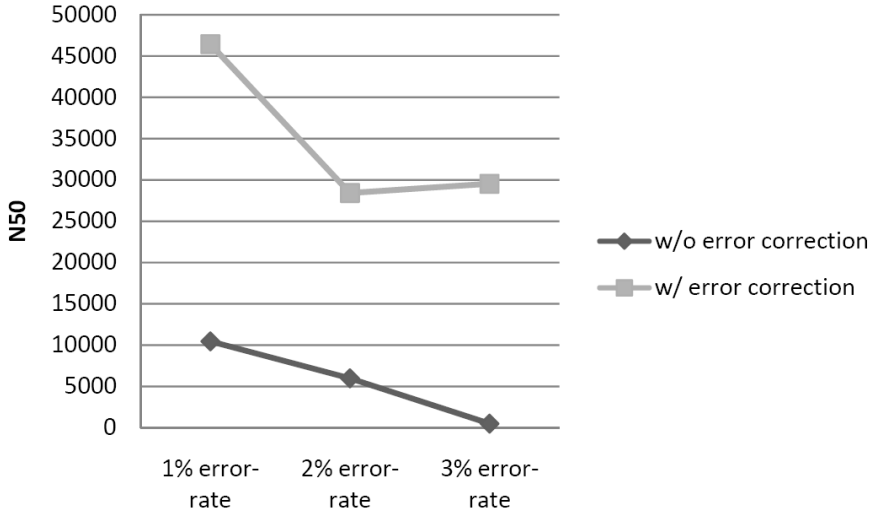
- MSA-CUDA on a GPU (GeForce GTX280)
  - Better Performance than ClustalW-MPI on a PC-cluster with 32 Cores for all tested datasets
  - Best Paper Award at IEEE ASAP 2009 (Boston)
- MSAProbs: Improving MSA accuracy
  - Overall mean scores and runtimes on **BAliBASE 3.0**

Aligner	SPS	CS	Time (hh:mm:ss)
MSAProbs	**89.09**	**64.51**	1:12:56
MAFFT	87.50	61.07	0:41:05
ProbCons	88.31	61.89	5:29:15
ClustalW	78.65	44.75	0:18:56

Short Read Error Correction with SAP



l-mer Spectrum = {…, TCA**A**, CA**A**C, A**A**CG, **A**CGT, …}



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

A	0	0	0	0	0	0	**4**	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0

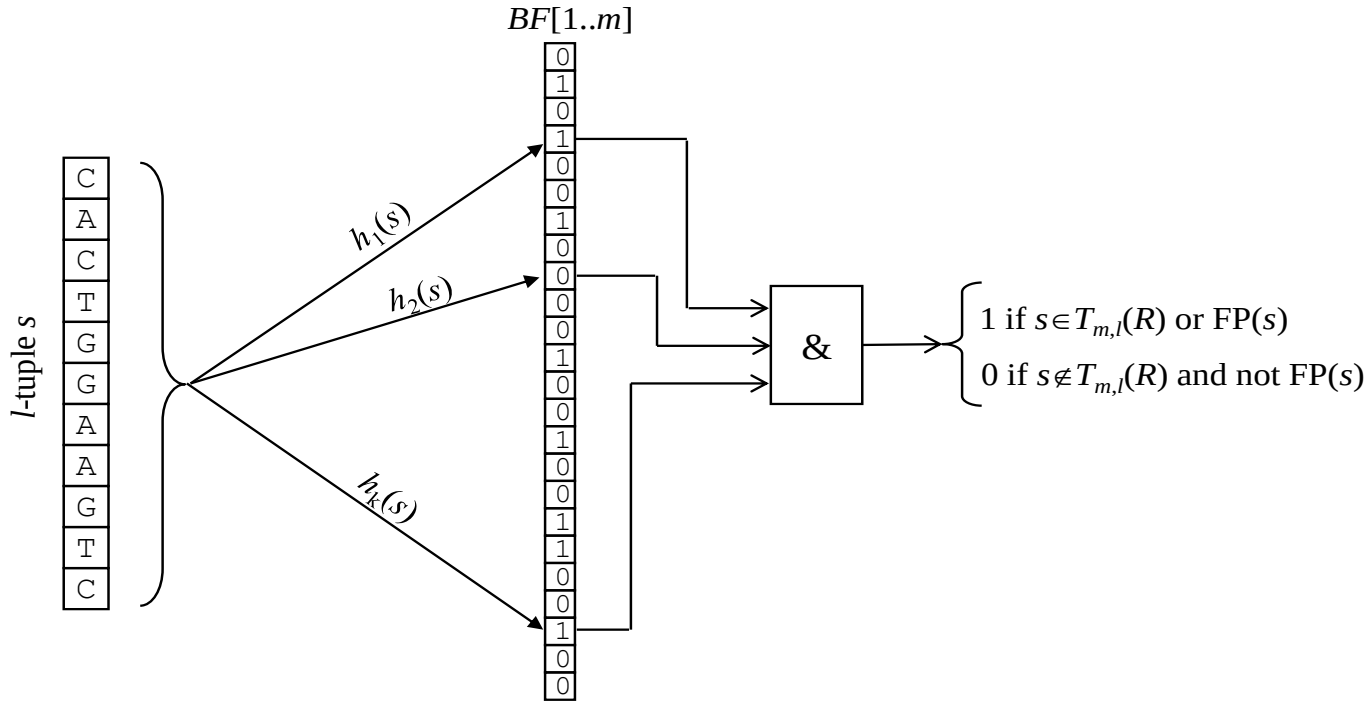
Changing the single error at position 6 in the given read from **G** to **A** results in *l* corresponding matches in the spectrum.

Corrected Read $r_i[6][A]$:

T	T	G	T	C	A	A	C	G	T	A

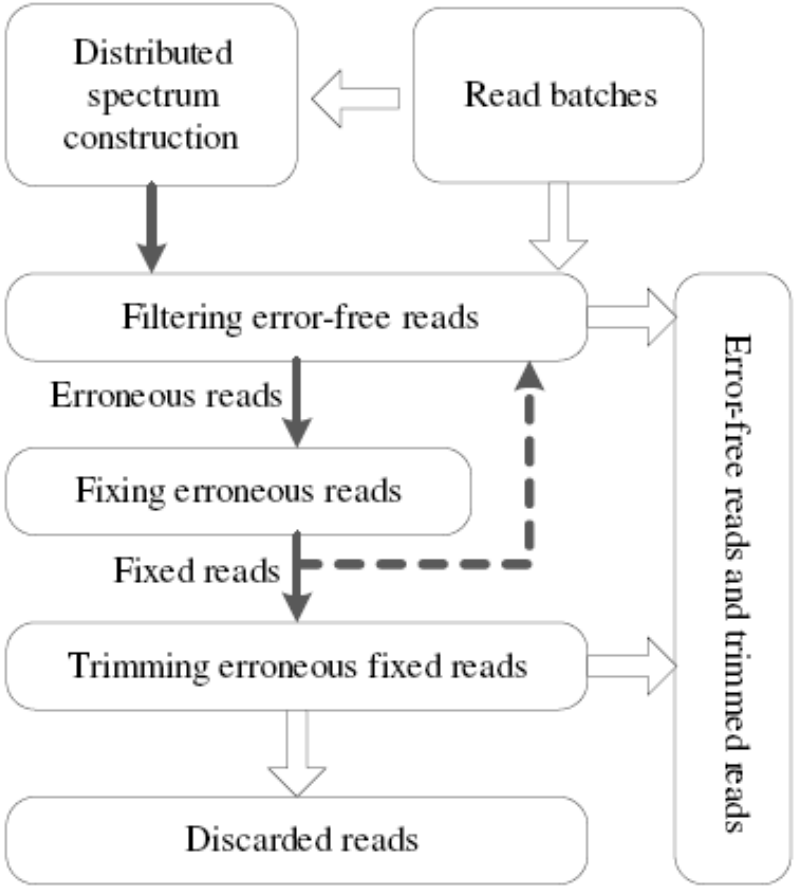
Bloom Filter Data Structure

- Membership test most important operation (test if an *l*-mer is in $T(m,R)$)
- $\rightarrow$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



DecGPU: Performance Evaluation

Dataset	Euler-SR (runtime in sec)		DecGPU (runtime in sec)		Overall Speedup
	Spectrum Construction	*Error Correction*	*Spectrum Construction*	*Error Correction*	
A	61	671	16	10	**28.2**
B	746	7016	222	94	**24.5**

Dataset	Assembler	N50	N90	#contigs
SRR006331	Velvet	5.3K	1.6K	310
	DecGPU-Velvet	10.4K	2.2K	213
SRR001665	Velvet	67.3K	17.0K	255
	DecGPU-Velvet	95.5K	30.8K	176

PASHA: Short read de-novo assembly using MPI + pThreads

- Assembly often modeled as finding an Eulerian tour of the **de Bruijn graph**
- Assemblies of complex genomes highly fragmented, since there is generally an exponential number of compatible sequences
- Efficient GPU parallelization is difficult

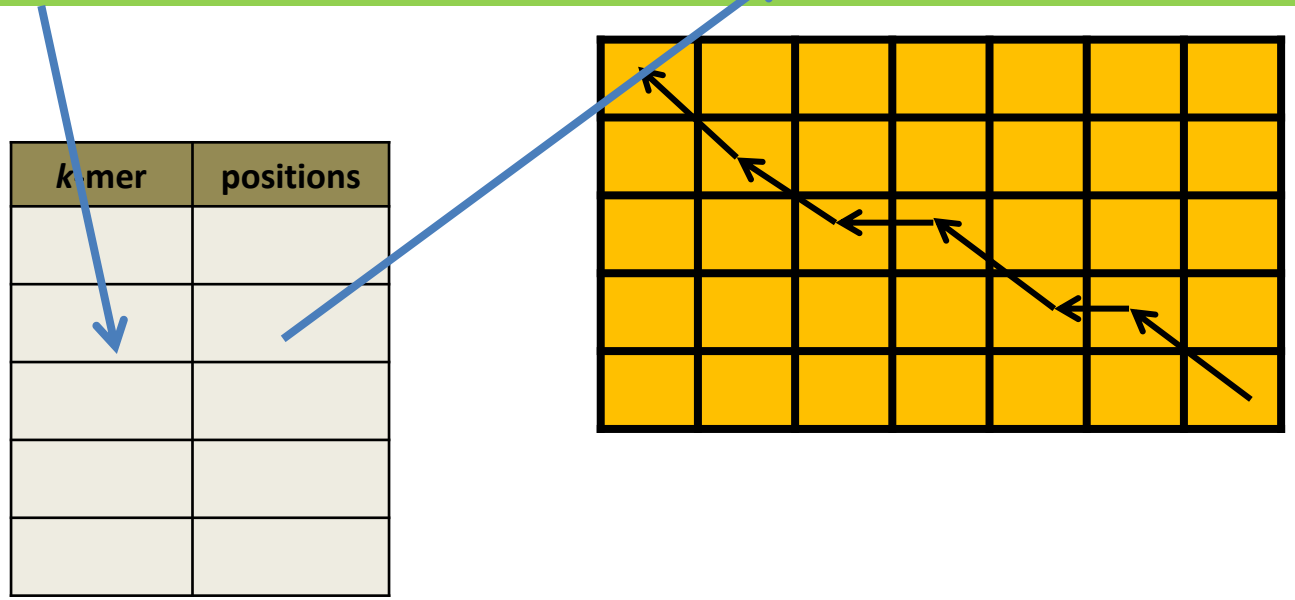
Assembly E.coli (4.6M genome size) 20.8M Reads (2x36bps), single core		
	PASHA	**ABySS**
N50	164.4K	96.3K
Genome Coverage	97.4%	95.6%
Runtime	5.4 mins	9.9 mins

Human Genome Assembly (3.1G genome size) 3.76 billion reads (36-42bps, 200bp library) 8-node CPU Cluster (8-cores and 24GB RAM per node)		
	PASHA	**ABySS**
N50	2.3K	1.3K
Genome Coverage	66.9%	71.5%
Runtime	21 hours	51 hours

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, e.g.
  - Build a hash table on all *k*-mers of genome
  - Segment query sequence into *k*-mer seeds

Ref Genome: …CAAACCAGCTCTTA**TGGTCAGAAC**TCTGAAAGACAACTGAGCTGCTG…
Read Seed: **TGGTCAGAAC**

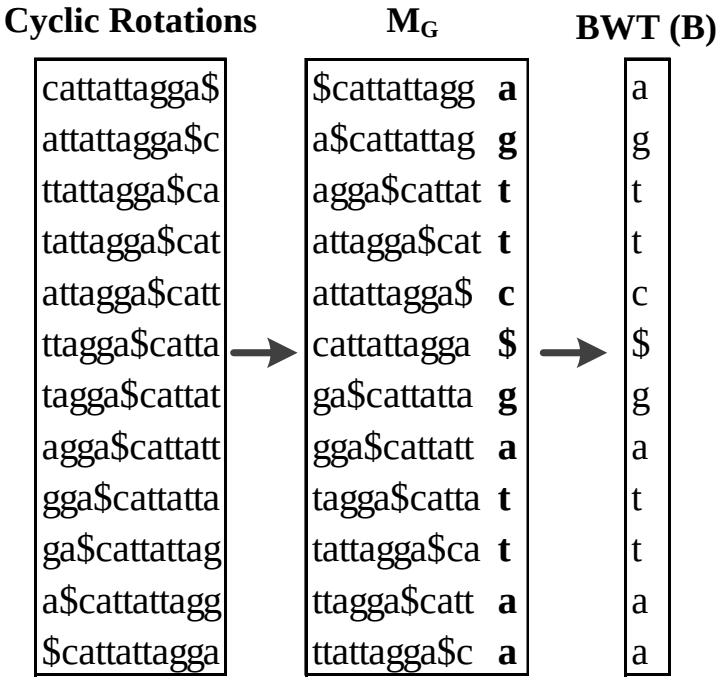


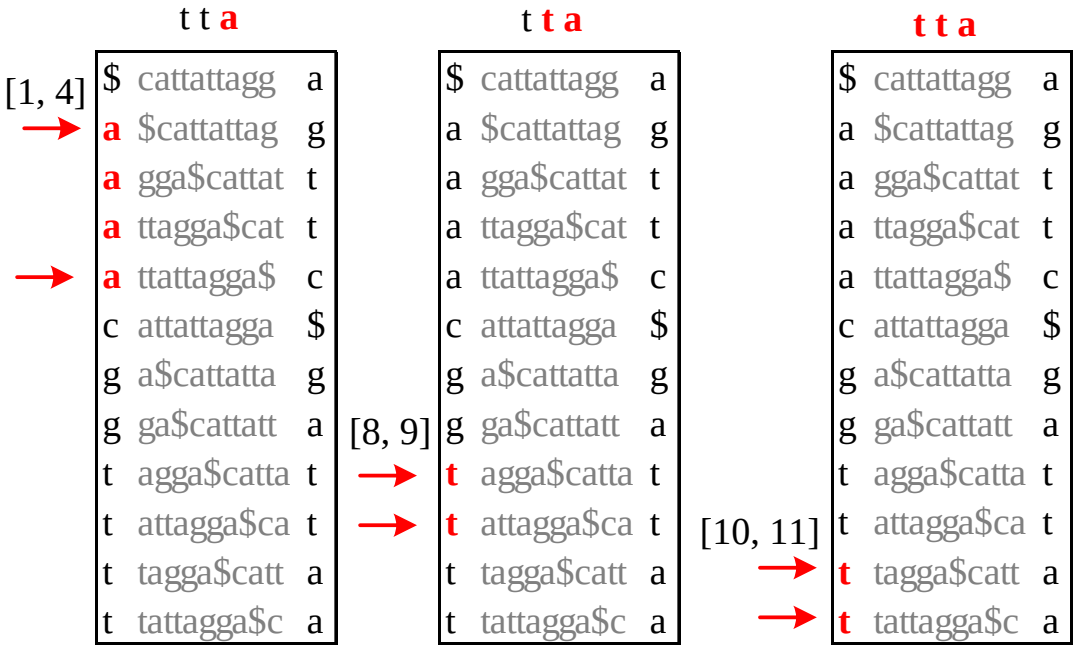
CUSHAW: short-read aligner to human genome based on BWT

- *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 $\Rightarrow$ 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***"

$$\begin{cases} I_a(i) = C(S[i]) + Occ(S[i], I_a(i+1) - 1) + 1, & 0 \le i < |S| \\ I_b(i) = C(S[i]) + Occ(S[i], I_b(i+1)), & 0 \le i < |S| \end{cases}$$



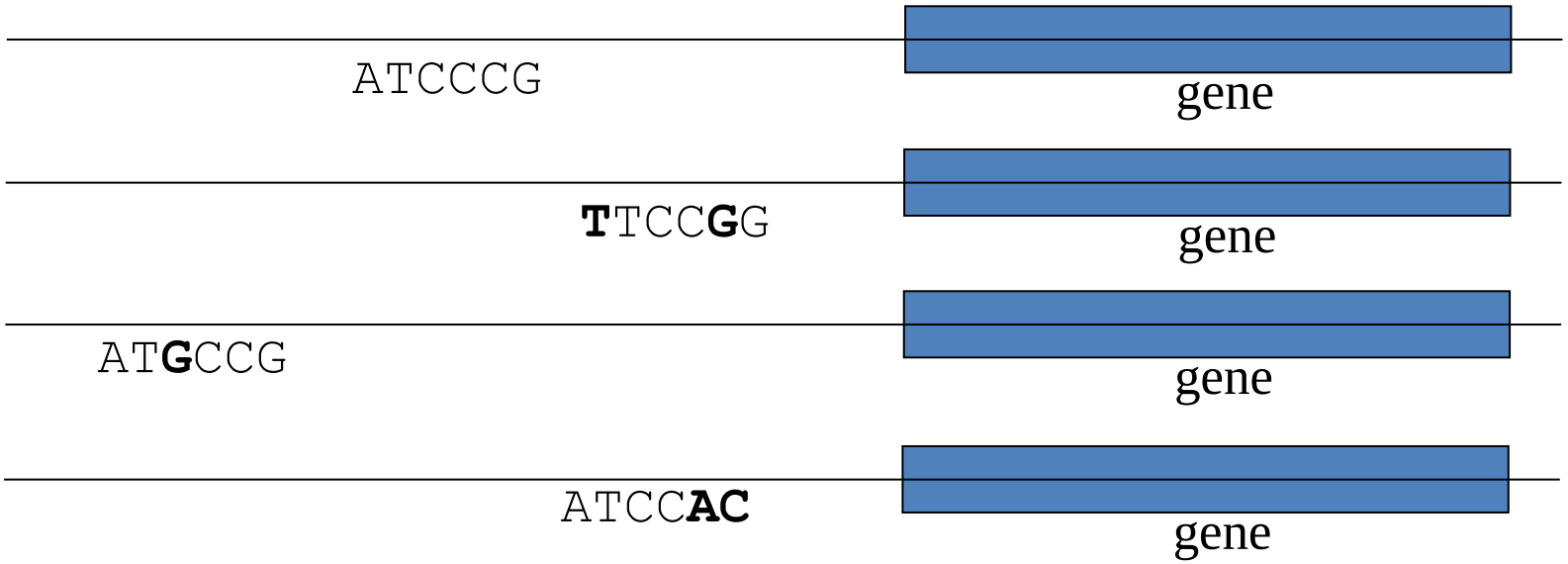


CUSHAW: short-read aligner to human genome based on BWT

	SRR002273 8.5M Reads, 36bp	ERR000589 24.3M Reads, 51bp	ERR002273 20.4M Reads, 75bps
CUSHAW (Tesla M2090)	1.1 mins	5.5 mins	33.3 mins
BWA 0.5.9 (AMD quad-core CPU, 4 threads)	14.9 mins	67.6 mins	91.9 mins
Speedup	12.8	12.2	2.8

- CUSHAW becomes less efficient with growing read length
- Reference
  - Y. Liu, B.Schmidt, D. Maskell: *CUSHAW: a CUDA compatible short read aligner to large genomes based on the Burrows-Wheeler transform*. **Bioinformatics**, 2012, doi: 10.1093/bioinformatics/bts276

mCUDA-MEME: Motif finding

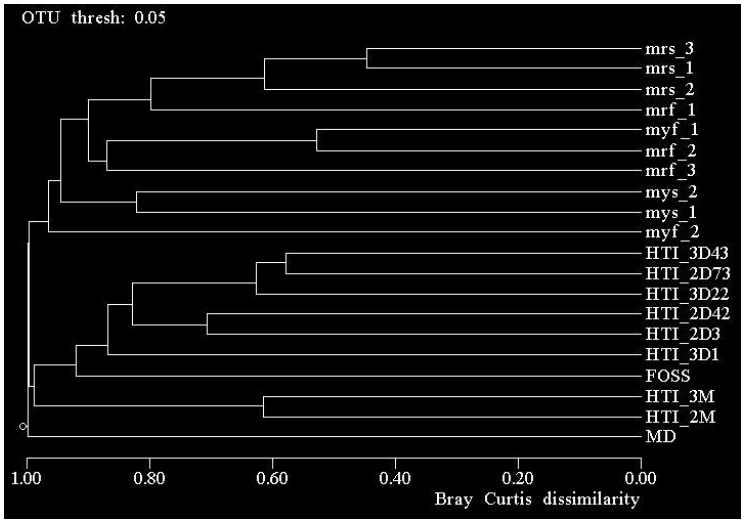


ChiP-Seq Dataset	mCUDA-MEME (4-node dual-S2050)		Parallel MEME (MPI)	Speedups	
	8 GPUs	1 GPU	32 CPU cores	8 GPUs	1 GPU
NRSF1000	6.5 mins	45.0 mins	43.6 mins	6.7	1.0
NRSF2000	27.7 mins	202.5 mins	230.5 mins	8.3	1.1

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

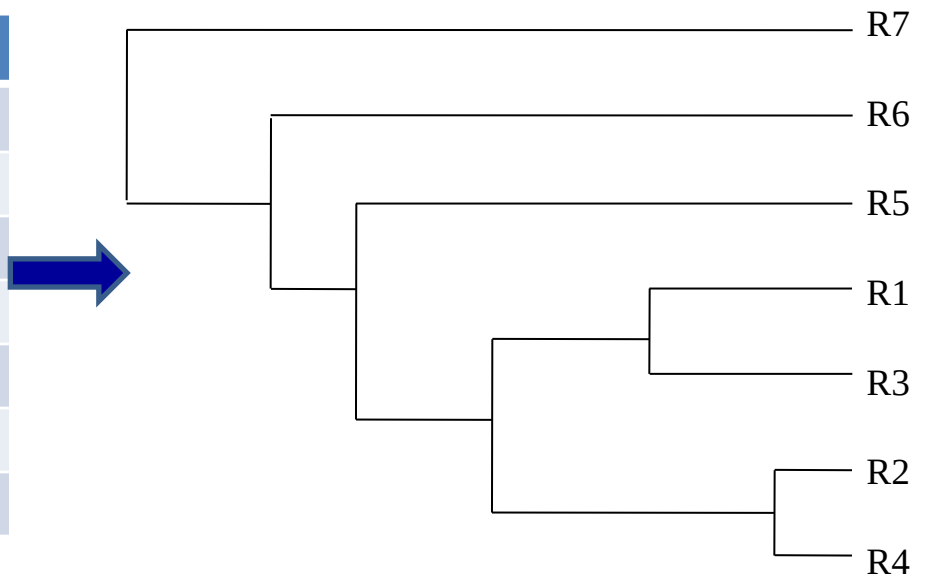
Taxonomic clustering of Metagnomic data

- 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 (454 sequencers)
  - Read length 200-600 bps
  - Number of reads 10K to 10M
- **Taxonomy-independent clustering approach**
  - hierarchical clustering from a distance matrix
  - Bin reads into **OTUs** (Operational Taxonomic Units) based on a distance threshold
- Bioinformatics challenges
  - Highly compute/memory-intensive
  - Scalability
  - Accuracy



CRiSPy-CUDA: Hierarchical Clustering

	R1	R2	R3	R4	R5	R6	R7
R1							
R2	21.1						
R3	32.9	19.7					
R4	20.7	39.0	20.4				
R5	11.0	9.8	10.3	9.7			
R6	9.3	8.6	9.6	8.4	7.0		
R7	7.1	7.3	7.5	7.4	7.3	4.3	

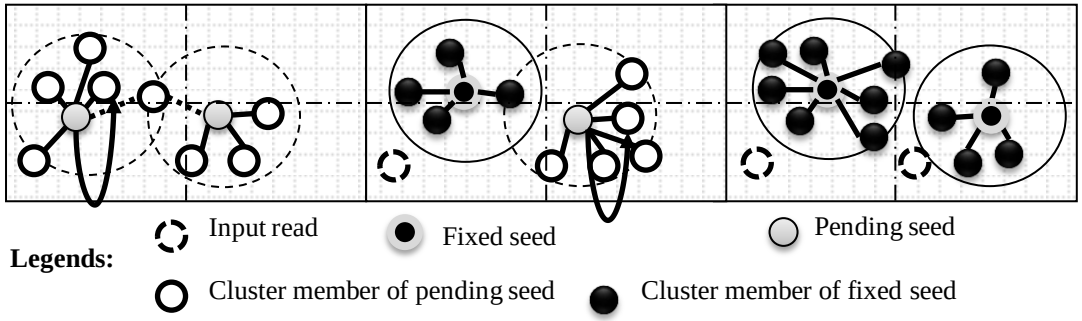


- Time: $O(n^2 \cdot l^2)$, Space: $O(n^2)$
- Techniques to reduce time and space complexities

1. Filtration using *k*-mer distance before genetic distance
2. Sparse Matrix Representation of distance matrix using thresholds
3. Space-efficient hierarchical clustering implementation using sorting

Dataset	#cleaned reads (length)	ESPRIT (*sequential runtime*)	CRiSPy-CUDA on C2050 (*speedup*)
SRR029122	15K (239bps)	2.8 hours	68
SRR013437	22K (267bps)	8.0 hours	78
SRR064911	17K (524bps)	3.3 days	94

DySC: Improved Greedy Short-Read Clustering



- Classical greedy sequence clustering
  - Cluster represented by one sequence (***seed***)
  - Unseen read is aligned to all seeds
  - If all alignments return distances above a threshold, the read is used as the seed of a new cluster. Otherwise, the read is added to the nearest clusters
- DySC (Dynamic Seed-based Clustering)
  - delays production of fixed clusters through usage of pending clusters

Tool	SIM	NMI-score (gut microbiome)	
		V2	V6
UCLUST	97%	0.73	0.64
CD-HIT		0.68	0.64
ESPRIT-Tree		**0.78**	0.68
DySC		**0.78**	**0.78**

Sequential Runtime: 4-Level Clustering of SRR069029 (2.7M Reads, 125 bps)			
UCLUST	**CD-HIT-EST**	**ESPRIT-Tree**	**DySC**
3.0 hours	3.8 hours	26.2 hours	**2.2 hours**

Z. Zheng, S. Kramer, B.Schmidt: *DySC: Software for Greedy Clustering of 16S rRNA Reads*. **Bioinformatics**, accepted

Conclusion

- NGS technologies establish the need for scalable Bioinformatics tools that can process massive amounts of short reads
- Parallel computing is a highly suitable technology to address this need
- NGS algorithms often need to be adapted since throughput and read length continues to increase
- Website
  - http://hpc.informatik.uni-mainz.de/

JG|U
JOHANNES GUTENBERG UNIVERSITÄT MAINZ