Extreme-scale Protein Similarity Search
PASTIS (ExaBiome/ExaGraph)

GPUs for Science, NERSC

Oct 25, 2022

Oguz Selvitopi (AMCR, LBNL)

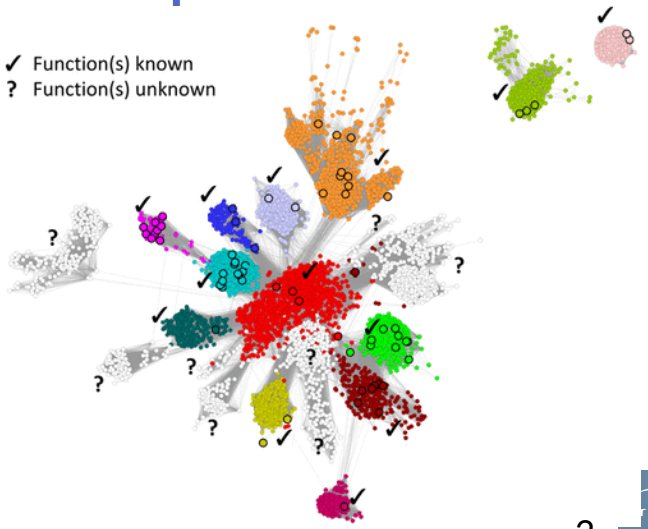
Protein similarity search & clustering

▷ Comparative genomics

- Functional or taxonomic contents of collected samples

▷ Similarity search between two sets of protein sequences

- Functional annotation
- Gene localization
- Studying protein evolution



▷ Common use case

- Protein similarity networks
- Detection of protein families

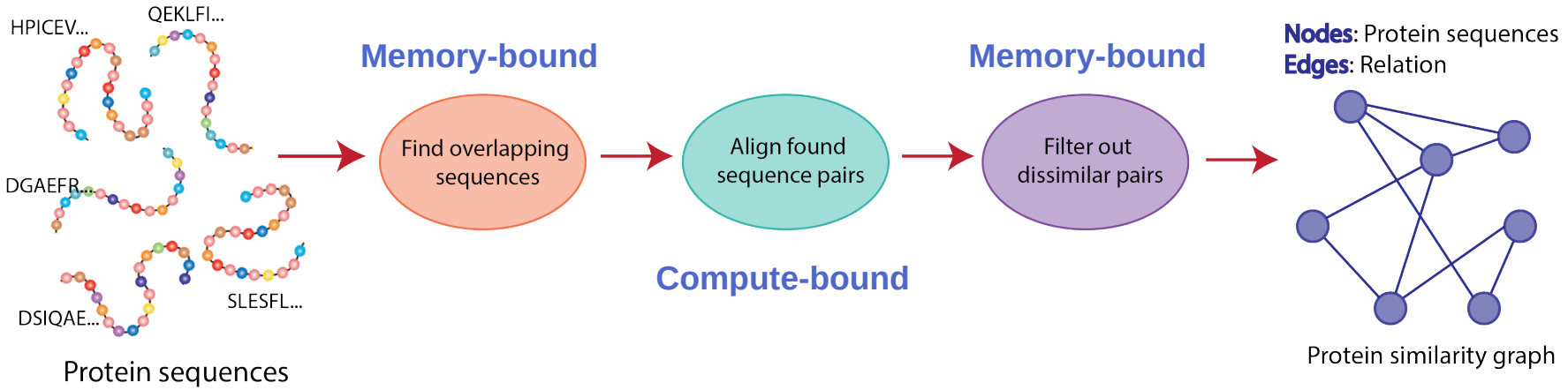
Motivation and Goals

- Distributed many-against-many protein similarity search
  - Parallel protein similarity search
  - Clustering
  - Metagenomics
    - Days, weeks, or even months

- **Existing software**: MMseqs2, LAST, DIAMOND, BLASTP
  - Optimized for shared-memory

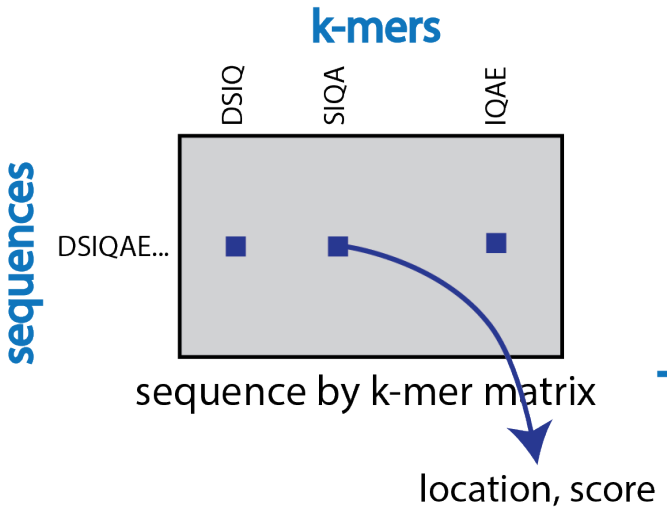
- **PASTIS:** Protein Alignment via Sparse Matrices for **SEARCH**
- **HipMCL:** High Performance Markov Clustering for **CLUSTERING**

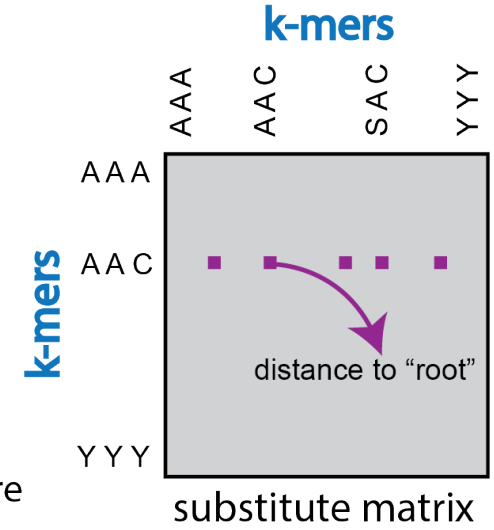
PASTIS Workflow

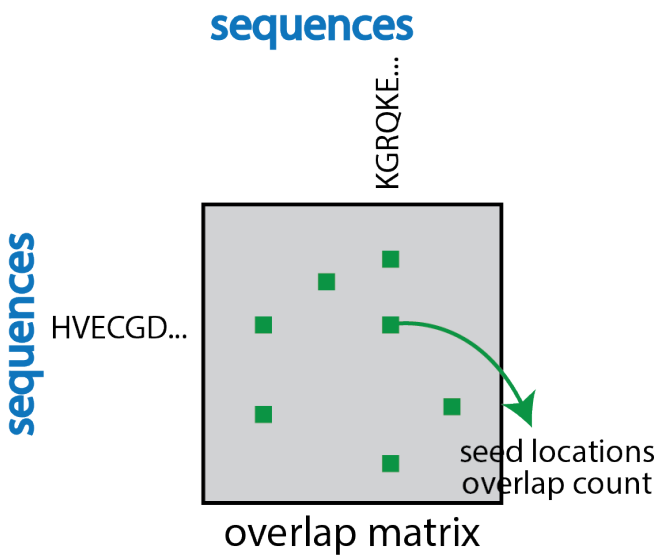


- Fixed size k-mers to discover overlapping sequences
  - Exact, substitute variants
- Large-scale **parallelism** via **CombBLAS** (distributed sparse matrix library)
- **Alignment** via external libraries (on-node)
- **Optimizations** for load balancing, memory management, overlapping, etc4

Sparse matrices in PASTIS

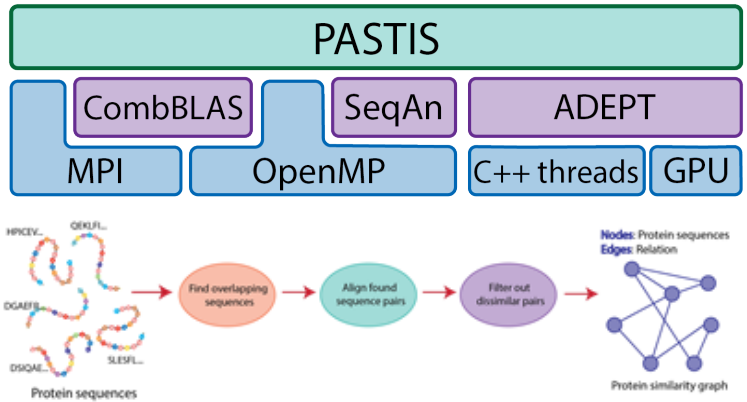






PASTIS: Parallelism

- Scalable distributed-memory implementation of the components
  - Hybrid MPI-OpenMP
- **Libraries** used in PASTIS
  - **CombBLAS**
  - **SeqAn/ADEPT**
- PASTIS itself also makes use of MPI-OpenMP



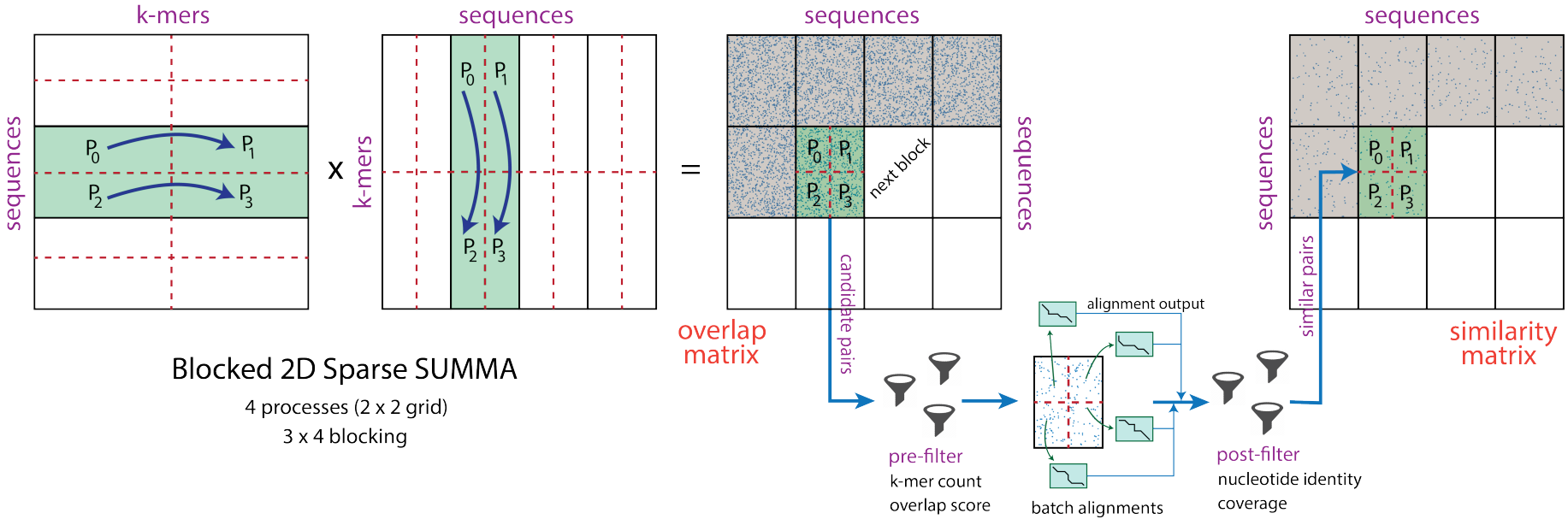
Challenges of similarity search on huge datasets

- **Computational patterns**
  - Search operations on sequences
  - Pairwise alignments
    - Edit distance computations
- **Memory requirements**
  - Many-against-many search
  - Example: 100 million sequences
    - Candidate pairs ~ **trillions (10^{12})**
    - Alignments ~ **hundreds of billions (10^{11})**
    - Similar pairs ~ **billions (10^{9})**

Techniques

- **Blocked (incremental) formation** of the similarity graph
  - Blocked 2D Sparse SUMMA
- **Load balancing**
  - Index-based
  - Triangularity-based
  - Interleaved with blocked formation
- **Pre-blocking**
  - GPU accelerators
  - Utilize all resources on nodes simultaneously

Blocked 2D Sparse SUMMA



Blocked 2D Sparse SUMMA

4 processes (2 x 2 grid)

3 x 4 blocking

Advantages

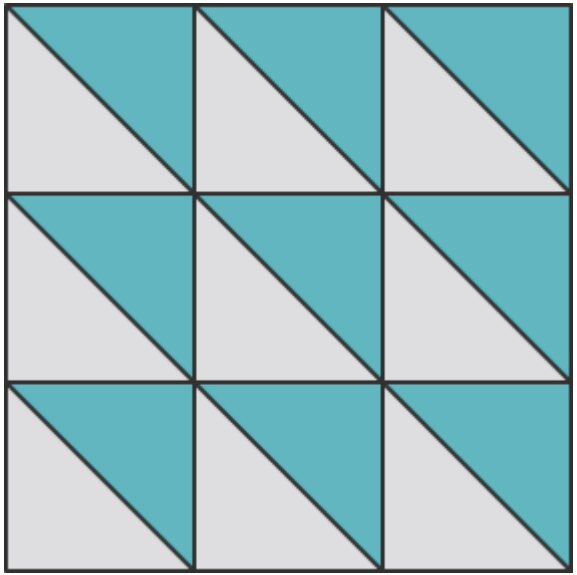
- Bounds the maximum memory utilization
- Enables in-memory search for huge datasets
- Opens up the path for several further optimizations

Disadvantages

- Increases time compared to doing all at once
- Increased communication

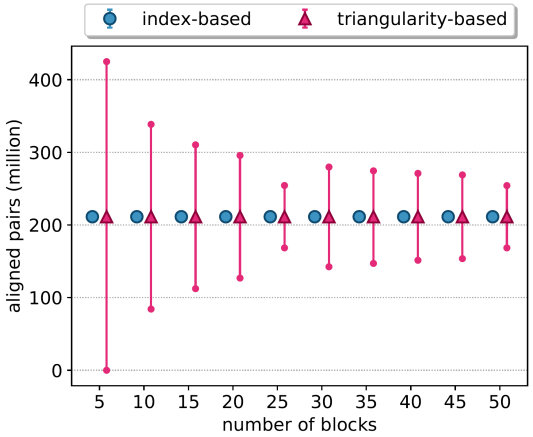
Load balancing

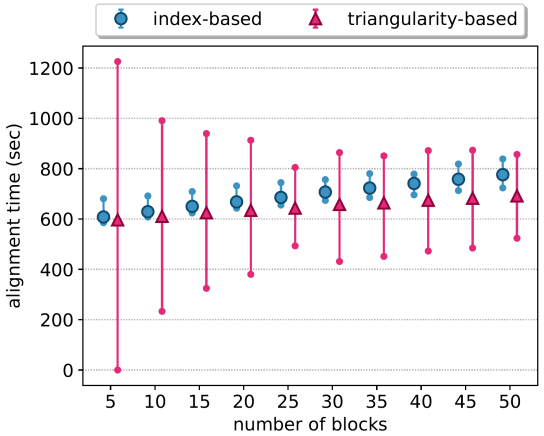
- **Symmetricity** of the overlap matrix
  - Can avoid alignments + perhaps sparse matrix computations
  - **Important** → Alignments are expensive
- How to achieve load balancing in blocked overlap detection?
- Approach #1: Symmetricity based on **triangularity**
- Approach #2: Symmetricity based on **indices**

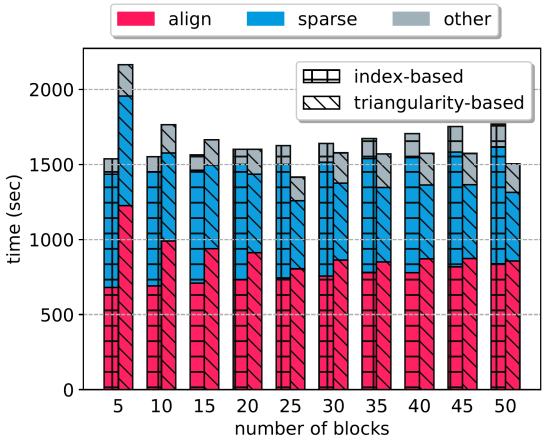


half of the processes would stay idle if used in blocked multiplication

Load balancing comparison







Index-based scheme has better load balance

Triangularity-based method can save from computations

Pre-blocking

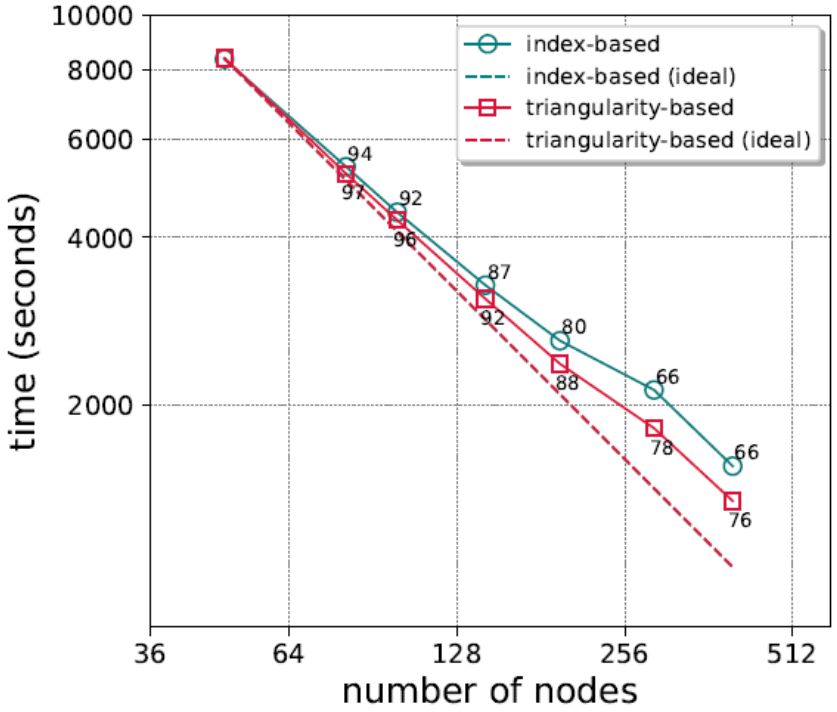
- **Incremental formation of the similarity graph**
  - Alignments → on GPUs
  - Sparse computations & other → on CPUs
- CPUs stay idle
  - Can go ahead and prepare the next batch(es)
- **Hide the overhead of sparse computations**
  - Distributed sparse computations
  - **Avoid collective communication + memory-bound low-intensity computations**

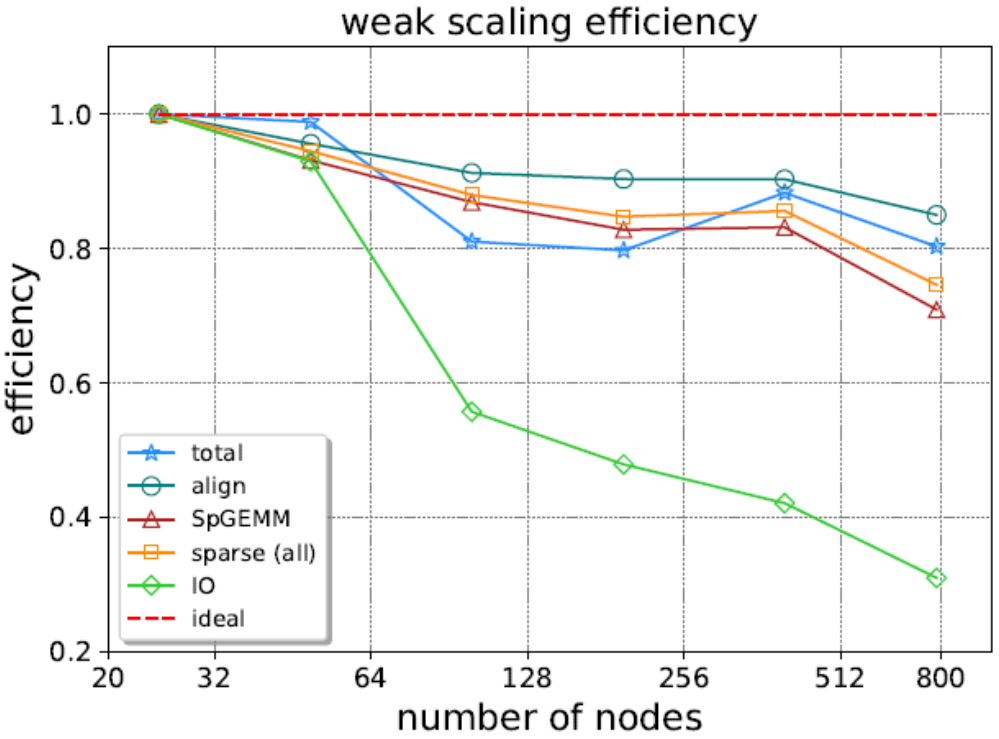
Runtime (sec)

#blocks	Plain	Pre-blocking	Total%
10	1555	1090	**70**
20	1606	1123	**70**
30	1659	1163	**70**
40	1724	1203	**70**
50	1774	1245	**70**

Overlap efficiency: >95%

Strong scaling & weak scaling

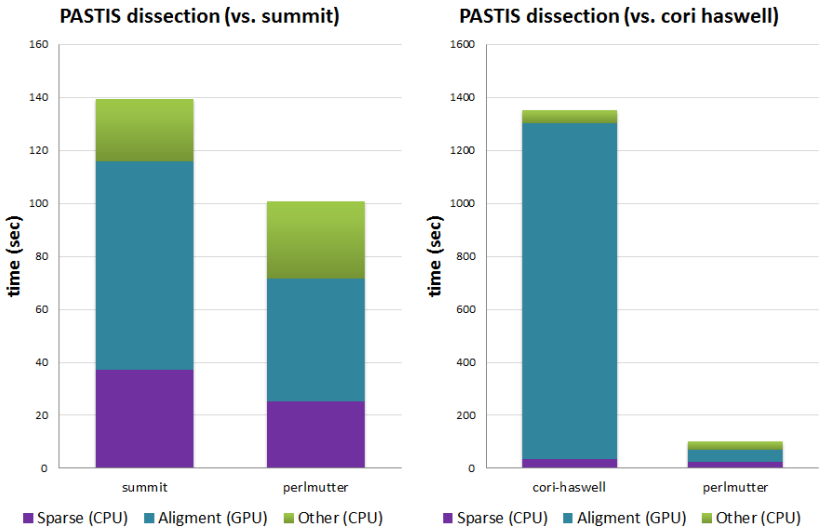




Perlmutter porting

- PASTIS was already running on Summit
  - Smooth porting
- **Remarks**
  - PASTIS needs and uses both CPU and GPU resources
  - PASTIS benefited from a faster CPU as well as faster GPUs on a node on Perlmutter

Small-scale performance
64 nodes



Protein similarity search at scale

- **Summit**
  - 4600+ nodes
  - 42 CPU cores (512 GB)
  - 6 GPUs (V100) (16 GB)
- **Dataset size**
  - 313 million (April)
  - 405 million (July)
- **Test scale**
  - 2025 nodes
  - 3364 nodes

- **Perlmutter**
  - 1500+ nodes
  - 64 CPU cores (256 GB)
  - 4 GPUs (A100) (40 GB)
- **Dataset size**
  - 157 million
  - 200 million (July)
- **Test scale**
  - 1024 nodes

Summit runs

Alignments per second: 320 million → 691 million
Cell updates per second: 143.9 TCUPs → 176.3 TCUPs

Problem size: ~1.67x
Run size: ~1.66x

- **2025 nodes**
- **313 million sequences**
- **Parameters**
  - Blocking factor: 14 x 14
  - Load balancing: triangularity
  - Pre-blocking
- **Results**
  - Discovered candidates: **53T**
  - Performed alignments: **4.5T**
  - Similar pairs: **215B**
  - Output size: **5.4 TB**
  - **3.89 hours**

- **3364 nodes**
- **405 million sequ**
- **Parameters**
  - Blocking f
  - Load b ularity
  - Pr
- **R**
  - D candidates: **96T**
  - Perfo ed alignments: **8.6T**
  - Similar pairs: **1.1T**
  - Output size: **27 TB**
  - **3.44 hours**

Gordon Bell finalist at SC22!

Perlmutter runs

- **1024 nodes**
- **200 million sequences**
- **Parameters**
  - Blocking factor: 20 x 20
  - Load balancing: triangularity
  - Pre-blocking
- **Results**
  - Discovered candidates: **18T**
  - Performed alignments: **1.6T**
  - Similar pairs: **76B**
  - Output size: **1.5 - 2TB**
  - **2.48 hours**
  - Excluding IO: **2.01 hours**

	Alignments per sec (per node)	
	Summit	Perlmutter
Overall	205294	173325
Excluding IO	218809	213844

	Summit	Perlmutter
Sparse (candidate disc.)	7427	5539
Alignment	10168	6402
IO%	5%	19%

1.37x

1.16x

Problem size comparison

- Tools with distributed memory search option
  - MMSeqs2, DIAMOND
- **DIAMOND (reported in 2021)**
  - 281 million sequences vs. 39 million sequences
  - 520 nodes (Cobra at Max Planck Society)
    - ~23 billion alignments
    - 5.42 hours (very sensitive) - 17.77 hours (ultra sensitive)
- **Alignment rate** **575.5x**
  - **1.2 million** alignments per sec vs. **690.6 million** alignments per sec
  - Per node: 2.3k vs. 205k
- **Search space** **15.0x**
  - **281 x 39 x 10^{12}** vs. **405 x 405 x 10^{12}**

Conclusions

- **Compute-intensive** and **huge memory footprint**
  - Accelerators for alignments
  - Algorithmic techniques for staying in-memory
    - No intermediate IO
  - Optimizations to take advantage of heterogeneous node architecture
    - Use all resources on the node
- **HPC for bioinformatics**
  - Cutting time-to-solution from days/weeks to hours

Thank you

QUESTIONS

Attend our talk at SC22!