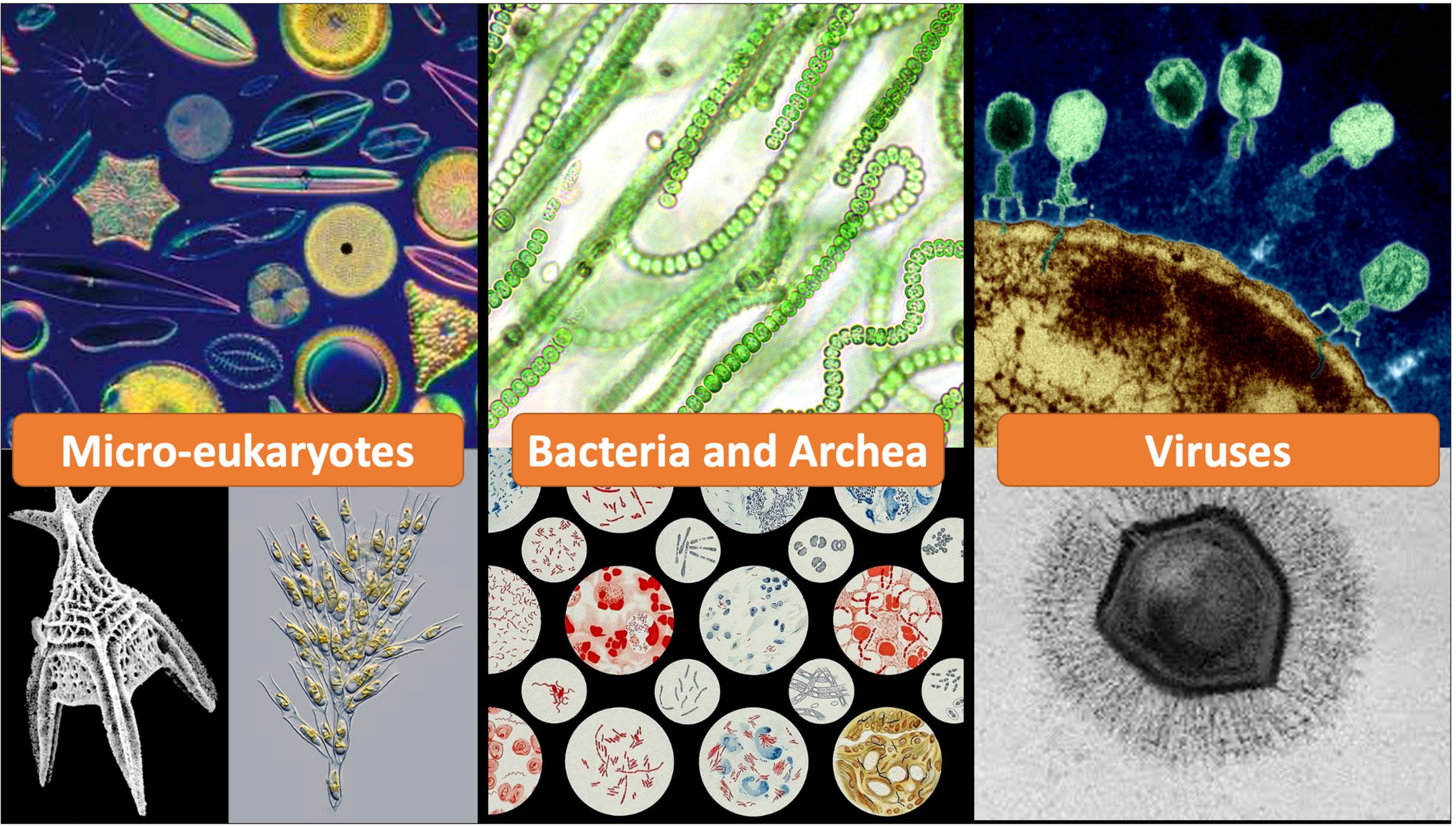
Hewlett Packard Enterprise

Exploring Suffix Array Algorithms in Chapel

Michael P. Ferguson, Bonnie Hurwitz, Shreyas Khandekar

PAW-ATM 2024
November 17, 2024

What is a microbiome?



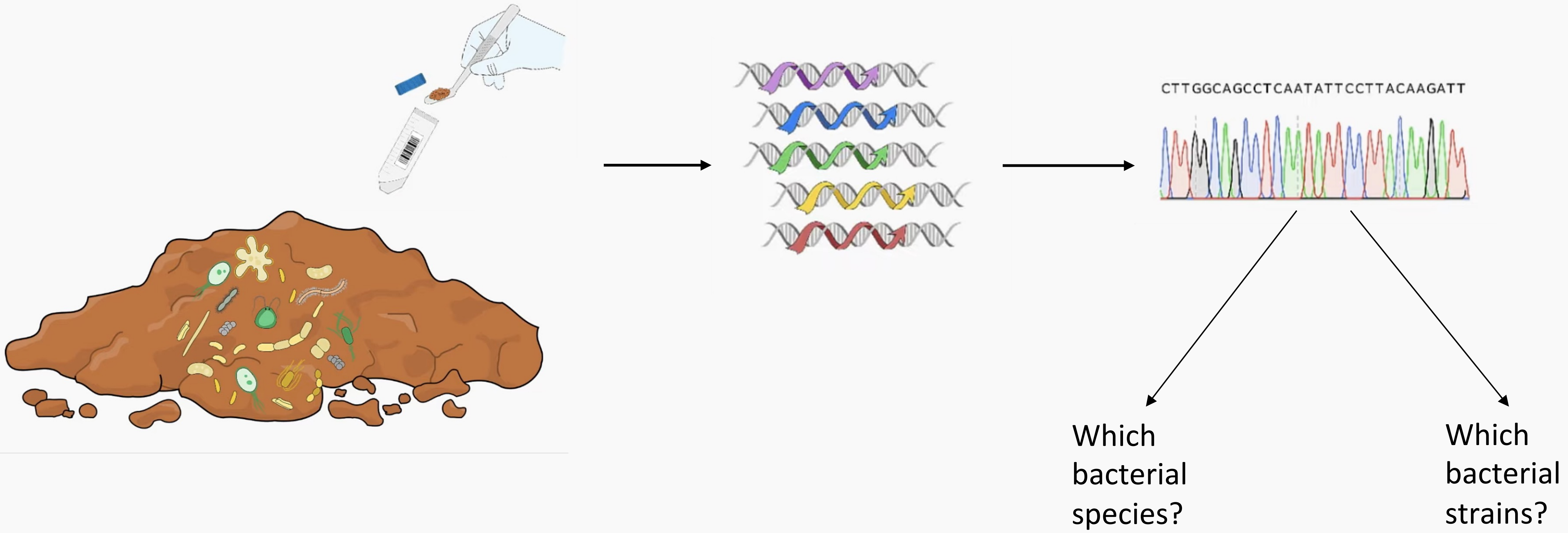
A **microbiome** is a complex mixture of microorganisms that reside in a specific environmental niche.

OHMI: https://doi.org/10.1186/s13326-019-0217-1

The human microbiome is an essential organ whose functions are central to human health

Microbial metagenomics allows us to explore the complex interaction between microbiota and host health

@thescientist



Metagenomics is the genomic analysis of microbial communities by extracting and sequencing their DNA. Metagenomics allows to study complex communities of microorganisms directly in their natural environment

Computational Approach

- Gather lots of genomes for known bacteria

- Analyze these genomes to find k-mers that identify a particular microbial strain
  - A k-mer is just snippet of the DNA sequence of length k; e.g. ACGTTATA is an 8-mer
  - An identifying k-mer should ideally only be present in the genome of a single microbial strain

Focus of this talk

- Search for these identifying k-mers within metagenomic samples to learn about microbial communities

Suffix Array

- Suffix Arrays are used in several bioinformatics applications including bowtie2 and MUMmer4 [1,2]
- We are exploring how suffix arrays can help to identify subsequences for strain-level detection
- What is a suffix array?

Suffixes of 'seeresses'

Suffix	Offset
seeresses	0
eeresses	1
eresses	2
resses	3
esses	4
sses	5
ses	6
es	7
s	8

Sorted Suffixes

Suffix	Offset
eeresses	1
eresses	2
es	7
esses	4
resses	3
s	8
seeresses	0
ses	6
sses	5

The Suffix Array

The Chapel Parallel Programming Language

Chapel is a language designed for productive parallel programming, particularly on large-scale systems. Chapel is ...

Easy to Use	"*We ask students at the master's degree to do stuff that would take 2 years and they do it in 3 months.*" Eric Laurendeau, Professor of Mechanical Engineering
Portable	HPE Cray EX, HPE Apollo, Cray XC, *nix systems, Mac, NVIDIA and AMD GPUs
Fast & Scalable	Achieved 8,500 GiB/s when sorting 256 TiB in 31 seconds on 8192 HPE Cray EX Nodes
GPU-Ready	Real-world applications were ported on GPUs with few changes, and run on leadership-class systems such as Frontier and Perlmutter
Open source	Team at HPE actively interacts with Chapel community at chapel-lang.org

What did we do?

- We implemented several tools to explore how suffix arrays can be applied to strain detection

- Suffix Array Construction in Chapel
  - Implemented the Difference Cover algorithm [3] focusing on v=133
  - Also implemented parallel sparse PLCP array construction [4,5]

- Computing Similarity based on the Suffix Array
  - Use case: detect reference genomes that are too related to be useful for strain identification
  - Explored & implemented several parallel algorithms to compute all-to-all similarity for a set of genomes

- Finding Unique Substrings based on the Suffix Array
  - Use case: finding identifying k-mers for strain-level detection
  - Implemented a parallel algorithm to find all substrings that appear in just one genome
  - If no unique substrings can be found for a genome, uses similarity computation to drop near-duplicate genomes

Experience Report: Productivity

- We were able to implement all 3 tools in less than a month

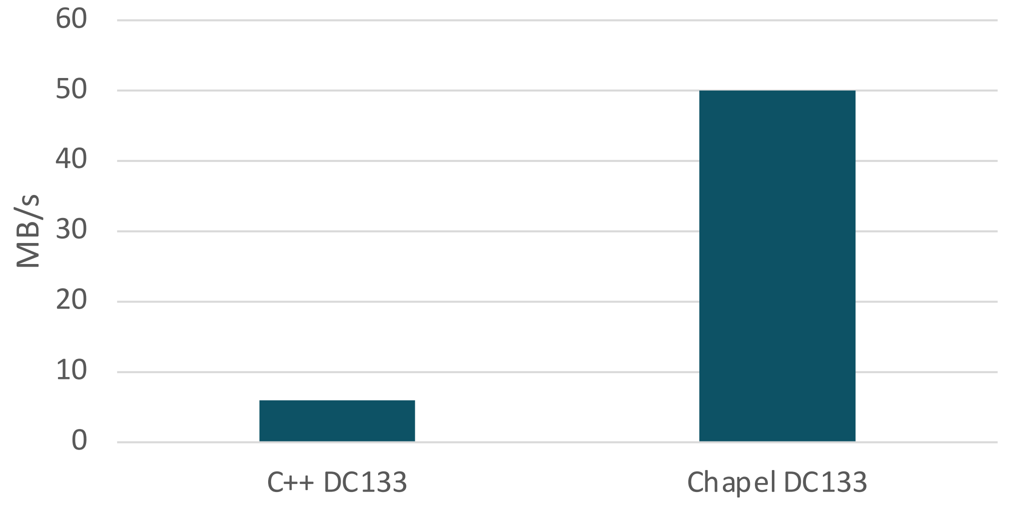
- We can compare the suffix array construction code against an earlier C++ implementation [6]
- Chapel version is about 2000 lines while the C++ implementation is closer to 8000 lines
  - Both implement the Difference Cover algorithm & have the same primary author
  - C++ implementation was parallel with MPI + OpenMP & optionally an external memory algorithm
  - The Chapel version is parallel and distributed (in-memory only)
  - Improvements are needed in both cases to achieve good scaling in distributed memory

- Challenges: hit occasional bugs, tricky to identify and fix problems with distributed memory performance
- Benefits:
  - High level parallel loops with ‘forall’ help with straightforward and portable code
  - Generic programming to make it easier to implement
    - Difference covers that are compiled in based on ‘param’
    - When switching some arrays to be distributed, existing functions can accept & work with the distributed arrays

Experience Report: Performance of the Suffix Array Construction

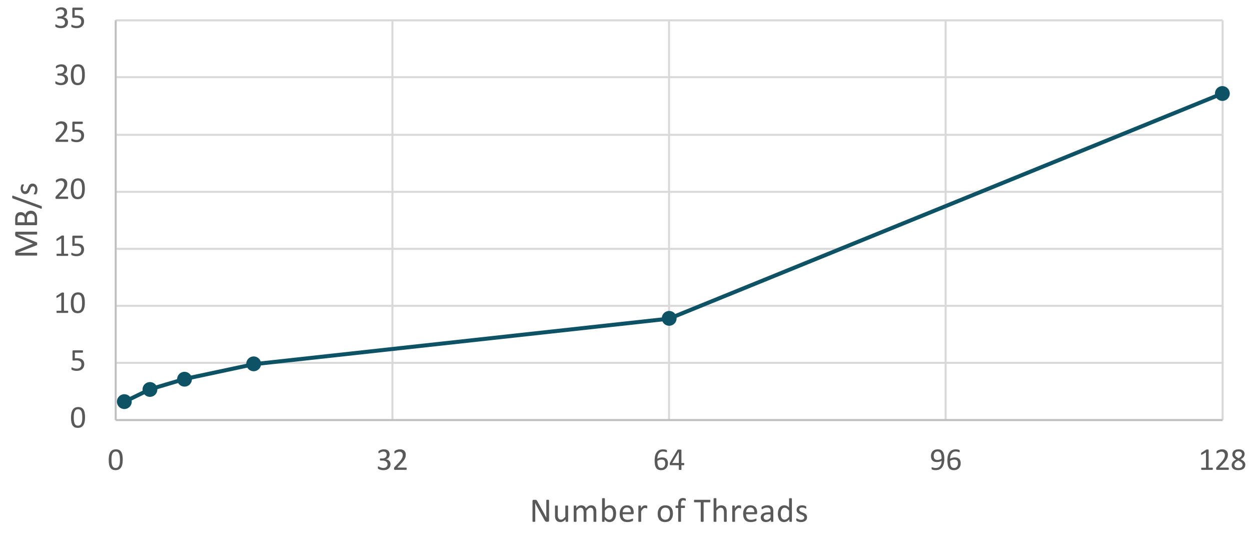
This chart compares the performance of the C++ and Chapel versions running in-memory on a single node

- On AMD Ryzen 9 7950X, 32 threads
- The C++ in-memory version is mostly serial
- Chapel version is parallel throughout



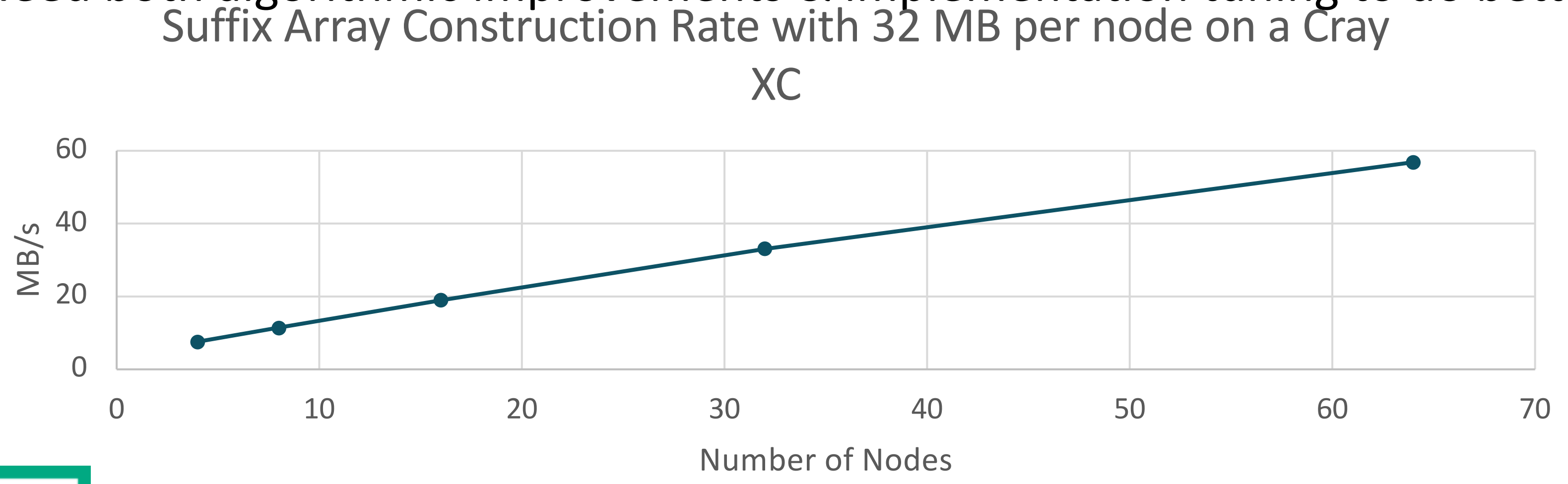
This chart shows single-node strong scaling

- On AMD EPYC 7543P 32-core processor
- Memory-intensive & can hit memory bottlenecks



Experience Report: Distributed Performance of the Suffix Array Construction

- Spent only a week making suffix array construction run multi-node
- Starting to use distributed arrays and run distributed was easy
- Trickier part was to identify and resolve performance issues
- Found the ‘local’ block in Chapel useful for identifying unexpected communication
  - Needed to arrange to do communication in a separate region for this strategy to work
- Achieved some scalability for a version with minimal random access (similar to external memory version)
  - This version is slower single-node & more memory intensive but reduces communication multi-node
  - Need both algorithmic improvements & implementation tuning to do better



Conclusion and Future Work

- This talk has shown some early results from our efforts
- We are using Chapel to implement tools to support strain-level metagenomic analysis

- We feel that Chapel has been particularly useful
  - for algorithmic exploration
  - to easily write parallel code
  - to easily move to distributed memory

- Scalable distributed-memory suffix array construction needs more work

- We are working towards publishing about our new tools in a bioinformatics journal

- Code is open-source & available at https://github.com/femto-dev/femto/tree/main/src/ssort_chpl

References

[1] B. Langmead, S. Salzberg, “Fast gapped-read alignment with Bowtie 2,” in *Nat Methods* vol 9, pp 357–359, 2012, doi: 10.1038/nmeth.1923

[2] G. Marçais, A. L. Delcher, A. M. Phillippy, R. Coston, S. L. Salzberg, and A. Zimin, “MUMmer4: a fast and versatile genome alignment system,” in *PLoS computational biology*, vol 14 no 1, 2018

[3] J. Kärkkäinen, P. Sanders, and S. Burkhardt, “Linear work suffix array construction,” in J. ACM vol. 53, no. 6, pp 918–936, Nov. 2006, doi: 10.1145/1217856.1217858

[4] J. Shun, "Fast parallel computation of longest common prefixes," SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, New Orleans, LA, USA, 2014, pp. 387-398, doi: 10.1109/SC.2014.37

[5] Juha Kärkkäinen, Giovanni Manzini, and Simon J. Puglisi. 2009. Permuted Longest-Common-Prefix Array. In Proceedings of the 20th Annual Symposium on Combinatorial Pattern Matching - Volume 5577. Springer-Verlag, Berlin, Heidelberg, 181–192.

[6] M. P. Ferguson, “FEMTO: fast search of large sequence collections,” CPM 2012: Combinatorial Pattern Matching 2012, in Lecture Notes in Computer Science, vol 7354, doi: 10.1007/978-3-642-31265-6_17 . Implementation available at https://github.com/femto-dev/femto/