FastIBS: Supplementary Documentation

# Overview

FastIBS is a high-performance toolkit designed for computing identity-by-state (IBS) distances and conducting related genomic analyses. It features an optimized C++ backend and supports containerized deployment via Docker and Singularity, enabling straightforward installation and seamless portability across diverse computing environments. This design is particularly well-suited for high-performance computing (HPC) systems, which are commonly used in this domain due to the large size of genomic datasets and the significant computational demands associated with their analysis, including requirements for storage, processing power, and memory.

FastIBS leverages the KMC API to implement IBS distance computation in a manner closely aligned with IBSpy, the current de-facto workflow for this type of analysis. However, unlike IBSpy, which suffers from significant memory bottlenecks that hinder parallelism, FastIBS provides a highly efficient and scalable C++ implementation. Initial benchmarks show that FastIBS can deliver over 100× speed-up compared to the base IBSpy pipeline, depending on underlying hardware and level of parallelism.

# Installation and Usage Guide

The following is an installation and usage guide for FastIBS. For further details and access to source files and job scripts, FastIBS is hosted at: <https://github.com/githubcbrc/FastIBS>

# Requirements

FastIBS is designed to run on Linux-based environments and relies on the following tools:

* **Docker**: Required for building the FastIBS environment. If Docker is not available on your HPC cluster, you can install it locally ([Docker installation guide](https://docs.docker.com/engine/install/)), set up FastIBS on your machine, and then transfer the installation directory to your cluster's home directory.
* **Singularity**: Necessary for generating the .sif container image, which enables smooth deployment on HPC systems. Installation instructions can be found in the [Singularity documentation](https://docs.sylabs.io/guides/3.0/user-guide/installation.html).
* **Bash**: The tool scripts are intended to be executed within a Bash shell environment.

**Note**: You do *not* need to install CMake or Ninja separately—both are included within the Docker image.

# Build & Install

1. Clone the repository:

mkdir <project-folder> && cd <project-folder>  
git clone https://github.com/githubcbrc/FastIBS.git .

2. (Optional) Edit Configuration in init/config.sh:

IMAGE\_NAME=fastibs\_img  
CONTAINER\_NAME=fastibs\_cont

3. Run the installation script:

bash install.sh

This script builds the Docker image, starts the container, compiles binaries, and produces a Singularity .sif image.

# Project Structure

Once the project is installed, its directory structure will look like:

project-root/

├── init/

│ ├── config.sh # Config variables (image/container names)

│ └── Dockerfile # Docker image setup

│

├── scripts/

│ ├── build\_img.sh # Builds Docker image

│ └── start\_cont.sh # Starts Docker container

│

├── src/ # C++ source code

├── build/ # CMake build directory (auto-generated)

├── bin/ # Compiled binaries (auto-generated)

├── compile.sh # CMake + Ninja build script

└── install.sh # Top-level installation script

# Output Binaries

After a successful build, the following executables will be available in /project/bin:  
• fastibs  
• fastibsmapper  
• KDBIntersect

# Running with **Docker**

To manually enter the running container:

docker exec -it <container\_name> bash

Or, using variables from the config:

source init/config.sh  
docker exec -it ${CONTAINER\_NAME} bash

# Running with Singularity

Once fastibs.sif is built (as part of running the installation script), you can test whether FastIBS tools have been properly installed:

singularity exec --bind .:/project fastibs.sif /project/bin/fastibs --help

singularity exec --bind .:/project fastibs.sif /project/bin/fastibsmapper --help

singularity exec --bind .:/project fastibs.sif /project/bin/KDBIntersect --help

If you want to use the tools do not forget to mount a data volume with a structure similar to the following:

FastIBSData/ # data volume root

├── FastIBS\_runs # results folder (initially empty)

│ ├── IG90747\_v\_TA1675\_50000.tsv # IBS distance report

│ ├── IG90747\_v\_TA1675\_genome.txt # mapping result

├── kmc\_sets # KMC database (A path to your KMC KBs)

│ └── IG90747

│ ├── IG90747.res.kmc\_pre

│ └── IG90747.res.kmc\_suf

└── reference # reference database

└── TA1675\_genome.fasta

Make sure to name your KMC sets and reference folders as indicated. To mount volumes, you can use the -v option with docker, or the --bind option with singularity.

## Clean Build

To remove all build artifacts and recompile FastIBS from scratch, use the following commands:

rm -rf build bin  
bash compile.sh

Alternatively, you can re-run the full installation pipeline:

bash install.sh

## IBS Distance Report

### Usage

/project/bin/fastibs <sourcePath> <referencePath> <resultsFolder> <windowSize>

### Example

/project/bin/fastibs /mnt/data/kmc\_sets/BW\_01002 /mnt/data/reference /mnt/data/FastIBS\_runs 50000

### Arguments

* <sourcePath>  
   Path to the folder containing the KMC dataset.  
   *Example:* /mnt/data/kmc\_sets/BW\_01002
* <referencePath>  
   Path to the folder containing reference genomes in FASTA format.  
   *Example*: /mnt/data/reference
* <resultsFolder>  
   Path to the folder where output results will be stored.  
   *Example:* /mnt/data/FastIBS\_runs
* <windowSize>  
   Length of the sequence window used for IBS calculation.  
   *Type:* Integer (e.g., 50000)

### Notes

* All specified folders should reside on a mounted data volume.
* Reference FASTA files may be gzip-compressed (i.e., .fasta.gz is supported).

#### Output

The output is a tab-delimited file containing IBS distance metrics for each analyzed window. The file includes the following columns:

1. seqname: The sequence name (typically the accession or identifier of the reference genome segment).
2. start: The start position (0-based) of the genomic window being analyzed.
3. end: The end position of the window (non-inclusive).
4. total\_kmers: Total number of k-mers generated from the reference window.
5. observed\_kmers: Number of k-mers found in the sample that match the reference set for that window.
6. variations: Number of variant sites (positions where the reference and sample differ within the window).
7. kmer\_distance: Computed IBS distance metric, reflecting the number of unique k-mers in the reference absent from the sample (or vice versa).

## KDB Reference Mapping: fastibsmapper

fastibsmapper computes a K-mer mapping of a given KMC base against given references, where each nucleotide position in the reference sequences is associated with a count of how many K-mers (of a fixed size, defined by the kmerSize of the KMC source) overlap that position and exist in the source KMC database.

A quick and easy k-mer mapping can be used to detect regions of high conservation/divergence from a given reference. Even though a KMC database does not retain positional information and only records k-mers and their frequencies from reads, the probability of false positives is small as long as sufficiently long k-mers are used (we use a default of 31 for kmerSize).

#### Usage

/project/bin/fastibsmapper <sourcePath> <referencePath> <resultsFolder>

#### Example

/project/bin/fastibsmapper /mnt/data/kmc\_sets/sample1 /mnt/data/reference /mnt/data/FastIBS\_runs

#### Arguments

* <sourcePath>  
   Path to the folder containing KMC database files.  
   *Example:* /mnt/data/kmc\_sets/<dataset\_name>
* <referencePath>  
   Path to the folder containing reference genomes in FASTA format.  
   *Example:* /mnt/data/reference
* <resultsFolder>  
   Destination folder where the mapping result files will be written.  
   *Example*: /mnt/data/FastIBS\_runs

#### Notes

* All input folders must be located on a mounted data volume.
* The tool automatically scans <referencePath> for .fasta files and processes each one against the KMC dataset found in <sourcePath>.
* Output file names follow the pattern:  
   <KMC\_prefix>\_v\_<reference\_stem>.txt
* Any existing output (mapping results) files will be skipped allowing for smooth re-runs..
* Errors encountered during processing are logged to a file named log.txt in the output directory.

#### Output

The tool computes a K-mer mapping for each reference genome. For every nucleotide position in the reference sequences, the tool outputs an integer matching score from 0-kmerSize. The resulting output can be used for further downstream analysis or visualization of sequence coverage and identity between sample and reference.

## KDB Intersection Size Tool: KDBIntersect

This tool computes the intersection size between two KMC databases, determining how many k-mers from the first database are present in the second.

#### Usage

/project/bin/KDBIntersect <kDB1Path> <kDB2Path>

#### Example

/project/bin/KDBIntersect /mnt/data/kmc\_sets/db1 /mnt/data/kmc\_sets/db2

#### Arguments

* <kDB1Path>  
   Path to the **first** KMC database directory.  
   *Example:* /mnt/data/kmc\_sets/db1
* <kDB2Path>  
   Path to the **second** KMC database directory.  
   *Example*: /mnt/data/kmc\_sets/db2

#### Notes

* Both KMC databases must use the same k-mer size.
* For performance optimization, the tool chooses the smaller database for random access during comparison.

#### Output

#### The output is a single integer, which represents the number of shared k-mers between the two databases.

## Running FastIBS on HPC Environments

A bash script (submit\_jobs.sh) is provided in the github repo for automating the generation and submission of multiple SLURM jobs. The script handles the parallel execution of the fastibs program for each accession in a list.

### Input Arguments:

* accessions\_file: A text file containing accession identifiers (one per line).
* data\_path: Path to the data directory.
* window\_size: Window size used for IBS distance computation.

### Functionality:

* For each accession listed in accessions\_file, a SLURM job script is dynamically generated. This job script sets job parameters, such as memory allocation, CPU cores, and runtime limits. It also calls the run.sh script to process the FastIBS program for that particular accession.
* Once the job script is generated for each accession, it can be submitted to the SLURM scheduler for parallel execution. (The submission line is currently commented out for safety, but can be enabled by uncommenting the line sbatch $job\_script.)
* fastibs also utilizes the thread\_pool library to achieve multithreading within the job by chunking reference sequences and processing them in parallel.

### Example Usage:

bash submit\_jobs.sh accessions.txt ./FastIBSData/ 50000

### Notes:

* Ensure job script parameters suit your use case and available HPC hardware / usage policy.
* This type of genomic analysis is highly I/O intensive. To maximize FastIBS performance, it's recommended to host your data volumes on high-speed storage systems. Consult with your cluster administrator to determine the optimal storage configuration that minimizes I/O bottlenecks and ensures efficient data throughput during analysis.

## Further Development

We intend to consolidate all existing functionality within one binary through parameterisation, review and extend our file format support including extensions to support other k-mer database formats such as Jellyfish and kmersGWAS, as well as extend the framework to provide visualisation capabilities.

## Conclusion

FastIBS is a high-performance tool for computing IBS distances and performing k-mer based mapping to a set of reference genomes. With an optimized C++ implementation and containerized deployment options, it is ideally suited for use in HPC environments. By leveraging the power of KMC and parallel processing, FastIBS offers a scalable, efficient solution for large-scale genomic data analysis.