scMSI Deployment Guide

Version: 1.0
Date Compiled: September 2024

Document Overview

This document provides intricate deployment procedures for scMSI, a sophisticated deconvolution model designed to estimate microsatellite instability at sub-clonal levels. This guide is intended for bioinformatics engineers and system administrators with expertise in high-performance computing (HPC) systems and large-scale genomics datasets.

System Environment Requirements

Operating System: CentOS Linux 8.4.2105 (Core), Ubuntu 20.04 LTS, or a compatible HPC environment with Slurm scheduler.
Kernel Version: 5.11.0-34-generic
Processor: AMD EPYC 7H12 64-Core Processor @ 2.60GHz, or Intel Xeon Platinum 9282
Architecture: x86_64
RAM: Minimum 512 GB
Disk Storage: Minimum 5TB SSD with NVMe for I/O-intensive operations
Virtualized Environment: Kubernetes, Docker Swarm (optional)
Supported CPU Features: SSE4, AVX2, AVX512, AVX512-VNNI, TSX, MPX, HTT

Dependency Software Packages

Python Libraries:
- numpy 1.21.0
- scipy 1.7.3 (compiled with Intel MKL for optimal performance)
- pandas 1.3.2
- scikit-learn 1.0.2
- matplotlib 3.4.2
- gurobipy 9.1.2

Additional Tools:
- bwa 0.7.17 (with pre-configured multi-threading support)
- samtools 1.15.1 (with HTSlib optimized for large-scale BAM file operations)
- bedtools 2.30.0 (compiled with OpenMP support)
- bcftools 1.13 (for high-throughput variant calling)

Installation Steps

1. **System Preparation**
   Ensure that the system is updated with the latest kernel optimizations for multi-threaded operations, including updates for OpenMP and AVX512 utilization:
   ```
   sudo apt-get update && sudo apt-get upgrade -y
   sudo yum update -y && sudo yum upgrade -y
   ```
   Install required compilers for Python and bioinformatics tools using dependency managers:
   ```
   sudo apt-get install build-essential gcc g++ cmake automake -y
   sudo yum groupinstall "Development Tools"
   ```

2. **Python Environment Setup**
   Set up an isolated Python environment using virtual environment management techniques, such as Conda with strict dependency resolution flags:
   ```
   conda create -n scMSI python=3.7 --strict-channel-priority
   conda activate scMSI
   ```

3. **Install Python Libraries**
   Install libraries compiled with high-performance computing libraries such as Intel MKL or OpenBLAS:
   ```
   conda install numpy==1.21.0 scipy==1.7.3 pandas==1.3.2 scikit-learn==1.0.2 matplotlib==3.4.2 -c intel
   ```

4. **Install Additional HPC-Optimized Tools**
   Install BWA, Samtools, and Bedtools with multi-threading and parallel I/O support:
   ```
   module load BWA/0.7.17
   module load SAMtools/1.15.1
   module load BEDtools/2.30.0
   ```

5. **Download and Configure scMSI Codebase**
   Clone the scMSI code repository and configure the source code for optimal parallel execution:
   ```
   ```

```
git clone https://github.com/SeqAnalysis/scMSI.git
cd scMSI
make -j64  # Utilize 64 cores for compilation
```

6. **Deployment Verification**
   Verify the deployment by running multi-threaded test executions:
   ```
   python main.py -t 64 --profile # Runs scMSI using 64 threads with performance profiling enabled
   ```

7. **Configuration**
   Configure HPC job scheduling with Slurm to automate parallel execution:
   ```
   sbatch --cpus-per-task=64 --mem=500G --time=48:00:00 --wrap="python main.py"
   ```

Performance Optimization

Users should leverage NUMA-aware CPU affinity settings to maximize parallelism:
```
numactl --physcpubind=0-63 --membind=0 python main.py -t 64
```
Additionally, fine-tune OpenMP thread settings for optimal parallelization of matrix operations:
```
export OMP_NUM_THREADS=64
export MKL_NUM_THREADS=64
export OPENBLAS_NUM_THREADS=64
```

Troubleshooting

Advanced debugging techniques include profiling with Valgrind or Intel VTune Amplifier to identify bottlenecks in large-scale data processing tasks. Additionally, error logs can be analyzed using custom scripts integrated with Logstash and Kibana for real-time monitoring of scMSI execution performance.