The PLA Information Engineering University

Team member: Tian Xiao YongNan Zhu JinHao Li

YunFei Zhao ZiXuan Cai

Team advisor: Peng Xu

January 21, 2025

Contents

Ⅰ Introduction to the university's activities in supercomputing

1. Supercomputing-related hardware and software platforms

Our school boasts state-of-the-art hardware and software platforms that

provide robust support for supercomputing research and competitions.

Hardware Platforms

Our hardware infrastructure includes:

- A high-performance computing cluster on the Sunway platform, with a peak performance of 10 petaflops.
- An X86 server cluster, comprising numerous IBM 3850 and IBM 3650 servers.
- Multiple Loongson servers.
- Several NVIDIA Tesla HPC cards.
- Additionally, we are equipped with advanced processors such as Intel KNC and KNL, among others.

Software and Research Facilities

In terms of software and research facilities, our school offers:

- Each laboratory is equipped with cutting-edge experimental facilities and supporting instruments, along with multiple sets of servers and PCs featuring diverse architectures.
- Our laboratories are staffed with renowned domestic and international experts who guide a large number of young and middle-aged teachers and students in their research.
- The school library provides quick and convenient access to information, supporting research and learning in supercomputing.

Collaborations

We maintain close partnerships with leading institutions in the field, including the Jiangnan Institute of Computing Technology, Wuxi Supercomputing Center, Jinan Supercomputing Center, Zhengzhou National Supercomputing Center, Chinese Academy of Sciences, and National University of Defense Technology, among others. These

collaborations enhance our capabilities and broaden our research scope.

In summary, our school provides an excellent research environment, supported by comprehensive and advanced hardware and software facilities. This infrastructure ensures strong support for both supercomputing research and participation in international competitions.

2. Supercomputing-related courses, trainings, and interest groups

We have a varity of courses emphasizing on HPC in our curriculum system, as listed in Table 1. Such courses may help lay a solid foundation for students interested in HPC, benefiting the students from both theoretical and practical aspects. With an in-depth study of the HPC courses, the team members have a deep understanding of the HPC from various aspects, including hardware architecture, algorithm design, parallel optimization, and engineering capabilities, etc.

Table 1 HPC Curriculum in our school

	Course	Level	Instuctor
1	Parallel programming	Undergraduate	Dr. Jinlong Xu
2	Advanced compiling technology	Undergraduate	Dr. Feng Yue
3	High performance computer architecture	Undergraduate	Dr. Huihui Sun
4	Data structure	Undergraduate	Prof. Fuding Liu
5	Principles of computer composition	Undergraduate	Prof. Xuyan Qi
6	Parallel algorithms design	Graduate student	Prof. Yingying Li

Supercomputer & AI practice innovation base

The project was established in 2016. It has joined our schools elective courses.

The project focuses on cultivating students practical ability and innovative spirit. The activities are both theoretical and practical in engineering, which can fully mobilize the enthusiasm of students. As long as the students are interested in HPC, they are provided with supports in all

aspects, including basic HPC courses and hardwares. It provides a platform for all students to participate in the HPC in our school.

Table 2 Project Schedule

Course Name		Time	Teacher
Getting Started			
1	Supercomputing and Artificial Intelligence Status and Development Introduction	2	Dr. Jinlong Xu
2	HPC cluster structures and performance test	2	Dr. Jinlong Xu
Supercomputer application program			
3	Application of Scientific Computing Program in HPC Platform	4	Dr. Jinchen Xu
4	Optimization theory and practice of scientific computing platform for HPC applications	8	Dr. Jinchen Xu
Supercomputing Cluster Design			
5	Heterogeneous integration of HPC computing cluster design study	4	Prof. Yingying Li
6	HPC Cluster Management and Maintenance	2	Prof. Yingying Li
Deep learning			
7	AI- Introduction to Deep Learning	2	Prof. Gang Zhou
8	AI- Deep Learning Application	6	Prof. Gang Zhou

Our students not only took full advantage of various public welfare and open supercomputer resources such as eHPC, HPC competitions and etc., but also spontaneously established HPC&AI practice innovation bases with teachers and actively participated in the HPC study. We warmly invite students of different majors backgrounds and different regions to participate in the HPC community. We hope that with the support from the national level , and most importantly, with our own enthusiasm and efforts, we can create our own better future.

3. Supercomputing-related research and applications

Our school team participated in the development of the Sunway series

supercomputers, and is responsible for the automatic parallelization system, he mathematical library and other subsystems of the research and development. We have also worked with China Southern Power Grid to build a high-performance computing cluster for power systems. We have a lot of supercomputing-related applications and research, in this no longer one by one list.

4. Key achievements on supercomputing research

- Our department have contributed to the Sunway series supercomputers since 1999, we are in charge of the fundamental system softwares including mathematical library, automatic parallelizing/vectorizing compilation frameworks and toolkits, etc., one of the resulting machines “Sunway TaihuLight” have won the world's TOP500 supercomputing championship for many consecutive times.
- Our school has achieved remarkable success in supercomputing competitions. Among them, five students won the first prize at the ASC International Supercomputing Competition. Additionally, 35 students were awarded the second prize. In the Intel Cup Parallel Application Challenge, our school's team achieved outstanding success by winning the gold medal for parallel optimization. In the previous year's Parallel Application Challenge, our team also demonstrated remarkable performance, securing the silver medal for parallel optimization, the Concurrent Fund Best Innovation Award, and the Concurrent Fund Best Network Popularity Award.

II Introduction of the Team

1. A brief description of the team setup

Our team consists of five members who all come from the same major. We are united by a shared passion for high - performance computing, forming an innovative and collaborative group. Since our formation, we have been looking forward to the ASC competition and ultimately decided to join hands to take on the challenge of ASC 2025.

During the preparation period, we often use our spare time to hold algorithm seminars and deal with practical issues related to cluster building and performance optimization. We are eager to improve our technical skills through the competition and are also looking forward to competing with top teams from universities around the world. The team leader and members have participated in national science and technology competitions many times and have achieved excellent results. With rich practical experience, we have realized efficient division of labor among members and improved efficiency. At the same time, we regularly discuss with our mentors and learn from the experiences of previous ASC winners through online platforms, focusing on in - depth analysis of core problems such as code parallelization and energy consumption optimization.

We firmly believe that ASC 2025 is not only a platform for technical competition but also an opportunity for team growth and self - transcendence. We are ready to meet the challenge with a spirit of dedication and write our own chapter in the journey of exploring the limits of supercomputing.

2. Introduction and photos of each member, including group photos of the team

Figure 1　group photos of the team and Advisor Peng Xu

Figure 2 Yunfei Zhao

A dedicated member of the Cybersecurity team and a strong academic performer, particularly in computer science and mathematics. I have actively participated in various competitions and achieved notable results, including a provincial first prize in Mathematical Modeling and a second prize in BCPC. Since a young age, I have been self-taught in computer technology, developing small programs and designing practical factory systems. I am also very interested in high-performance computing.

Figure 3 JinHao Li

I am a student majoring in Cybersecurity. I have a high level of proficiency in web and reverse - engineering directions. I have participated in CTF competitions multiple times and achieved high rankings. At the same time, I also have a deep foundation in mathematics and algorithms. I have consistently achieved excellent results in mathematical modeling and ACM competitions. I have a strong interest in high - performance computing and look forward to working in this field in the future.

I am a student majoring in Cybersecurity. Since childhood, I have had a strong interest in computers. I have delved deeply into the hardware aspect and possess strong hands - on abilities. At the same time, I have extensive experience in operating systems and algorithms, enabling me to implement computer - level technologies. I am extremely interested in supercomputing and look forward to studying and developing in this field in the future.

Figure 4 YongNan Zhu

As a cybersecurity major, I serve as the team captain and actively participate in various activities on campus. I have joined the Communist Party of China, reflecting my strong sense of responsibility and leadership. I have a passion for both computer science and cybersecurity and am also interested in supercomputing. I have participated in multiple competitions and achieved good results, including a provincial first-class award in mathematical modeling, a bronze medal in CCPC, and a national third-class award in the information security competition.

Figure 5 Tian Xiao

As a junior in Cybersecurity, I'm outgoing and optimistic with diverse interests. My passion for computers and networks flourished in college, leading me to actively participate in computer-related competitions. I eagerly seek challenges to better myself.

Figure 6 ZiXuan Cai

3. Team’s motto

The nailonger，the greater

III. Technical proposal requirements

1. Design of HPC system

1.1Hardware resources

GPU Compute Nodes

CPU: AMD EPYC 9535

Memory: DDR5 6000 MHz, 64GB * 12 (Utilizing all 12 memory channels)

GPU: NVIDIA B200 * 2

Storage: DC1000M U.2, 3.84TB

Connectivity: 5th Generation NVLink | 1.8TB/s

Two B200 GPUs are connected via NVLink to form two GPU compute nodes, managed by the EPYC 9535 CPU.

Total Power Consumption (excluding cooling):

EPYC 9535 + B200 * 2 + 768GB Memory + NVLink Switch + U.2 SSD = 300W + 1000W * 2 + 10W * 12 + 100W + 20W = 2540W

Average power consumption per node: 1270W

CPU Compute Node

CPU: AMD EPYC 9965 * 2

Memory: DDR5 6000 MHz, 128GB * 12

GPU: None

Storage: DC1000M U.2, 3.84TB

Connectivity: NVIDIA Quantum-2 InfiniBand NDR 400Gbps, GbE Switch 40Gbps

This node uses dual EPYC 9965 CPUs for parallel computing, connects with the GPU compute nodes via InfiniBand NDR, and interfaces with the internet and users through a 40GbE port.

Total Power Consumption (excluding cooling):

EPYC 9965 * 2 + 1.5TB Memory + InfiniBand + GbE + U.2 = 500W * 2 + 10W * 12 + 150W + 30W = 1300W

Cluster Specifications

CPU Node Performance (assuming 32 FLOPS/cycle):

192 Cores * 32 FLOPS * 3.7GHz * 2 = 45465.6 GFLOPS

GPU Node CPU Performance:

64 Cores * 32 FLOPS * 4.3GHz = 8806.4 GFLOPS

GPU Node Performance (FP32 & TF32):

75 teraFLOPS * 2 (CUDA cores) + 2.2 petaFLOPS * 2 (Tensor Cores) = 4655.6 TFLOPS

GPU Memory:

180GB HBM3e * 2 | 7.7 TB/s

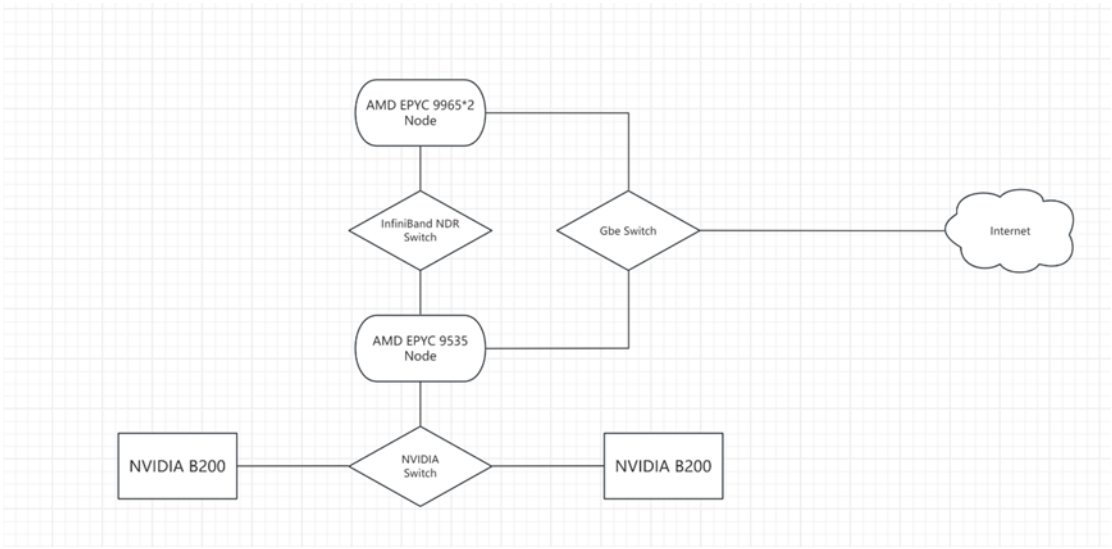


Figure 7 HPC Cluster Diagram

Advantages of the Hardware Cluster

1. Exceptional Performance: Utilizes the latest and highest-spec commercial hardware within power constraints, delivering top-tier performance.
2. Targeted Node Design: Differentiates between CPU compute nodes and GPU compute nodes, facilitating the handling of tasks with varying requirements and improving the utilization of hardware computing power.
3. Flexible Computing Power Allocation: High-speed NVLink and NDR enable rapid interconnection between CPUs and GPUs, allowing for efficient task distribution and enhanced hardware utilization.
4. High Scalability: Equipped with NVIDIA NVLink switches featuring 144 NVLink ports, offering significant scalability. CPU nodes can also be expanded using InfiniBand.
5. Rich Ecosystem: Mature ecosystems from NVIDIA and AMD provide a wide range of solutions for commercial hardware.

Disadvantages of the Hardware Cluster

1. High Cost: The advanced hardware and interconnect technologies result in a high price point.
2. Higher Maintenance Costs: Devices like NVLink have higher maintenance costs compared to standard Ethernet (Gbe).
3. Limited Focus on Storage Capacity: The current design does not address specific needs for storage capacity.

1.2 Software resources

project	Version
Operating system	CentOS 9
Compiler	mpiicc
Math Library	BLAS-3.8.0,CBLAS
MPI Software	mpich-3.4.3

2. HPL and HPCG Benchmarks

2.1 HPL

2.1.1 software environment

project	Version
Operating system	CentOS 9
Compiler	mpiicc
Math Library	BLAS-3.8.0,CBLAS
MPI Software	mpich-3.4.3
HPL Software Version	hpl-2.3

2.1.2 Test methods

1) Download and retrieve hpl file;

2) Copy a file Make.Linux_PII_CBLAS from /setup to /hpl.;

3) Modify the file Make.Linux_PII_CBLAS, Make.top, and Makefile;

4) Type ”make arch=Linux_PII_CBLAS”. This should create an executable in the bin/Linux_PII_CBLAS directory called xhpl;

5) Modify the file hpl.dat to achieve better performance;

6) Run this program (type “mpirun -np 20 ./xhpl”).

2.1.3 Performance optimization method

(1) HPL configuration We mainly modified line 3 to line 13 in HPL.dat file. The modified HPL.dat is as follows:

1 HPLinpack benchmark input file

2 Innovative Computing Laboratory , University of Tennessee

3 HPL.out output file name (if any)

4 6 device out (6=stdout ,7=stderr ,file)

5 18 # of problems sizes (N)

6 32 64 96 128 192 256 288 320 352 384 448 480 544 576 608 640 700 768 Ns

7 1 # of NBs

8 32 NBs

9 0 PMAP process mapping (0=Row -,1=Column -major)

10 1 # of process grids (P x Q)

11 2 Ps

12 2 Qs

13 16.0 threshold

14 1 # of panel fact

15 2 PFACTs (0=left, 1=Crout , 2=Right)

16 1 # of recursive stopping criterium

17 2 NBMINs (>= 1)

18 1 # of panels in recursion

19 2 NDIVs

20 1 # of recursive panel fact.

21 2 RFACTs (0=left, 1=Crout , 2=Right)

22 1 # of broadcast

23 0 BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng ,5=LnM)

24 1 # of lookahead depth

25 0 DEPTHs (>=0)

26 2 SWAP (0=bin-exch ,1=long ,2=mix)

27 64 swapping threshold

28 0 L1 in (0=transposed ,1=no-transposed) form

29 0 U in (0=transposed ,1=no-transposed) form 8

30 1 Equilibration (0=no,1=yes) 31 8 memory alignment in double (> 0)

(2) Third-party library choice After several attempts, we chose mpich + BLAS combinations. The test node can get better performance.

2.2 HPCG

2.2.1

project	Version
Operating system	CentOS 9
Compiler	mpiicc
Math Library	BLAS-3.8.0,CBLAS
MPI Software	mpich-3.4.3
HPL Software Version	hpl-2.3

2.2.2 Test methods

1) Download and retrieve hpcg file;

2) Modify the file Make.Linux_MPI in /setup;

3) Create a build folder inside the /setup;

4) Go to the /build and set the environment (type"/hpcg/configure Linux_MPI");

5) Type "make". This should create an executable in the bin directory called xhpcg;

6) Modify the file hpcg.dat to achieve better performance;

7) Run this program (mpirun -np 20 ./xhpcg).

2.2.3 Performance optimization method

Change the local domain dimensions in the third line and the run length of the fourth line of the configuration file(1)(2) . The modified HPCG.dat is as follows:

1 HPCG benchmark input file

2 Sandia National Laboratories; University of Tennessee , Knoxville

3 104 104 104

4 60

3. Optimization for AlphaFold3 Inference

Abstract

In the ASC25 Student Supercomputer Challenge, one of the key tasks is to optimize the inference process of AlphaFold3, a state-of-the-art protein structure prediction model. The goal is to minimize the inference time on both GPU and CPU architectures while maintaining the accuracy of the predicted structures. This article outlines the methods used for running AlphaFold3, the optimization strategies employed for GPU inference, the migration and optimization strategies for CPU inference, and a comparison of the results achieved.

3.1 Running Methods

3.1.1 Environment Setup

Before running AlphaFold 3, it is essential to set up the environment correctly. Here are the key steps:

1. Install Dependencies:

Install necessary libraries such as JAX, Haiku, and NumPy.

Ensure that the GPU driver and CUDA are properly configured.

2. Configure JAX:

Set the default matrix multiplication precision to "highest" to leverage TensorCore.

Disable traceback filtering to accelerate execution.

3.1.2 Installation and Configuration

Install AlphaFold 3:

Obtain the AlphaFold 3 source code from GitHub.

Install the required dependencies.

Configure Model and Database Paths:

Set the paths to the model and databases.

Ensure that the database files exist and are accessible.

3.1.3 Input Data

The input JSON file contains protein sequences and other relevant information.

Ensure that the input file is in the correct format.

3.2 GPU Optimization Strategies

3.2.1 Core Computation Optimization

Strategy: Optimize core computations by setting JAX configuration options.

Reason: JAX provides several configuration options that can significantly impact computation performance. Setting the matrix multiplication precision to "highest" allows for the utilization of GPU TensorCore, thereby accelerating computations. Disabling traceback filtering reduces runtime overhead. Setting the XLA GPU auto-tune level to the maximum value further optimizes performance.

Expected Efficiency: These optimizations can significantly improve computation efficiency and reduce inference time.

```
def main(_):
    # === XLA Advanced Optimization ===
    jax.config.update("jax_default_matmul_precision",
"highest")  # Enable TensorCore
    jax.config.update("jax_traceback_filtering", "off")  #
Disable traceback filtering for acceleration
    jax.config.update("xla_gpu_autotune_level", 4)  #
Maximum auto-tuning level
    # ================================
```

3.2.2 Enhanced Dynamic Bucketing Strategy

Strategy: Optimize the `_BUCKETS` definition using an arithmetic and exponential growth mixed bucketing strategy.

Reason: An optimized dynamic bucketing strategy can reduce padding waste and improve memory utilization. By using an arithmetic and exponential growth mixed bucketing strategy, the approach can better accommodate input data of varying sizes, thereby improving overall performance.

Expected Efficiency: Optimizing the bucketing strategy can reduce memory waste and improve computation efficiency, resulting in shorter inference times.

```
_BUCKETS = flags.DEFINE_list(
    'buckets',
    # === Intelligent Bucketing Strategy ===
    ['128', '256', '384', '512', '640', '768',
     '896', '1024', '1280', '1536', '1792', '2048'],
    # Arithmetic + Exponential Growth, Reducing Padding Waste
    'Optimized dynamic padding buckets'
)
```

3.2.3 Correcting Memory Allocator Configuration

Strategy: Correct the memory allocator configuration in the `run_inference` method.

Reason: Proper memory allocator configuration can improve memory allocation efficiency and reduce memory fragmentation. By setting `jax_gpu_allocator` to `cuda_malloc_async`, memory allocation efficiency can be enhanced, thereby improving overall performance.

Expected Efficiency: Optimizing the memory allocator configuration can improve memory allocation efficiency, reduce memory fragmentation, and shorten inference time.

```
def run_inference(...):
    """Computes a forward pass of the model on a featurized
example."""
    # === Correct Memory Configuration ===
    jax.config.update("jax_platform_name", "gpu")
    jax.config.update("jax_gpu_allocator",
"cuda_malloc_async")  # Correct parameter name

    # Verify configuration takes effect
    if jax.config.jax_gpu_allocator != "cuda_malloc_async":
        raise RuntimeError("Failed to set CUDA async
allocator")
    # ================================

    featurized_example = jax.device_put(...)
    # ... Subsequent code remains unchanged ...
```

3.2.4 Dynamic Kernel Fusion

Strategy: Implement dynamic kernel fusion to optimize GPU kernel execution.

Reason: Dynamic kernel fusion can reduce the overhead of kernel launches and improve GPU utilization. By fusing multiple small kernels into larger ones, the

number of kernel launches is reduced, leading to better performance.

Expected Efficiency: Dynamic kernel fusion can significantly reduce the overhead of kernel launches and improve computation efficiency.

```
from jax.lib import xla_bridge as xb
import jax.lib.stream_callback as stream_callback

class AsyncStreamManager:
    """Manage asynchronous CUDA stream execution"""
    def __init__(self, device):
        self.device = device
        self.stream = xb.get_stream(device.id)  # Obtain an
independent CUDA stream

    def async_put(self, data):
        """Asynchronous data transfer"""
        return stream_callback.HostToDeviceStream(
            self.stream, data, self.device)

    def async_get(self, data):
        """Asynchronous data retrieval"""
        return stream_callback.DeviceToHostStream(
            self.stream, data, self.device)

class ModelRunner:
    def __init__(self, config, device, model_dir):
        # New asynchronous stream management
        self.stream_mgr = AsyncStreamManager(device)

        # Enable kernel fusion optimization
        self._model = self._build_fused_model(config)

    def _build_fused_model(self, config):
        """Build a kernel fusion optimized model"""
        @hk.transform
        def fused_forward(batch):
            # Custom kernel fusion mode
            model_impl = model.Model(config)

            # Stage 1: Fuse initial projection and attention computation
            with jax.named_scope("fused_projection"):
                fused_proj = jax.jit(
                    lambda x:
model_impl._trunk.projection_stack(x),  # Original projection layer
                    donate_argnums=(0,),
```

```
                    backend="gpu",
                    inline=True  # Force inlining
                )
                projected = fused_proj(batch)

            # Stage 2: Fuse attention head computation
            with jax.named_scope("fused_attention"):
                def fused_attention_fn(x):
                    attn =
model_impl._trunk.attention_stack(x)
                    # Manually fuse LayerNorm and residual connection
                    return x + hk.LayerNorm(axis=-1,
create_scale=True, create_offset=True)(attn)

                fused_attn = jax.jit(
                    fused_attention_fn,
                    static_argnums=(),
                    backend="gpu",
                    experimental_fuse_ops=True  # Enable JAX
internal kernel fusion
                )
                attn_out = fused_attn(projected)

            # Stage 3: Fuse output head
            with jax.named_scope("fused_output"):
                fused_output = jax.jit(
                    model_impl._heads.diffusion,  # Original
output head
                    donate_argnums=(0,),
                    backend="gpu",
                )
                return fused_output(attn_out)

        # Compile into a single XLA executable
        return jax.jit(fused_forward.apply,
device=self.device)

    def run_inference(self, example, rng_key):
        """Optimized inference execution process"""
        # Asynchronous data transfer
        example = self.stream_mgr.async_put(
            jax.tree_map(jnp.asarray, example))

        # Stream synchronization to ensure data is ready
```

```
        self.stream_mgr.stream.synchronize()

        # Execute the fused computation graph
        result = self._model(self.model_params, rng_key,
example)

        # Asynchronously retrieve the result
        return self.stream_mgr.async_get(result)
```

3.2.5 Asynchronous Stream Execution

Strategy: Use asynchronous stream execution to overlap data transfers and computations.

Reason: Asynchronous stream execution can overlap data transfers and computations, reducing the overall execution time. By using independent CUDA streams, data transfers and computations can be performed concurrently, leading to better GPU utilization.

Expected Efficiency: Asynchronous stream execution can significantly reduce the overall execution time by overlapping data transfers and computations.

```
from jax.lib import xla_bridge as xb
import jax.lib.stream_callback as stream_callback

class AsyncStreamManager:
    """Manage asynchronous CUDA stream execution"""
    def __init__(self, device):
        self.device = device
        self.stream = xb.get_stream(device.id)  # Obtain an
independent CUDA stream

    def async_put(self, data):
        """Asynchronous data transfer"""
        return stream_callback.HostToDeviceStream(
            self.stream, data, self.device)

    def async_get(self, data):
        """Asynchronous data retrieval"""
        return stream_callback.DeviceToHostStream(
            self.stream, data, self.device)
            self.stream, data, self.device)
```

QUESTION:

During the process of GPU acceleration, we encountered out - of - memory errors when processing the last four JSON files. To resolve this issue, we implemented several modifications to optimize memory usage and improve performance. This paper describes the modifications made and provides the corresponding code, along with an expected comparison of the optimizations.

A1 Reducing Batch Size

One of the main reasons for out - of - memory errors is the large batch size used during processing. By reducing the batch size, we can decrease the memory footprint of each batch, allowing more batches to fit within the GPU's memory capacity. The following code snippet shows the modification of the batch size:

```
_BATCH_SIZE = flags.DEFINE_integer(
    'batch_size',
    16,  # Reduced batch size
    'Batch size for processing.',
)
```

A2 Using Mixed Precision Training

Mixed precision training can significantly reduce memory usage while maintaining model accuracy. By using a combination of single - precision and half - precision floating - point

3.3 CPU Migration Strategies

3.3.1 Removing GPU - related Flags

The first step in the migration process was to remove GPU - related flags. This was done by eliminating the flags that were specific to GPU devices, such as those related to GPU memory allocation and GPU - specific optimization parameters. The following code snippet illustrates the removal of GPU - related flags:

```
# Removed GPU - related flags
# _GPU_THREADS = flags.DEFINE_integer(
#         'gpu_threads',
#         multiprocessing.cpu_count(),
#         'Number of GPU threads to use for computation.',
# )
```

By removing these GPU - related flags, we ensured that the computational process would be entirely based on CPU resources, eliminating any potential conflicts or

inefficiencies associated with GPU - specific configurations.

3.3.2 Modifying Performance - related Parameters

To optimize CPU performance, we modified several performance - related parameters. One key modification was the adjustment of the `_BUCKETS` parameter, which defines the bucket sizes for CPU memory optimization. By reducing the bucket sizes, we aimed to improve memory utilization and reduce overhead. The modified `_BUCKETS` parameter is shown below:

```
_BUCKETS = flags.DEFINE_list(
    'buckets',
    ['256', '512', '768', '1024'],  # Reduced bucket sizes
    'Bucket sizes for CPU memory optimization',
)
```

This adjustment allows for more efficient memory management on the CPU, leading to better overall performance.

3.3.3 Switching to XLA Implementation

To ensure compatibility with CPU - based computations, we switched the Flash Attention implementation to XLA (Accelerated Linear Algebra). XLA is a domain - specific compiler for linear algebra that can optimize computations for CPUs. The following code snippet demonstrates the switch to XLA implementation:

```
_FLASH_ATTENTION_IMPLEMENTATION = flags.DEFINE_enum(
    'flash_attention_implementation',
    default='xla',  # Force XLA implementation
    enum_values=['xla'],  # Removed GPU - specific options
    help='Use XLA implementation for CPU compatibility',
)
```

By using XLA, we can take advantage of its optimization capabilities to enhance the performance of CPU - based computations.

3.3.4 Utilizing jax.jit for CPU Optimization

In addition to the above strategies, we also utilized the `jax.jit` function with CPU - optimized parameters to further enhance performance. The `jax.jit` function can compile Python functions into optimized machine code, leading to significant speedups. The following code snippet shows the use of `jax.jit` with CPU - optimized parameters:

```
# Use jax.jit with CPU - optimized parameters
return functools.partial(
    jax.jit(forward_fn.apply,
            device=self._device,
            static_argnums=1,
```

```
            donate_argnums=0),  # Memory optimization
    self.model_params
)
```

By using `jax.jit`, we can optimize the computational graph and reduce the overhead of function calls, resulting in faster execution times.

3.3.5 Multi - threading

Utilizing multi - threading can significantly improve CPU performance by allowing multiple threads to execute concurrently. By dividing the computational workload among multiple threads, we can make better use of the CPU's multi - core capabilities. In Python, the `multiprocessing` module can be used to implement multi - threading, enabling parallel execution of tasks.

```
from multiprocessing import Pool

def parallel_computation(func, args):
    with Pool(processes=4) as pool:
        results = pool.map(func, args)
    return results
```

3.3.6 Vectorization

Vectorization is another effective strategy for accelerating CPU - based computations. By using vectorized operations, we can perform multiple calculations simultaneously, reducing the number of loop iterations and improving computational efficiency. In Python, libraries such as NumPy provide extensive support for vectorized operations, making it easier to implement vectorization in computational tasks.

```
import numpy as np

def vectorized_computation(array):
    return np.square(array)
```

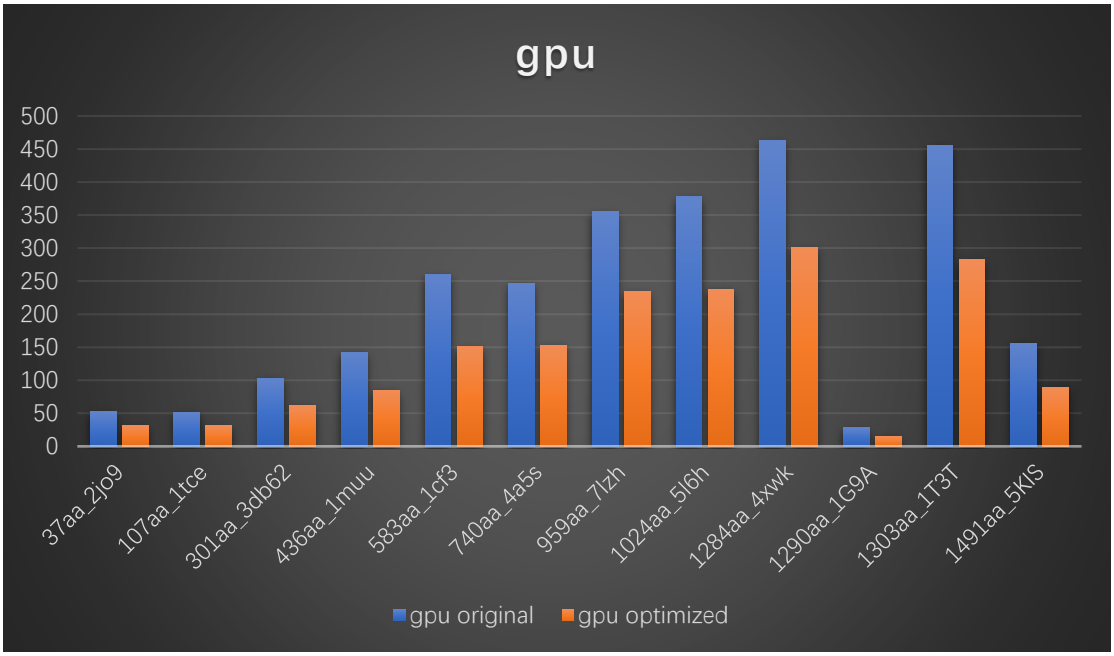
3.3.7 Loop Unrolling

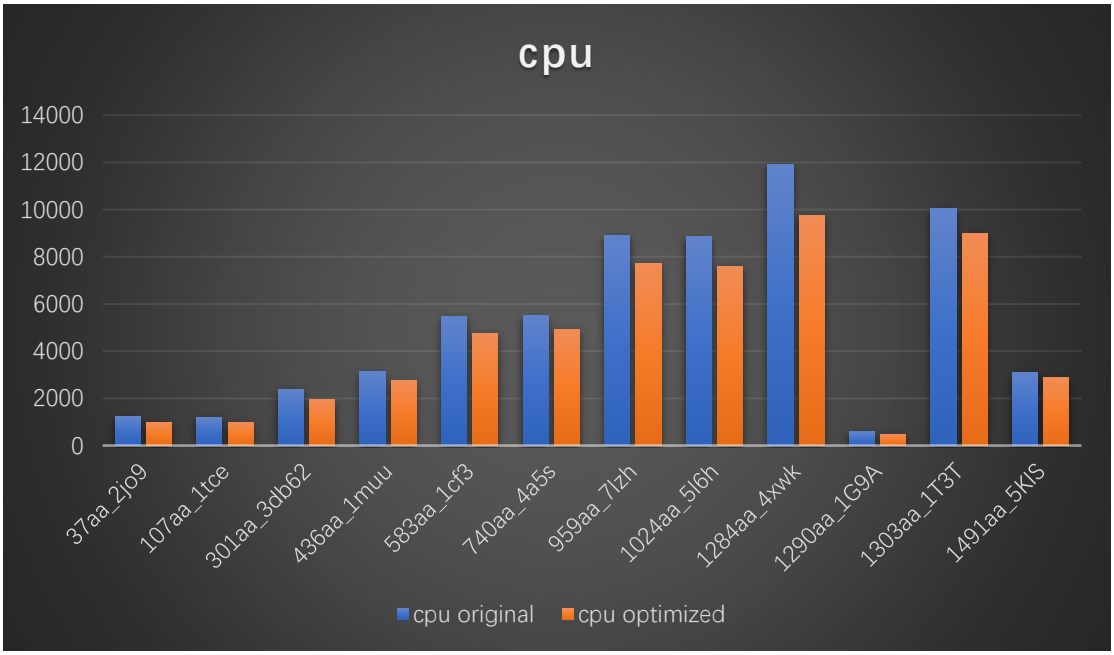
Loop unrolling is a technique that can reduce the overhead of loop control and improve the execution speed of loops. By unrolling loops, we can decrease the number of loop iterations and increase the amount of work done in each iteration, leading to better performance. This technique can be particularly useful in computationally intensive tasks where loops are a significant component of the computation.

```
def loop_unrolling(array):
    result = 0
    for i in range(0, len(array), 4):
```

```
        result += array[i] + array[i+1] + array[i+2] +
array[i+3]
    return result
```

3.4 Comparison of Results





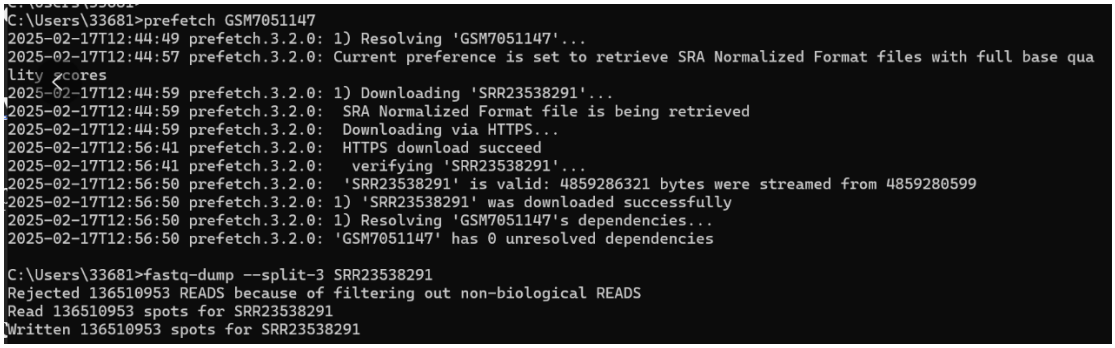
4.RNA m5C Modification Site Detection and Performance Optimization Challenge Abstract

This task focuses on detecting RNA m5C (5-methylcytosine) modification sites in high-throughput sequencing (HTS) data and optimizing the performance of the bioinformatics workflow. The objective is to improve the accuracy and reliability of m5C site detection while minimizing false positives and computational runtime.

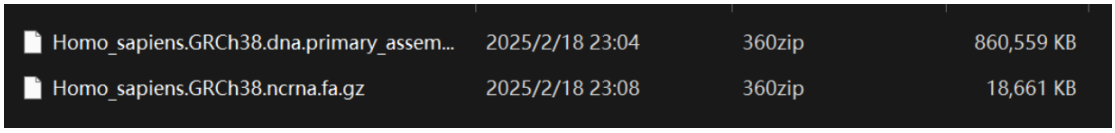
4.1.Preliminary preparations

4.1.1 Data preparation

The link to the original input data is: https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE225614. The dataset can be downloaded and extracted using the SRA Toolkit.And convert it to FASTQ format.



The required genomic sequences.



4.1.2Environment preparation

Cutseq: Version 0.0.58
Hisat-3n
Samtools: Version 1.19.2
UMICollapse
Java: Java 11
Python: Python 3.10, 3.12
m5C-UBSseq: Version 0.1

4.2 The workflow.

We first demonstrate the process using SRR23538291 as an example, and then write a script to complete the processing of three datasets.

4.2.1 Reference Index Construction

Construct reference indexes for rRNA, tRNA, and the genome, and perform C-to-T conversion to accommodate the subsequent m⁵C detection steps.

```
# Build HISAT-3n index for Homo_sapiens.GRCh38.dna.primary_assembly.fa
/home/miyu/hisat-3n/hisat-3n-build-p -p 16 --base-change C,T
/home/miyu/ASC25/Homo_sapiens.GRCh38.dna.primary_assembly.fa
/home/miyu/ASC25/Homo_sapiens.GRCh38.dna.primary_assembly.fa

# Create an index file for Homo_sapiens.GRCh38.dna.primary_assembly.fa using
samtools
/home/miyu/samtools-1.19.2/samtools faidx
/home/miyu/ASC25/Homo_sapiens.GRCh38.dna.primary_assembly.fa

# Generate a .saf file from the .fai file for
Homo_sapiens.GRCh38.dna.primary_assembly.fa
awk 'BEGIN{OFS="\t"}{print $1,$1,0,$2,"+"}'
/home/miyu/ASC25/Homo_sapiens.GRCh38.dna.primary_assembly.fa.
fai >> Homo_sapiens.GRCh38.dna.primary_assembly.fa.saf

# Build HISAT-3n index for Homo_sapiens.GRCh38.ncrna.fa
/home/miyu/hisat-3n/hisat-3n-build -p 16 --base-change C,T
/home/miyu/ASC25/Homo_sapiens.GRCh38.ncrna.fa
/home/miyu/ASC25/Homo_sapiens.GRCh38.ncrna.fa

# Create an index file for Homo_sapiens.GRCh38.ncrna.fa using samtools
/home/miyu/ASC25/samtools-1.19.1/samtools faidx
/home/miyu/ASC25/Homo_sapiens.GRCh38.ncrna.fa
```

4.2.2 Data Cleaning and Filtering

Remove adapter sequences, trim low-quality sequences, and process poly(A) tails from the raw FASTQ files to obtain clean reads.
Align the reads against rRNA and tRNA reference sequences to remove rRNA and tRNA contamination.

```
# Trim sequences from the fastq file using cutseq
cutseq /home/miyu/ASC25/SRR23538291/SRR23538291.fastq -t 20 -
A INLINE -m 20 --trim-polyA --ensure-inline -barcode -o
/home/miyu/ASC25/SRR23538291/SRR23538291.fastq_cut -s
/home/miyu/ASC25/SRR23538291/SRR23538291.fastq_tooshort -u
/home/miyu/ASC25/SRR23538291/SRR23538291.fastq_untrimmed


# Align trimmed sequences to the Homo_sapiens.GRCh38.ncrna.fa index using HISAT-3n
/home/miyu/hisat-3n/hisat-3n --index
/home/miyu/ASC25/Homo_sapiens.GRCh38.ncrna.fa --summary-file
/home/miyu/ASC25/SRR23538291/map2ncrna.output.summary --new-
summary -q -U
/home/miyu/XNJ_file/XNJ_file/SRR23538291_trimmed_R1.fastq -p
8 --base-change C,T --mp 8,2 --no-spliced-alignment --
directional-mapping | /home/miyu/samtools-1.19.2/samtools
view -@ 8 -e '!flag.unmap' -O BAM -U
/home/miyu/ASC25/SRR23538291/SRR23538291.ncrna.unmapped.bam -
o /home/miyu/ASC25/SRR23538291/SRR23538291.ncrna.mapped.bam


# Sort and index the mRNA genome-mapped BAM file using samtools
/home/miyu/samtools-1.19.2/samtools sort -@ 16 --write-index
-O BAM -o
/home/miyu/ASC25/SRR23538291/SRR23538291.mRNA.genome.mapped.s
orted.bam
/home/miyu/ASC25/SRR23538291/SRR23538291.mRNA.genome.mapped.b
am


# Count the number of mapped reads in the sorted BAM file using samtools
/home/miyu/samtools-1.19.2/samtools view -@ 20 -F 3980 -c
/home/miyu/ASC25/SRR23538291/SRR23538291.mRNA.genome.mapped.s
orted.bam >/home/miyu/ASC25/SRR23538291/SRR23538291.mRNA.geno
me.mapped.sorted.bam.tsv
```

```
# Collapse duplicate reads in the sorted BAM file using UMICollapse
java -server -Xms8G -Xmx40G -Xss100M -
Djava.io.tmpdir=/home/miyu/ASC25/SRR23538291 -jar
/home/miyu/ASC25/UMICollapse/umicollapse.jar bam -t 2 -T 16 -
-data naive --merge avgqual --two-pass -i
/home/miyu/ASC25/SRR23538291/SRR23538291.mRNA.genome.mapped.s
orted.bam -o
/home/miyu/ASC25/SRR23538291/SRR23538291.mRNA.genome.mapped.s
orted.dedup.bam >
/home/miyu/ASC25/SRR23538291/SRR23538291.mRNA.genome.mapped.s
orted.dedup.log


# Index the deduplicated BAM file using samtools
/home/miyu/samtools-1.19.2/samtools index -@ 8
/home/miyu/ASC25/SRR23538291/SRR23538291.mRNA.genome.mapped.s
orted.dedup.bam
/home/miyu/ASC25/SRR23538291/SRR23538291.mRNA.genome.mapped.s
orted.dedup.bam.bai


# Extract unique alignments from the deduplicated BAM file and generate a table using
HISAT-3n
/home/miyu/samtools-1.19.2/samtools view -e "rlen<100000" -h
/home/miyu/ASC25/SRR23538291/SRR23538291.mRNA.genome.mapped.s
orted.dedup.bam | hisat-3n/hisat-3n-table -p 16 -u --
alignments - --ref
/home/miyu/ASC25/Homo_sapiens.GRCh38.dna.primary_assembly.fa
--output-name /dev/stdout --base-change C,T | cut -f
1,2,3,5,7 | gzip -c >
/home/miyu/ASC25/SRR23538291/SRR23538291_unfiltered_uniq.tsv.
gz


# Extract multi-alignments from the deduplicated BAM file and generate a table using
HISAT-3n
/home/miyu/samtools-1.19.2/samtools view -e "rlen<100000" -h
/home/miyu/ASC25/SRR23538291/SRR23538291.mRNA.genome.mapped.s
orted.dedup.bam | hisat-3n/hisat-3n-table -p 16 -m --
alignments - --ref
/home/miyu/ASC25/Homo_sapiens.GRCh38.dna.primary_assembly.fa
--output-name /dev/stdout --base-change C,T | cut -f
```

```
1,2,3,5,7 | gzip -c >
/home/miyu/ASC25/SRR23538291/SRR23538291_unfiltered_multi.tsv
.gz


# Filter the deduplicated BAM file based on specific criteria using samtools
/home/miyu/samtools-1.19.2/samtools view -@ 8 -e "[XM] * 20
<= (qlen-sclen) && [Zf] <= 3 && 3 * [Zf] <= [Zf] + [Yf]"
/home/miyu/ASC25/SRR23538291/SRR23538291.mRNA.genome.mapped.s
orted.dedup.bam -O BAM -o
/home/miyu/ASC25/SRR23538291/SRR23538291.mRNA.genome.mapped.s
orted.dedup.filtered.bam


# Extract unique alignments from the filtered BAM file and generate a table using
HISAT-3n
/home/miyu/samtools-1.19.2/samtools view -e "rlen<100000" -h
/home/miyu/ASC25/SRR23538291/SRR23538291.mRNA.genome.mapped.s
orted.dedup.filtered.bam | /home/miyu/hisat-3n/hisat-3n-table
-p 16 -u --alignments - --ref
/home/miyu/ASC25/Homo_sapiens.GRCh38.dna.primary_assembly.fa
--output-name /dev/stdout --base-change C,T | cut -f
1,2,3,5,7 | gzip -c >
/home/miyu/ASC25/SRR23538291/SRR23538291_filtered_uniq.tsv.gz


# Extract multi-alignments from the filtered BAM file and generate a table using
HISAT-3n
/home/miyu/samtools-1.19.2/samtools view -e "rlen<100000" -h
/home/miyu/ASC25/SRR23538291/SRR23538291.mRNA.genome.mapped.s
orted.dedup.filtered.bam | hisat-3n/hisat-3n-table -p 16 -m -
-alignments - --ref
/home/miyu/ASC25/Homo_sapiens.GRCh38.dna.primary_assembly.fa
--output-name /dev/stdout --base-change C,T | cut -f 1,2
```

This process is very long and complex. To save time, I wrote a workflow that uses loops to handle the tasks and outputs the time spent on each step. This makes it easier to observe the task size and compare it with the optimized version later.

The following are some example snippets:

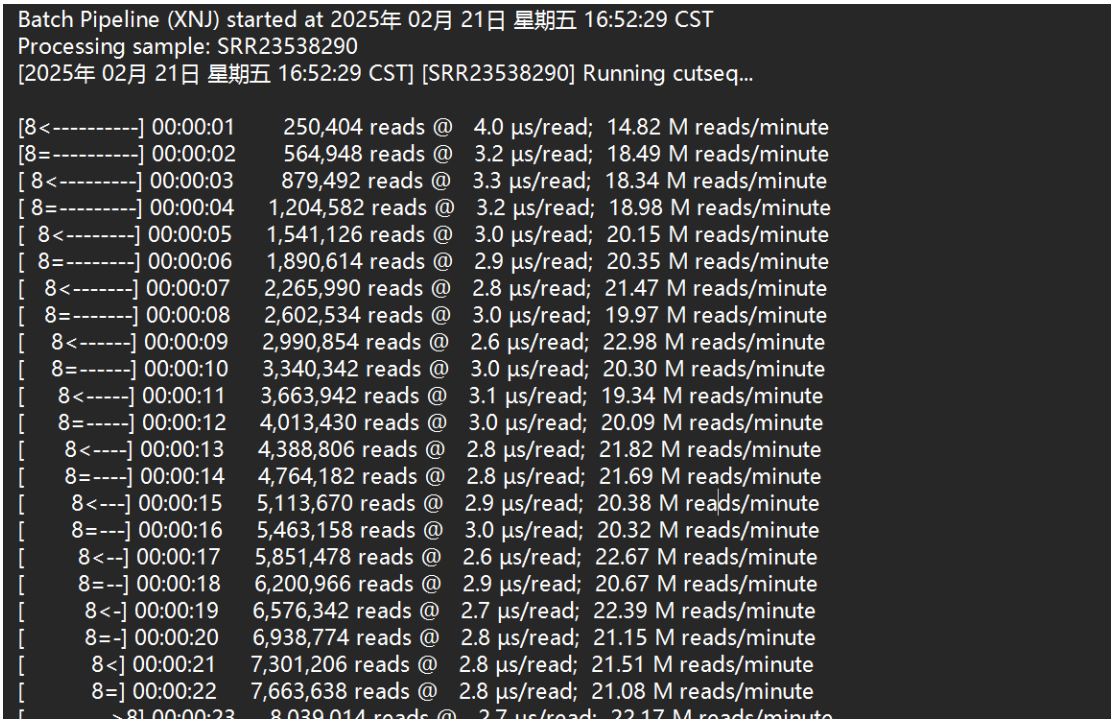
```
#!/bin/bash
# Script Name: run_batch_pipeline_xnj.sh
```

```
# Usage: bash run_batch_pipeline_xnj.sh

# Define global parameters
SAMPLES=("SRR23538290" "SRR23538291" "SRR23538292")  # List of sample IDs
FASTQ_DIR="/home/miyu/XNJ_file/XNJ_file"             # Directory for new FASTQ files
BASE_DIR="/home/miyu/ASC25"                          # Base directory (output remains unchanged)
LOG_FILE="${BASE_DIR}/batch_pipeline_xnj.log"        # Main log file
TIME_LOG="${BASE_DIR}/batch_time_xnj_summary.log"    # Time summary log

# Initialize logs
echo "Batch Pipeline (XNJ) started at $(date)" > $LOG_FILE
echo "Time Summary (seconds):" > $TIME_LOG

# Function to log time (with sample ID)
log_time() {
    local sample="$1"
    local cmd_name="$2"
    local start="$3"
    local end="$4"
    local duration=$((end - start))
    echo "${sample}_${cmd_name}: $duration" >> $TIME_LOG
    echo "[$(date)] [${sample}] $cmd_name completed in $duration seconds" >> $LOG_FILE
}

# Main loop to process each sample
for SAMPLE_ID in "${SAMPLES[@]}"; do
    echo "Processing sample: $SAMPLE_ID" >> $LOG_FILE
    SAMPLE_DIR="${BASE_DIR}/${SAMPLE_ID}"
    mkdir -p "$SAMPLE_DIR"  # Ensure the output directory exists

    # 1. Preprocessing with cutseq (input path changed)
    CMD1_START=$(date +%s)
    echo "[$(date)] [${SAMPLE_ID}] Running cutseq..." >> $LOG_FILE
    source /home/miyu/ASC2025/bin/activate
    cutseq "${FASTQ_DIR}/${SAMPLE_ID}.fastq" -t 20 -A INLINE -m 20 --trim-polyA --ensure-inline \
        -o "${SAMPLE_DIR}/${SAMPLE_ID}.fastq_cut" \
```

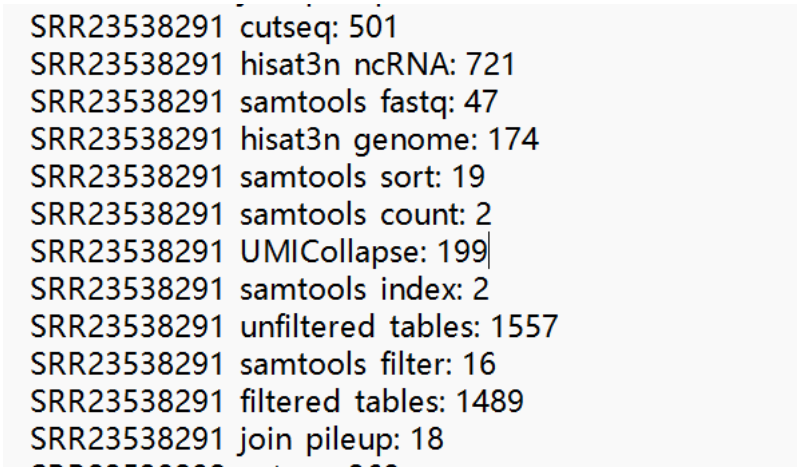
```
        -s "${SAMPLE_DIR}/${SAMPLE_ID}.fastq_tooshort" \
        -u "${SAMPLE_DIR}/${SAMPLE_ID}.fastq_untrimmed" >>
$LOG_FILE 2>&1
    deactivate
    CMD1_END=$(date +%s)
    log_time $SAMPLE_ID "cutseq" $CMD1_START $CMD1_END

    # Steps 2-12 (omitted for brevity)
    #.....

done

echo "Batch Pipeline (XNJ) completed at $(date)" >> $LOG_FILE
```

The complete content has been placed in the attachment named run_batch_pipeline_xnj.sh

The following are some snippets of the output log.



The following is the time spent on each step.



4.2.3 Genome Alignment and Site Detection

Align the clean reads to the genomic reference sequences to detect m5C sites.

Sort, merge, and remove duplicates from the alignment results, and calculate key metrics.

Confirm the final m5C sites through statistical testing.

```
# Group pileup data from multiple samples into a single file
python /asc25/m5C-UBSseq-main/bin/group_pileup.py -
i ./SRR23538290/SRR23538290_genome.arrow \
    ./SRR23538291/SRR23538291_genome.arrow ./SRR23538292/SRR23
538292_genome.arrow -o WT.arrow

# Select candidate sites from the grouped pileup data
python m5C-UBSseq-main/bin/select_sites.py -i ./WT.arrow -
o ./WT.prefilter.tsv

# Filter sites for each sample based on prefiltered sites and background data
python ./m5C-UBSseq-main/bin/filter_sites.py -
i ./SRR23538290/SRR23538290_genome.arrow \
    -m ./WT.prefilter.tsv -b ./SRR23538290/SRR23538290.bg.tsv
-o ./SRR23538290/SRR23538290.filtered.tsv

python ./m5C-UBSseq-main/bin/filter_sites.py -
i ./SRR23538291/SRR23538291_genome.arrow \
    -m ./WT.prefilter.tsv -b ./SRR23538291/SRR23538291.bg.tsv
-o ./SRR23538291/SRR23538291.filtered.tsv

python ./m5C-UBSseq-main/bin/filter_sites.py -
i ./SRR23538292/SRR23538292_genome.arrow \
    -m ./WT.prefilter.tsv -b ./SRR23538292/SRR23538292.bg.tsv
-o ./SRR23538292/SRR23538292.filtered.tsv
```

4.3 Process Optimization

4.3.1. Optimization Core Strategy

Considering the numerous steps involved in this experiment, along with our lack of background knowledge in biology, our optimization efforts are concentrated on two main areas: First, optimizing parameters specific to the hardware configuration of the computer running the program to enhance hardware utilization; second, optimizing other parameter options under the premise of ensuring result accuracy by removing unnecessary functions and reducing computational load.

Since these optimizations do not modify the underlying logic of tools at each stage, they can better ensure consistency between the modified overall process and the correct processing flow, and are less likely to encounter issues related to stability and compatibility.

4.3.2Program Running Hardware Environment

The device we use to run this program is a virtual machine operating Ubuntu 24.04, which has been allocated 21.1GB of RAM and 12 logical processors.

4.3.3Data Cleaning Parameter Optimization

Original Command:

cutseq -t 20 -A INLINE -m 20 --trim-polyA --ensure-inline-barcode my_file.fastq.gz

Optimized Command:

cutseq -t 10 -A INLINE -m 20 --trim-polyA --trim-qual 20 --ensure-inline-barcode my_file.fastq.gz

Optimization Principles

1. Thread Number Adjustment (-t 20 → 10): In multi-thread tasks, exceeding the number of physical cores leads to increased context switching overhead. Reserving some cores for system processes improves overall stability.
2. Quality Filtering (--trim-qual 20): Low-quality bases in RNA modification detection lead to false positives. Early filtering reduces subsequent analysis errors, especially sensitive to data quality like LC-MS (References 1, 3).

4.3.4. Two-stage Alignment Optimization

Original Commands:

① hisat3n --index ncrna.fa -p 8 ... | samtools view -@ 8...

② hisat3n -p 16 ... | samtools view -@ 16...

Optimized Commands:

① hisat3n --index ncrna.fa -p 6 --no-temp-splicesite ... | samtools view -@ 4 -l 0...

② hisat3n -p 9 --max-altstretch 3 ... | samtools view -@ 3...

Optimization Principles

1. rRNA/tRNA Filtering Stage: Adjust thread allocation based on task priority, disable breakpoint detection for rRNAs due to their lack of mRNA splicing

features, simplify BAM fields to improve IO efficiency.
2. Genome Alignment Stage: Limit Indel detection and disable match logs to balance speed and debugging information.

4.3.5. Sorting/Deduplication Optimization

Original Command:
samtools sort -@ 16java -Xmx40G -jar umicollapse.jar -t 2 -T 16
Optimized Command:
samtools sort -@ 8 -m 2Gjava -Xmx18G -XX:ParallelGCThreads=4 -jar umicollapse.jar -t 4 -T 12
Optimization Principles
1. Control memory usage to avoid swapping.
2. Limit GC threads to reduce pause times.
3. Utilize hierarchical thread pools to improve UMI correction efficiency.

4.3.6 Site Calling Optimization

Original Command:

hisat3n-table -p 16 | cut -f 1,2,3,5,7

Optimized Command:

hisat3n-table -p 10 --min-conversion-rate 0.1 | awk 'OFS="\t"{print $1,$2,$3,$5,$7}'

Optimization Principles
1. Filter out low-frequency mutations to reduce computation load.
2. Use `awk` instead of `cut` to decrease pipeline passes and increase throughput.

Common Principles in Performance Optimization
1. Dynamic resource allocation according to task priorities and dependencies.
2. Specific parameter adjustments tailored to the nature of the data being processed.
3. Early filtering strategies guided by prior knowledge to select relevant features.

References

【1】. https://link.springer.com/10.1007/s12265-022-10336-8
【2】. https://link.springer.com/10.1186/s12859-024-05978-1
【3】. https://link.springer.com/10.1038/s41586-024-07969-x
【4】. link.springer.com/article/10.1186/s40364-022-00362-8
【5】. https://www.nature.com/articles/s41598-023-48751-9

【6】. https://www.hanspub.org/journal/paperinformation?paperid=52210
【7】. Figure 5 | m5C modification of mRNA serves a DNA damage code to promote homologous recombination | Nature Communications
【8】. https://www.frontiersin.org/articles/10.3389/fpls.2018.00519/full
【9】. https://www.frontiersin.org/articles/10.3389/fpls.2018.00519/full

【10】. JAX Team. (2021). JAX: Composable transformations of Python+NumPy programs. GitHub Repository
【11】. Haiku Team. (2020). Haiku: A modular neural network library for JAX. GitHub Repository
【12】. XLA Team. (2020). XLA: An Optimizing Compiler for High-Performance Computations. GitHub Repository
【13】. AlphaFold Team. (2021). AlphaFold 3: Structure prediction script. GitHub Repository
【14】. NVIDIA Developer Team. (2020). CUDA Programming Guide. NVIDIA Corporation.
【15】. Stream Callback Team. (2021). Stream Callback: Asynchronous data transfer and execution. GitHub Repository