HPC Lab Manual: Distributed Computing and Parallel Programming

Table of Contents

System Access

SSH Connection

```bash
ssh username@master-node
```

Replace `username` with your actual username.

Navigate to Summer School Directory

```bash
cd /mnt/lustre/nabeel/summer_school
```

Directory Listing

```bash
ls -la
```

Directory Structure Overview

The summer school materials are organized in the following hierarchy:

```
/mnt/lustre/nabeel/summer_school/
├── Demonstration_Mnist_Dist/        # Data Level Parallelism (MNIST 70k)
│   ├── data/                  # Dataset files
│   ├── nccl_1259.err           # Error logs
│   ├── nccl_1259.out           # Output logs
│   ├── py_train.slurm          # Python training SLURM script
│   └── train.py               # Training script
├── Dist_DL/                   # Distributed Deep Learning
│   ├── GPU_Off/               # GPU Off examples
│   └── model_shard/           # Model Level Parallelism (Model Sharding)
├── job_sch_new/               # Job Scheduling & Distributed Training (10k dataset)
│   ├── comm_1264.err           # Communication error logs
│   ├── dataset.py             # Dataset handling
│   ├── ft_1263.err            # Fault tolerance error logs
│   ├── ft_1263.out            # Fault tolerance output logs
│   ├── job_sched1.slurm        # Main scheduling script
│   ├── model.py               # Model definition
│   ├── nccl_1261.err          # NCCL error logs
│   ├── nccl_1261.out          # NCCL output logs
│   ├── nccl_1262.err          # Additional NCCL error logs
│   ├── nccl_1262.out          # Additional NCCL output logs
│   ├── __pycache__/           # Python cache
│   ├── run_local.sh           # Local execution script
│   ├── slurm_comm.sh           # Node communication script
│   ├── slurm_ft.slurm         # Fault tolerance SLURM script
│   └── train.py               # Training script with time metrics
├── Job_Sch_Res_Man/           # Job Scheduling Resource Management
├── MPI/                       # Message Passing Interface
├── openmp_prog/               # OpenMP Programming
└── SMP/                       # Symmetric Multi-Processing
```

SLURM Job Scheduler

Basic SLURM Script Structure

Create a file named `job_script.slurm`:

```
bash
```

```bash
#!/bin/bash
#SBATCH --job-name=my_job          # Job name
#SBATCH --nodes=1                  # Number of nodes
#SBATCH --ntasks-per-node=4        # Tasks per node
#SBATCH --cpus-per-task=4          # CPUs per task
#SBATCH --gpus-per-node=1          # GPUs per node (if needed)
#SBATCH --time=01:00:00            # Time limit (HH:MM:SS)
#SBATCH --output=nccl_%j.out       # Output file
#SBATCH --error=nccl_%j.err        # Error file

# Master node address and port
export MASTER_ADDR=192.168.20.15
export MASTER_PORT=29500

# Set Ethernet interface for NCCL
export NCCL_SOCKET_IFNAME=ensif1

# Disable InfiniBand (if not available)
export NCCL_IB_DISABLE=1

# Optional: Enable debug output
export NCCL_DEBUG=INFO
export PYTHONUNBUFFERED=1

# Export distributed environment variables
export RANK
export WORLD_SIZE
export MASTER_ADDR
export MASTER_PORT

echo "RANK=$RANK on $(hostname) using interface $NCCL_SOCKET_IFNAME"

# Run your program
python3 train.py
```

SLURM Parameter Explanation

Parameter	Description	Example Values
--job-name	Name for your job	mnist_training, mpi_test
--nodes	Number of compute nodes	1, 2, 4
--ntasks-per-node	Tasks per node	1, 4, 8
--cpus-per-task	CPU cores per task	1, 4, 8
--gpus-per-node	GPUs per node	1, 2, 4
--time	Maximum runtime	00:30:00, 02:00:00
--partition	Queue/partition name	gpu, cpu, debug
--mem	Memory per node	8G, 16G, 32G

Essential SLURM Commands

Submit a Job

```bash
sbatch job_script.slurm
```

Check Job Status

```bash
squeue                # All jobs
squeue -u username    # Your jobs only
squeue -j jobid       # Specific job
```

Cancel a Job

```bash
scancel jobid         # Cancel specific job
scancel -u username   # Cancel all your jobs
```

Job Information

```bash
sinfo                 # Node information
sacct -j jobid        # Job accounting info
```

Distributed Machine Learning

Data Level Parallelism (MNIST 70,000 Images)

Navigate to the MNIST distributed training directory:

```bash
cd /mnt/lustre/nabeel/summer_school/Demonstration_Mnist_Dist
```

Directory Contents:

- `data/` - Contains the MNIST dataset
- `train.py` - Main training script for distributed learning
- `py_train.slurm` - SLURM batch script for Python training
- `nccl_*.out` - Training output logs
- `nccl_*.err` - Error logs

Running MNIST Distributed Training:

Step 1: Review the SLURM Script

```bash
cat py_train.slurm
```

Step 2: Submit the Job

```bash
sbatch py_train.slurm
```

Step 3: Monitor Job Progress

```bash
# Check job status
squeue -u $USER

# Monitor real-time output
tail -f nccl_*.out

# Check for errors
tail -f nccl_*.err
```

Step 4: View Results

```bash
# View complete output
cat nccl_*.out

# Check training metrics and performance
grep -i "epoch\|loss\|accuracy" nccl_*.out
```

Key Features:

- Uses 70,000 MNIST images for training
- Implements data parallelism across multiple nodes
- NCCL backend for efficient communication
- Automatic logging of training progress

Distributed Deep Learning Options

GPU Off Examples

Navigate to the GPU off directory:

```bash
cd /mnt/lustre/nabeel/summer_school/Dist_DL/GPU_Off
```

Model Level Parallelism (Model Sharding)

Navigate to the model sharding directory:

```bash
cd /mnt/lustre/nabeel/summer_school/Dist_DL/model_shard
```

Running Model Sharding

```bash
# Submit the job
sbatch model_shard.slurm

# Check logs
cat model_shard_*.out
```

Job Scheduling and Time Metrics (10,000 Image Dataset)

Navigate to the job scheduling directory:

```bash
cd /mnt/lustre/nabeel/summer_school/job_sch_new
```

Directory Contents:

- `train.py` - Training script with comprehensive time metrics
- `model.py` - Neural network model definition
- `dataset.py` - Dataset handling and preprocessing
- `job_sched1.slurm` - **Main scheduling script for training**
- `slurm_ft.slurm` - **Fault tolerance SLURM script**
- `slurm_comm.sh` - **Node communication script**
- `run_local.sh` - Local execution script
- `nccl_*.out/err` - Output and error logs
- `ft_*.out/err` - Fault tolerance logs
- `comm_*.err` - Communication logs

Running Job Scheduling Experiments:

Option 1: Standard Distributed Training with Scheduling

```bash
# Submit the main scheduling job
sbatch job_sched1.slurm

# Monitor progress
squeue -u $USER
tail -f nccl_*.out
```

Option 2: Fault Tolerance Training

```bash
# Submit fault tolerance job
sbatch slurm_ft.slurm

# Monitor fault tolerance logs
tail -f ft_*.out
```

Option 3: Node Communication Testing

```bash
```

```
# Run communication script
bash slurm_comm.sh

# Check communication logs
tail -f comm_*.err
```

Option 4: Local Testing

```
bash

# Run locally for debugging
bash run_local.sh
```

Changing Number of Nodes for Performance Testing:

Edit SLURM Scripts to Test Different Node Configurations:

```
bash

# Edit the main scheduling script
nano job_sched1.slurm

# Modify these parameters:
#SBATCH --nodes=1          # Change to 1, 2, 4, 8 nodes
#SBATCH --ntasks-per-node=4 # Adjust tasks per node
```

Example configurations for testing:

Single Node:

```
bash

#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4
```

Two Nodes:

```
bash

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4
```

Four Nodes:

```
bash
```

```
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=2
```

Analyzing Time Metrics:

View Training Time Results:

```bash
# Check training completion times
grep -i "training time\|epoch time\|total time" nccl_*.out

# Compare performance across different node configurations
grep -i "throughput\|images/sec\|batch time" nccl_*.out
```

Performance Comparison Workflow:

1. Submit job with 1 node: `sbatch job_sched1.slurm`
2. Record job ID and wait for completion
3. Edit script to use 2 nodes, submit again
4. Compare time metrics in output files
5. Repeat for 4, 8 nodes to analyze scaling

Key Features:

- Uses 10,000 image dataset for faster experimentation
- Comprehensive time metrics and performance monitoring
- Multiple execution modes (standard, fault tolerance, communication testing)
- Easy node configuration changes for scaling experiments
- Detailed logging for performance analysis

Job Scheduling Resource Management

Navigate to the resource management directory:

```bash
cd /mnt/lustre/nabeel/summer_school/Job_Sch_Res_Man
```

This directory focuses on:

- Resource allocation strategies
- Job queue management

- System resource optimization
- Scheduling algorithms

Parallel Programming with C/C++

MPI (Message Passing Interface)

```bash
cd /mnt/lustre/nabeel/summer_school/MPI
```

Compilation:

```bash
mpicc -o program program.c        # C program
mpicxx -o program program.cpp     # C++ program
```

SLURM Script for MPI:

```bash
#!/bin/bash
#SBATCH --job-name=mpi_job
#SBATCH --nodes=2
#SBATCH --ntasks=8
#SBATCH --time=00:30:00

mpirun -np 8 ./program
```

OpenMP Programming

```bash
cd /mnt/lustre/nabeel/summer_school/openmp_prog
```

Compilation:

```bash
gcc -fopenmp -o program program.c   # GCC
icc -qopenmp -o program program.c   # Intel compiler
```

SLURM Script for OpenMP:

```bash
#!/bin/bash
#SBATCH --job-name=openmp_job
#SBATCH --nodes=1
#SBATCH --cpus-per-task=8
#SBATCH --time=00:30:00

export OMP_NUM_THREADS=8
./program
```

Symmetric Multi-Processing (SMP)

```bash
cd /mnt/lustre/nabeel/summer_school/SMP
```

Compilation:

```bash
gcc -o program program.c          # Basic compilation
gcc -O3 -o program program.c      # Optimized compilation
```

SLURM Script for SMP:

```bash
#!/bin/bash
#SBATCH --job-name=smp_job
#SBATCH --nodes=1
#SBATCH --cpus-per-task=4
#SBATCH --time=00:30:00

./program
```

Common Commands Reference

File Operations

```bash
```

```
ls -la                  # List files with details
cp file1 file2          # Copy file
mv file1 file2          # Move/rename file
rm file                 # Delete file
mkdir dirname           # Create directory
```

Viewing Files

bash

```
cat filename            # Display entire file
head -n 20 filename     # First 20 lines
tail -n 20 filename     # Last 20 lines
tail -f filename        # Follow file updates
less filename           # Page through file
```

Process Management

bash

```
ps aux                  # List all processes
top                     # System monitor
htop                    # Enhanced system monitor
kill -9 PID             # Kill process
```

System Information

bash

```
nvidia-smi              # GPU status
free -h                 # Memory usage
df -h                   # Disk usage
lscpu                   # CPU information
```

Troubleshooting

Common Issues and Solutions

Job Fails to Start

1. Check SLURM script syntax
2. Verify resource availability: `sinfo`
3. Check job queue: `squeue`

Out of Memory Errors

```bash
#SBATCH --mem=32G                # Increase memory allocation
```

GPU Not Detected

```bash
#SBATCH --gpus-per-node=1        # Explicitly request GPU
export CUDA_VISIBLE_DEVICES=0    # Set GPU device
```

NCCL Communication Issues

```bash
export NCCL_DEBUG=INFO           # Enable debug output
export NCCL_IB_DISABLE=1         # Disable InfiniBand
export NCCL_SOCKET_IFNAME=ensif1 # Set network interface
```

Python Path Issues

```bash
export PYTHONPATH=/path/to/modules:$PYTHONPATH
which python3                    # Verify Python location
```

Log File Locations

Check these files for debugging:

- `nccl_*.out` - Standard output
- `nccl_*.err` - Error messages
- `slurm-*.out` - SLURM job output
- Individual directory logs for specific applications

Getting Help

1. Check log files first
2. Use `squeue` to verify job status
3. Review SLURM script parameters
4. Test on smaller datasets/shorter time limits
5. Check system resource availability with `sinfo`

Quick Start Checklist

1. **Connect to HPC**: `ssh username@master-node`
2. **Navigate to directory**: `cd /mnt/lustre/nabeel/summer_school`
3. **Choose your task**:
   - Data parallelism: `cd Demonstration_Mnist_Dist`
   - Model parallelism: `cd Dist_DL/model_shard`
   - Job scheduling: `cd job_sch_new`
   - MPI/OpenMP/CUDA: `cd MPI/` or `cd OpenMP/` etc.
4. **Review the SLURM script**: `cat job_script.slurm`
5. **Submit job**: `sbatch job_script.slurm`
6. **Monitor**: `squeue -u username`
7. **Check results**: `tail -f output_file.out`

Remember to always check the specific requirements and parameters for each exercise in their respective directories!