CS455 Help Sheet

OpenMP Help

OpenMP is an API for parallel programming in shared memory systems.

Common Directives

- `#pragma omp parallel`: Defines a parallel region.
- `#pragma omp for`: Divides loop iterations among threads.
- `#pragma omp sections`: Divides tasks into independent sections.
- `#pragma omp single`: Ensures a block is executed by one thread.
- `#pragma omp critical`: Ensures mutual exclusion.
- `#pragma omp barrier`: Synchronizes threads.
- `#pragma omp reduction(op: list)`: Combines results from threads.

Environment Variables

- `OMP_NUM_THREADS`: Set the number of threads.
- `OMP_SCHEDULE`: Configure loop scheduling (`static`, `dynamic`).

Performance Tips

- Use `OMP_SCHEDULE` for load balancing.
- Avoid false sharing by aligning data structures.
- Minimize critical sections to reduce contention.
- Use `omp_get_wtime` for timing in OpenMP.

Examples

Parallel Loop:

```
#pragma omp parallel for
for (int i = 0; i < n; i++) {
    a[i] = b[i] + c[i];
}
```

Reduction Example:

```
int sum = 0;
#pragma omp parallel for reduction(+:sum)
for (int i = 0; i < n; i++) {
    sum += a[i];
}
```

Critical Section:

```
#pragma omp critical
{
    shared_var += local_val;
}
```

MPI Help

MPI (Message Passing Interface) is used for distributed-memory parallel programming.

Common Functions

- `MPI_Init`: Initialize the MPI environment.
- `MPI_Comm_rank`: Get the rank of the process.
- `MPI_Comm_size`: Get the total number of processes.
- `MPI_Send / MPI_Recv`: Send/receive messages.
- `MPI_Barrier`: Synchronize processes.
- `MPI_Reduce`: Perform a reduction operation across processes.
- `MPI_Bcast`: Broadcast a message from one process to all other processes.
- `MPI_Finalize`: Clean up and exit the MPI environment.

Environment Variables

- `mpirun -np <num_processes> <program>`: Run a program with the specified number of processes.
- `MPICH_DEVICE`: Define the network device for communication.

Performance Tips

- Minimize communication between processes.
- Use non-blocking communication (`MPI_Isend`, `MPI_Irecv`) for overlap.
- Decompose work to minimize idle time among processes.
- Use `MPI_Wtime` for measuring execution time in MPI programs.

Examples

Basic MPI Program:

```
#include <mpi.h>
int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("Hello from rank %d of %d\n", rank, size);
    MPI_Finalize();
    return 0;
}
```

Send, Receive, and Barrier Example:

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    int rank, size, data;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (rank == 0) {  // Sender
        data = 42;  // Example data
        MPI_Send(&data, 1, MPI_INT, 1, 0,
            MPI_COMM_WORLD);
        printf("Process 0 sent data %d to Process
            1\n", data);
    }
    else if (rank == 1) {  // Receiver
        MPI_Recv(&data, 1, MPI_INT, 0, 0,
            MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Process 1 received data %d from
            Process 0\n", data);
    }

    MPI_Barrier(MPI_COMM_WORLD);  // Synchronize all
        processes
    printf("Process %d reached the barrier\n", rank);

    MPI_Finalize();
    return 0;
}
```

C++ Help

C++ is used in HPC for its efficiency and control over system resources.

Common Data Structures and Functions

- `std::vector`: Dynamic arrays.
- `std::unordered_map`: Fast key-value lookups.
- `std::unordered_set`: Unique elements with fast lookups.
- `std::sort`: Sorts elements.
- `std::move`: Transfers data without copying.
- `std::unique_ptr`: Exclusive ownership.
- `std::shared_ptr`: Shared ownership.
- `std::chrono::high_resolution_clock`: Timing function.

Common Flags

- `-O0`: No optimization (default flag).
- `-O1`: Basic optimizations.
- `-O2`: Balanced optimizations.
- `-O3`: Aggressive optimization (may increase compile time and cause unexpected behavior).
- `-Ofast`: More aggressive optimization (may cause unexpected behavior).
- `-g`: Generates debug information.
- `-Wall -Wextra`: Enable extra warnings.

Examples

Chrono Timing Example:

```cpp
#include <iostream>
#include <chrono>

int main() {
    auto start =
        std::chrono::high_resolution_clock::now();
    // Code to time
    auto end =
        std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> elapsed = end -
        start;
    std::cout << "Elapsed time: " << elapsed.count()
        << " seconds\n";
}
```

Lambda Example:

```cpp
auto add = [](int a, int b) -> int {
    return a + b;
};
```

Linux Help

Linux is widely used in HPC.

Basic Commands

- `ls`: List directory contents.
- `pwd`: Present working directory.
- `cd <dir>`: Change directory.
- `cp <src> <dest>`: Copy files.
- `mv <src> <dest>`: Move/rename files.
- `rm <file>`: Delete files.
- `ssh <user>@<host>`: Connect to a remote machine.
- `scp <src> <dest>`: Copy files between local and remote machine.

Git Help

Git is a version control system.

Basic Commands

- `git init`: Initialize a repository.
- `git clone <url>`: Clone a repository.
- `git status`: Display files status.
- `git checkout <branch>`: Switch branch.
- `git add <file>`: Stage changes.
- `git commit -m "message"`: Commit changes.
- `git push`: Push changes to a remote repository.
- `git pull`: Pull changes from a remote repository.

PBS Pro Help

PBS Pro is a job scheduler commonly used on HPC systems.

Common Commands

- `qsub <script>`: Submit a job script.
- `qstat`: Check the status of submitted jobs.
- `qdel <job_id>`: Cancel a running or queued job.
- `pbsnodes -a`: Display information about compute nodes.
- `tracejob <job_id>`: View job history and diagnostic information.

Typical PBS Pro Job Script

```bash
#!/bin/bash
#PBS -N MyJob
#PBS -l nodes=1:ppn=4
#PBS -l walltime=01:00:00
#PBS -q short
#PBS -j oe

# Change to working directory
cd ${PBS_O_WORKDIR}
./my_program
```

Slurm Help

Slurm (Simple Linux Utility for Resource Management) is another popular job scheduler.

Common Commands

- `sbatch <script>`: Submit a Slurm job script.
- `squeue`: View queued and running jobs.
- `scancel <job_id>`: Cancel a running or pending job.
- `sinfo`: Display partition and node information.
- `srun <executable>`: Launch a parallel job step (often used inside scripts).

Typical Slurm Job Script

```bash
#!/bin/bash
#SBATCH --job-name=MySlurmJob
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4
#SBATCH --time=01:00:00
#SBATCH --partition=short

#SBATCH --output=%x.%j.out
#SBATCH --error=%x.%j.err

# Run the program
./my_program
```