High Performance Computing (HPC) - Answer Sheet

Q1.

(a)

i. **Three Advanced Computation Problems Solved Using HPC:**

- **Weather Prediction:** Simulating and predicting complex weather systems requires massive computational power and is accomplished using HPC systems.
- **Molecular Modeling:** HPC helps simulate molecular interactions and protein folding, which are critical in drug discovery.
- **Astrophysical Simulations:** HPC is used to model galaxy formation, black hole dynamics, and other phenomena that require immense data processing.

ii. **HPC in Scientific Computing:**

Scientific computing relies on solving mathematical models and simulations. HPC provides the computational power necessary to handle large-scale simulations. Diagrammatically, scientific computing involves:

- Data Input -> Mathematical Model -> HPC Computation -> Visualization/Results

iii. **Fast CPU Requires Sufficiently Fast Memory:**

Yes, if memory is slow, the CPU remains idle while waiting for data. For example, in matrix multiplication, if data is fetched slowly from RAM, the CPU cannot utilize its full speed, creating a bottleneck.

iv. **Heat Flux Increases with Speed:**

Engineers overcome this by:

- Using heat sinks and cooling systems

- Optimizing power consumption

- Designing processors with better thermal dissipation

(b)

i. **True** - Moore's law correctly states an exponential increase in component density approximately every 18 months.

ii. **True** - Protected mode in IA-32 ensures memory protection among processes.

iii. **False** - 16-bit programs require compatibility mode or emulation; they do not run natively in 64-bit protected mode.

(c)

i. **Task of the Program:** The program likely measures time taken to execute a computation, such as a loop or function, using Python libraries (e.g., `time`, `timeit`).

ii. **Measuring Performance:**

- Run the same program on multiple computers.

- Record execution time on each.

- Compare times to determine which computer is faster or more efficient.

Q2.

(a)

i. **Massively Parallel Processors Needed:**

To solve large problems efficiently by dividing them into smaller tasks processed simultaneously, reducing total execution time.

ii. **Tesla vs Fermi Architecture:**

- **Tesla:** Focused on high parallelism, lacked error correction.
- **Fermi:** Introduced ECC memory, improved cache, and was more programmable.

iii. **CUDA Function Declarations:**

- `__global__`: Called from host, runs on device.

```cpp
__global__ void add(int *a, int *b, int *c) {
    c[threadIdx.x] = a[threadIdx.x] + b[threadIdx.x];
}
```

- `__device__`: Called and executed on device.

```cpp
__device__ int square(int x) {
    return x * x;
}
```

- `__host__`: Called and executed on host (default).

```cpp
__host__ void print() {
    printf("Hello CPU\n");
}
```

(b)

i. **True** - Kernel calls are asynchronous by default.

ii. **True** - GPU threads require minimal overhead compared to CPU threads.

iii. **True** - Each block contains multiple threads.

iv. **True** - Threads from different blocks cannot synchronize or share memory directly.

(c)

i. **Processing Flow in CUDA:**

- Allocate memory on host and device

- Copy data to device

- Launch kernel

- Copy result back to host

- Free memory

ii. **Tasks of Code Statements:**

- Memory allocation (cudaMalloc)

- Copying data (cudaMemcpy)

- Launching kernel (kernel<<<blocks, threads>>>)

- Freeing memory (cudaFree)

Q3.

(a)

i. **OpenMP vs MPI:**

- OpenMP: Shared memory, thread-based

- MPI: Distributed memory, process-based

ii. **Three Issues in Parallel Programming:**

- Data dependency

- Synchronization

- Load balancing

iii. **Amdahl's Law:**

Even with infinite processors, the non-parallel portion limits speedup. E.g., if 30% of task is serial, maximum speedup is ~3.33x.

iv. **MPI Terms:**

- **mpi4py:** Python bindings for MPI

- **Processes:** Independent instances executing in parallel

- **Ranks:** Unique IDs for each process

(b)

i. **True** - Each SPMD processor runs the same code on different data.

ii. **False** - Send/Recv are not always paired, especially with collective operations.

iii. **False** - Standard Send/Recv are blocking unless explicitly set to non-blocking.

(c)

[Python MPI Calculator Code included in response above.]

Q4.

(a)

i. **Cluster Systems Cost-Effective:**

Clusters use commodity hardware and can be scaled incrementally, making them cheaper than building a single supercomputer.

ii. **Limitations of Virtual Clusters:**

- Network latency between VMs

- Limited resource allocation

- Performance degradation due to host sharing

iii. **Ways to Create High-Performance Programs:**

- Optimize algorithm and data structures

- Use parallelism (multi-threading, GPU)

- Reduce I/O and memory overhead

iv. **Linux Commands:**

1. `sudo chown mpiuser:mpiuser /path/to/shared/dir` - Change ownership of shared directory

2. `sudo ufw allow from 192.168.1.0/24` - Allow firewall access from local network

3. `mpiexec -n 3 /home/mpiuser/bin/cpi` - Run program `cpi` using 3 MPI processes

(b)

i. **False** - Interconnect speed and software overheads also affect performance

ii. **True** - HPC clusters can include diverse nodes (CPUs, GPUs, FPGAs)

iii. **False** - `nvtop` monitors GPU usage, not network activity

(c)

i. **Maximum Speedup:**

Using Amdahl's law:

S = 1 / (0.3 + (0.7 / 8)) = 1 / (0.3 + 0.0875) = 1 / 0.3875 ~ 2.58x

ii. **Actual Time:**

Time = 200 / 2.58 ~ 77.52 seconds