Day4

Table of Contents

1. PI calculator Serial

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<sys/time.h>

#define N 999999999

int main()
{
    int i, j;
    double area, pi;
    double dx, y, x;
    double exe_time;

    struct timeval stop_time, start_time;

    dx = 1.0/N;
    x = 0.0;
    area = 0.0;

    gettimeofday(&start_time, NULL);

    for(i=0;i<N;i++)
```

```
    {
        x = i*dx;
        y = sqrt(1-x*x);
        area += y*dx;
    }

    gettimeofday(&stop_time, NULL);
    exe_time = (stop_time.tv_sec+(stop_time.tv_usec/1000000.0)) - (start_time.tv_

    pi = 4.0*area;
    printf("\n Value of pi is = %.16lf\n Execution time is = %lf seconds\n", pi,

    return 0;
}
```

1.1. Compile

```
gcc -o pi1.out pi1.c -lm
```

1.2. Run

```
./pi1.out
```

```

Value of pi is = 3.1415926555902138
Execution time is = 3.672017 seconds
```

2. PI calculator Parallel

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <pthread.h>
#include <sys/time.h>

#define N 999999999
#define T 20 // Number of threads

double dx;
pthread_mutex_t mutex;
double area = 0.0;

struct thread_data {
    int start;
    int end;
};

void *compute_area(void *arg) {
    struct thread_data *data = (struct thread_data *)arg;
    double local_area = 0.0;
```

```
    double x, y;

    for (int i = data->start; i < data->end; i++) {
        x = (i + 0.5) * dx; // Midpoint of the interval
        y = sqrt(1 - x * x);
        local_area += y * dx;
    }

    // Acquire mutex before updating global area
    pthread_mutex_lock(&mutex);
    area += local_area;
    pthread_mutex_unlock(&mutex);

    pthread_exit(NULL);
}

int main() {
    pthread_t threads[T];
    struct thread_data thread_data_array[T];
    double pi, exe_time;
    struct timeval start_time, stop_time;

    dx = 1.0 / N;

    gettimeofday(&start_time, NULL);

    // Initialize mutex
    pthread_mutex_init(&mutex, NULL);

    // Create threads
    for (int t = 0; t < T; t++) {
        thread_data_array[t].start = t * (N / T);
        thread_data_array[t].end = (t + 1) * (N / T);
        pthread_create(&threads[t], NULL, compute_area, (void *)&thread_data_arr
    }

    // Join threads
    for (int t = 0; t < T; t++) {
        pthread_join(threads[t], NULL);
    }

    // Destroy mutex
    pthread_mutex_destroy(&mutex);

    gettimeofday(&stop_time, NULL);
    exe_time = (stop_time.tv_sec + (stop_time.tv_usec / 1000000.0)) - (start_tim

    pi = 4.0 * area;
    printf("\nValue of pi is = %.16lf\nExecution time is = %lf seconds\n", pi, e

    return 0;
}
```

2.1. Compile

```
gcc -o pi_parallel.out pi_parallel.c -lm -lpthread
```

2.2. Run

```
./pi_parallel.out
```

```


Value of pi is = 3.1415926535799135
Execution time is = 0.596960 seconds
```

3. Introduction to DPDK and SPDK

3.1. DPDK (Data Plane Development Kit)

DPDK is a set of libraries and drivers for fast packet processing. It enables user-space applications to perform low-latency and high-throughput networking. It's widely used in telecom, data centers, and other networking solutions.

3.1.1. Key Points:

- High Performance: Achieves high packet processing rates.
- User-Space Drivers: Bypasses the kernel network stack, leading to reduced latency.
- Core Components: Includes libraries for memory management, queues, ring buffers, and poll mode drivers for various NICs.

3.2. SPDK (Storage Performance Development Kit)

SPDK is a set of tools and libraries for writing high-performance, scalable, user-mode storage applications. It leverages DPDK for high-speed, low-latency operations.

3.2.1. Key Points:

- High Performance: Focused on NVMe and NVMe over Fabrics (NVMe-oF).
- User-Space Drivers: Provides user-space NVMe drivers, bypassing the kernel.
- Core Components: Includes libraries for NVMe, NVMe-oF, iSCSI, vhost, and blob storage.

4. Detailed Overview and Resources

4.1. DPDK

4.1.1. 1.1 What is DPDK?

- Definition: DPDK is a set of libraries and drivers for fast packet processing in user space.
- History: Originally developed by Intel and now an open-source project under the Linux Foundation.
- Importance: Enables high-throughput and low-latency networking, essential for data centers, telecoms, and enterprise networks.

4.1.2. 1.2 DPDK Architecture

- Core Components:
  - EAL (Environment Abstraction Layer): Provides a standard interface for hardware and memory operations.
  - MBUF Library: Manages memory buffers used for packet storage.
  - RTE Ring: Implements lockless ring buffers.
  - Poll Mode Drivers (PMD): Provides drivers for NICs, bypassing the kernel network stack.
- User-Space vs Kernel-Space:
  - User-Space: Reduces context switches, leading to lower latency and higher throughput.
  - Kernel-Space: Traditional networking stack with higher latency due to kernel overhead.

4.1.3. 1.3 Setting Up DPDK

- Hardware Requirements: Modern CPUs with support for large page memory.
- Software Requirements: Compatible Linux kernel, GCC, and make.
- Installation Steps:
  1. Download the DPDK source code.
  2. Compile the DPDK libraries.
  3. Load necessary kernel modules and configure hugepages.
  4. Bind NICs to DPDK-compatible drivers.

4.2. SPDK

4.2.1. 2.1 What is SPDK?

- Definition: SPDK is a set of tools and libraries for writing high-performance, scalable, user-mode storage applications.

- History: Developed by Intel to improve storage performance and efficiency.
- Importance: Provides user-space NVMe drivers and libraries for building high-performance storage solutions.

4.2.2. 2.2 SPDK Architecture

- Core Components:
  - NVMe Driver: User-space driver for NVMe devices.
  - NVMe-oF Target: Implements NVMe over Fabrics.
  - iSCSI Target: Provides iSCSI target functionality.
  - Blobstore: Lightweight, user-space blob storage library.
- Integration with DPDK: Uses DPDK for memory management and networking.

4.2.3. 2.3 Setting Up SPDK

- Hardware Requirements: NVMe devices, modern CPUs.
- Software Requirements: Compatible Linux kernel, GCC, and make.
- Installation Steps:
  1. Download the SPDK source code.
  2. Compile the SPDK libraries.
  3. Configure the environment for hugepages and bind NVMe devices to SPDK-compatible drivers.

Author: Abhishek Raj
Created: 2024-06-28 Fri 14:15