Design Document

Design Document for Benchmarking

CPU

The program takes following command line arguments:

- Total operation: Number of time Integer or floating point operation are carried out. For ex. 40000000
- Statistical Mode: This is to command that whether code will create statistical data or not. If provided 1 then it will create statistical data, if 0 is provided then it will do normal benchmarking.

It has following major functions:

- perform_benchmark(float totalOperation, int threadCount, int isStatistical, int turn)

Input:

totalOperation: As provided via command line

threadCount: The number of thread that will be created

isStatistical: Whether to generate statistical data or do benchmarking

turn: 0 for Integer, 1 for floating point. Applicable only if isStatistical is set as 1

Work:

It open number of thread and check the duration in which number of operations are performed. totalOperation is divided between threads so that threads can share work.

- iops_thread(void *args)

Input:

```
typedef struct{
        long int min;
        long int max;
}Sequence;
```

Work:

It take the generic pointer as input which is cast to above structure and then each thread perform Integer operation max-min times.

- flops_thread(void *args)

Input:

```
typedef struct{
        long int min;
        long int max;
}Sequence;
```

Work:

It take the generic pointer as input which is cast to above structure and then each thread perform Floating point operation max-min times.

Memory

The program takes following command line arguments:

- Loop time: It multiplies the operating time. Provide 1 for normal operations. In case the duration of program is less then increase this parameter.

It has following major functions:

- memory_benchmark(long int blockSize, int loop)

Input:

blockSize: It is the block size for ex. 8KB, 8MB, 80MB

Loop: It is the command line argument passed to the program

Work:

It call throughput() with varying number of threads.

- sequential_read(void *args)

It read sequentially from the memory.

- sequential_write(void *args)

It writes sequentially to the memory.

- random_read(void *args)

It read randomly from the memory

- random_write(void *args)

It writes randomly to the memory

- throughput(long int blockSize, int loopTime, int latency, int threadCount)

Input:

blockSize: size of block for which throughput is calculated

loopTime: same as command line argument

Latency: 1 if latency is calculated or 0 if performing normal benchmark

threadCount: Number of thread to be created

Work:

It creates thread for normal_read, normal_write, random_read, random_write and calculates the time for the benchmark.

Disk

The program takes following command line arguments:

- Loop time: It multiplies the operating time. Provide 1 for normal operations. In case the duration of program is less then increase this parameter.

It has following major functions:

- disk_benchmark(long int blockSize, int loop)

Input:

blockSize: It is the block size for ex. 8KB, 8MB, 80MB

Loop: It is the command line argument passed to the program

Work:

It call throughput() with varying number of threads.

- sequential_read(void *args)

It read sequentially from the file.

- sequential_write(void *args)

It writes sequentially to the file.

- random_read(void *args)

It read randomly from the file.

- random_write(void *args)

It writes randomly to the file.

- throughput(long int blockSize, int loopTime, int latency, int threadCount)

Input:

blockSize: size of block for which throughput is calculated

loopTime: same as command line argument

Latency: 1 if latency is calculated or 0 if performing normal benchmark

threadCount: Number of thread to be created

Work:

It creates thread for normal_read, normal_write, random_read, random_write and calculates the time for the benchmark.

Network

It has two parts. One is server and client. Let us see each of them.

Server

It takes following command line argument:

- Port_no: It is the port on which server will run.
- Thread_count: It is the number of thread to be opened

It has following important functions:

- invoke_server(int serverPort, int threadCount)

Input:

serverPort: port on which server is running

threadCount: Number of thread to be opened

Work:

It creates thread which listen for incoming connection

- server_thread(void *args)

Input:

```
typedef struct{
    int serverFd;
    int threadIndx;
}OperationInfo;
```

Work:

It takes generic pointer to above structure and then listen to the port for incoming connection and send message to client and then client resent the same message.

Client

It takes following command line argument:

- ip: The ip address on which server is running.
- Port No: Port on which server is listening
- Thread Count: Number of thread server has created

It has following important functions:

- invoke_client(int threadCount)

Input:

threadCount: Number of thread to be opened

Work:

It creates thread which communicates with the server

- client_thread(void *args)

Input:

None

Work:

It connects to the server and then receive message from the server and then resend the same message