Abhinav Pimpalgaonkar(A20387324)

Sandeep Vuzzini(A20243379)

CS553 PA1 Fall' 2017

PA1-Design Document

Introduction

This report aims at describing the overall program design and design tradeoffs considered. It also describes the possible improvements and extension to the programs. Benchmarking experiment includes various experiments on CPU, Memory, Disk and Network.

General Consideration

- All the experiments were developed in C language with strong scalability and multithreading.
- Pthread library was used for multithreading.
- Chameleon Cloud KVM instance () was used to run all the experiments
- All the programs require user parameters to run the experiments. The parameters vary from program to program. It can be number of threads, block size, operation to be performed or type of network server.
- Time was measured using clock () function. The difference of start time and end time was used to calculate metrics like GFlops, GIops, latency or throughput.

CPU Benchmarking

- Experiment was performed for floating point and integer arithmetic operation like addition, subtraction etc.
- AVX instructions were used for floating and integer operations.
- Private variables of all the threads will be initialized first and then all the thread will perform operations
- Code runs a loop based on number of threads for scalability and performed operations
- Scalability is achieved by dividing the iterations equally among all threads and each thread will perform operations on data parallelly.

Memory Benchmarking

- Various experiments were performed based on thread count, block size and operations to be performed.
- Pthread mutex lock was used to avoid deadlocks between threads while reading and writing to/from memory
- An array of specific size was created so that read+write operation is performed on that array as one operation (Copying an array to another array includes both read+write)
- Bytes to be read/written in the memory was equally divided among the threads and each thread will perform read/write operation based on block size.
- Scalability was achieved by dividing the bytes to be written/read from memory among threads and maintaining a position variable so that each thread will perform operations on the specific range of locations.
- Functions like memcpy() and memset() was used to read/write.

Disk Benchmarking

- Various experiments were performed based on thread count, block size and operations to be performed.

- Pthread mutex lock was used to avoid deadlocks between threads while reading and writing to/from disk
- A file of specific size was created so that read+write operation is performed on that file as one operation
- Each thread is writing fixed amount of bytes in the file and then reading those content before releasing the lock (for read+write operation)
- For operations like sequential read and sequential write a file is created first so that each thread will perform read operation within a specific range.
- Reading is performed as follows: data is written first in buffer from file and then threads are reading the data from buffer
- Writing operation is performed as follows: data is first written the buffer and then data is transferred to the disk.
- A position variable is maintained and lseek () function is used to retrieve the file offset or cursor position.
- Bytes to be read/written in the disk was equally divided among the threads and each thread will perform read/write operation based on block size.
- Scalability was achieved by dividing the bytes to be written/read from/to disk among threads and maintaining a position variable so that each thread will perform operations on the specific range of locations.
- Functions like memcpy() ,memset(), read() ,write () and lseek() was used to perform read/write to disk.

Network Benchmarking

- Server and Client were created for TCP and UDP.
- Socket Programming was used to create server and client
- Both server and client supports multithreading.
- Pthread mutex lock was used so as avoid deadlock and handle thread synchronization between server and client.
- The amount of data to be transferred is divided among the thread to achieve scalability
- Thread synchronization works as follows : bytes to be transferred is divided equally among threads on client and also bytes received by server will be divided by the threads on servers
- 8 bytes is transferred at a time
- Bytes to be sent and received is kept fixed for both server and client

Improvements

- Clock timing can be improved by improving the accuracy of clocks in nanoseconds(example using gettimeofday function in c).
- Benchmarking experiments can be improved by isolating the effect of cache.
- We should find an alternate way to generate the random numbers as the rand() function takes time.
- There is high possibility that packets might be loss during the transmission. We should find an effective way to avoid loss of packets
- We can use compiler optimization to improve performance.
- Scripts can be created to automate the experiments

Extensions

- For network we can extend our experiment on two nodes with different networks
- All the experiments can be extended for a large block size.
- We can test our experiment for various library functions

References:

AVX

- https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX
- https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions
-

Socket Programming

- http://www.cs.dartmouth.edu/~campbell/cs50/socketprogramming.html
- https://www.codeproject.com/Articles/586000/Networking-and-Socket-programming-tutorial-in-C
- https://www.cs.rutgers.edu/~pxk/417/notes/sockets/udp.html

Multithreading

- https://randu.org/tutorials/threads/#pthreads
- http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html
- http://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html
- https://vcansimplify.wordpress.com/2013/03/08/pthread-tutorial-simplified/

File Handing

- http://www.studytonight.com/c/file-input-output.php
- http://pubs.opengroup.org/onlinepubs/009695399/functions/fcntl.html
- http://man7.org/linux/man-pages/man2/fcntl.2.html

Memory Functions

- https://www.tutorialspoint.com/c_standard_library/c_function_memset.htm
- https://embeddedartistry.com/blog/2017/3/7/implementing-memset-memcpy-and-memmove

Chameleon Cloud

- https://www.chameleoncloud.org/docs/getting-started/

Abhinav Pimpalgaonkar(A20387324)

Sandeep Vuzzini(A20243379)

CS553 PA1 Fall' 2017