# DESIGN DOCUMENT


## NAME: PUNEET SINGH

## CWID: A20413330

# CPU BENCHMARK:

- This benchmark is written to measure the performance of Intel Xeon E5 processor of Haswell family.
- The benchmark takes type of operation and number of threads as input parameters from dat files. Strong scaling is performed on 1 trillion operations with number of threads to achieve maximum performance.
- Type of operations include Double Precision, Single Precision , Half Precision, and Quatre precision . The operations are being performed using AVX and FMA instructions.
- After completion of 1 Trillion operations , performance is measured in Gigaflops by dividing total no operations with measured time and efficiency of this measurement is calculated against theoretical values of the processor.

Main () Function:

- Main function reads the .dat files and created pthreads to execute specified type of operation. Threads call specific function for specific operation.
- After execution of thread main function writes actual and theoretical values in output file.

dp-operations():

- Performs 1 trillion Double precision operations using FMA instructions

sp-operations():

- Performs 1 trillion Single precision operations using FMA instructions.

hp-operations():

- Performs 1 trillion Half precision operations using AVX2 instructions.

qp-operations():

- Performs 1 trillion Quater precision operations using AVX2 instructions.

AVX instructions:

- We have used 256 bit wide vectors which can bundle 4 double precision values. These values can be set using various functions _mm256_set_pd for double precision, _mm256_set_ps single precision, _mm256i_set1_epi16 for short integers and _mm256i_set1_epi8 for quatre precision.

FMA instructions:

- We have used FMA instructions which provide capability to perform two type of operation on 256 bit wide vector and two of them in one cycle i.e. instruction _mm256_fmadd_pd(a,b,c) to perform 16 DP operations in one cycle, _mm256_fmadd_ps(a,b,c) to perform 32 SP operations in one cycle.

Calculating no of Flops:

- Total number of operations is divided by no of flops performed in one cycle. For Double precision no of flops is 16 using FMA instructions i.e performing 2(multiply and addition) operations on 4 DP values and 2 of these instructions in one cycle.
- For Single Precision operation we have used FMA instruction which performs 32 SP operations in one cycle .
- For Half precision FMA instructions are are not defined thus we use AVX instructions to perform equivalent no of flops . _mm256_add_epi16 instructions used in half precision is capable of performing 16 operation in one cycle and using 4 of these gives 64 flops. Though these 64 flops are not performed in one cycle , but we will have to consider them equivalent to FMA instructions.
- Similarly for quater precision AVX instruction _mm256_add_epi8 is capable of performing 32 quater precision operation in one cycle and using 4 of them will give 128 flops . Though these 128 flops are not performed in one cycle , but we will have to consider them equivalent to FMA instructions.

LINPACK Benchmark:

- Linpack is well known benchmark for measuring cpu performance . We have downloaded linpack tar file , modified and executed the benchmarks as per the instructions , output screenshots and instructions to execute are given in execution guide.

## MEMORY BENCHMARK:

- Memory benchmarks are written to measure throughput performance and latency of memory for fixed size of data and iterating over it multiple time with different blocksize.
- Size of data and size of Blocksize is carefully taken to minimize the cache hits and efficiently measure performance of memory.
- We have used different access patterns for accessing memory random and sequential read. The benchmarks are fully strong scaled and each threads have been provided with memory offset and bounded by amount of memory called sizeperthread. Each thread takes blocksize values and perform memcpy for that size and this logic is iterated 100 times.
- After completion of all the thread we calculate the total time lapsed and total amount of data which gives throughput of the memory.

Main function ():

- Main function receives dat file and and reads all the input values . Creates thread and individual thread is assigned memory offset and thread id. Then we calculate size per thread. These threads call functions for specific memory operation.

rws_operation():

- ❏ This function performs sequential read write , each thread calls the function parallelly and executes memcpy() for defined blocksize. The sequential logic is as follows , every thread has a structure passed with it as parameter this structure has offset , threadid and size limit for the thread. Then we calculate no of blocks and iterate over memcpy that many times. With every iteration threre is increment of blocksize.

rwr_operation():

- ❏ This function performs random read write on the memory. Execution of the function is similar , but logic for random read and write is different. We generate a random no which is between o to no of blocks , and we ensure that the inner loop runs for number of blocks times. The random number generated is used to memcpy at that location.

calc_latency_rwr():

- ❏ In this function we are performing random read and write for 1 byte of data and total 100 million operations. Random read read and write logic is same and we calculate total time and latency is calculated after all the threads have finished execution.

calc_latency_rws():

- ❏ This function we are performing sequential read and write for 1 byte of data and total 100 million operations. Sequential read write logic is same as rws_operation.

Value calculation:

- ❏ After completion of operations we measure time and calculate throughput and latency for the measured time by dividing total data (100 GB) by time for throughput and latency is calculated dividing total time with total operation (1 million).

PMBW BENCHMARK:

- ❏ This is a well known benchmark to measure memory performance. The benchmark was already installed on the cluster. Execution screenshots and instructions to execute and be found on execution guide.

## DISK BENCHMARK:

- ❏ Disk benchmark are written to measure throughput performance and latency of disk for fixed size of data and iterating over it multiple time with different blocksize.
- ❏ We have used blocksize of 1MB, 10MB, 100MB and multiple thread 1, 2, 4 for strong scalling.
- ❏ We have used different access patterns for accessing disk random and sequential read and write. The benchmarks are fully strong scaled and each threads have been provided with memory offset and bounded by amount of memory called sizeperthread. Each thread takes blocksize values and perform random and sequential read write for that size and this logic run over 10 GB file size.
- ❏ After completion of all the thread we calculate the total time lapsed and total amount of data which gives throughput of the disk.

Main function():

- ❏ Main function receives dat file and and reads all the input values .Creates thread and individual thread is assigned memory offset and thread id. Then we calculate size per thread. These threads call functions for specific disk operation.

rr_operation():

- ❏ This function reads randomly from the disk. Every thread calling this function has its own file pointer and memory offset to point the file pointer to that location. Once the memory offset is set , we generate random and drag the file pointer to that particular location in order to perform I/O operation . SEEK_SET seeks the file pointer from the begining of the file. Thus the the random variable is multiplied by the blocksize and added to the offset.

rs_operation():

- ❏ This function performs sequential read from the disk. Every thread calling this function has its own file pointer and memory offset to point the file pointer upto that location. Since fread automatically sets the file pointer to the end of the memory space there is no need of using fseek.

ws_operation():

- ❏ This function perform sequential write to the disk.we have defined size per thread and with blocksize we calculate no of blocks per thread or no of write operations per thread. The loop is iterated upto number of blocks and for each block we are performing write operation using fwrite .

wr_operation():

- ❏ For performing random write we follow the same logic we calculate no of blocks per thread and generate random number upto no of threads. This value is added to the offset to drag the pointer to the particular location on the file.

Value calculation ():

- ❏ After performing operations we will calculate total time , throughput and latency for no of operation the measured time by dividing file size (10 GB) by time for throughput and latency is calculated dividing total time with total operation (1 million).

IOZONE():

- ❏ IOZONE is a well known benchmark for measuring disk performance. IOZONE is already installed on the cluster , output screenshot and execution instruction can be found on execution guide.

<u>NETWORK BENCHMARK:</u>

- ❏ Network benchmark is written to measure the performance of the network in terms of bandwidth and latency. In this benchmark we are measuring performance of network for two protocols TCP and UDP.

- We have used socket programming in C to implement the benchmark . Both the client and server are written in same file . However while running the benchmarks we are executing the server on one machine and client on one machine.
- TCP protocol is connection oriented ie it requires the client to connect to the server . Since we are executing server and client on different machine on cluster we are providing server IP to the client using shell script , which can be found with code.
- In order to differentiate the execution of server and client we are passing a parameter to the executable on the basis of which server or client function is called.

tcp_server():

- Creates server socket and bind the socket to an address. Once bind is done server listens on the socket for incoming connection and accepts the incoming connection once received. After connection is estaiblished transfer of data takes place. Tcp server uses recv function to receive packets coming from the client.

tcp_client():

- This function creates multiple socket and usses these socket to connect it to the server address created from server IP address and port nos. Once server accepts the connection client start sending the data to the server using these sockets.

send_data():

- This function is called by the tcp client to send data to the server. Total data sent is over the internet is 1 gb file and iterated over 100 times.

recv_data():

- This function is called by tcp server after accepting the connection and it receives data from multiple client sockets.

server_ping()

- Server uses this function to receive and send back 1B of data to the client.

Client_ pong()

- Client uses this function to send and receive 1B of data to the client.

udp _server():

- Creates a server socket and binds the socket with the server address so that server could receive udp packets sent from the client.

udp_client():

- This function is a udp client and it does not tries to connect to the server rather , it created multiple sockets and start sending data to the server using these sockets.

Values calculation:

- At the end of data transfer total time spent on sending and receiving the data is measured and throughput and latency of the network is calculated.

IPERF BENCHMARK:

- This is the well known benchmark for the measuring network performance and latency. Execution screenshots and instruction to execute can be found in execution guide.