

Mannual Report

Name : Puneet Singh

CWID: A20413330

CPU Benchmarking:

1.Steps to Run CPU Benchmark :

- Use Makefile to compile MyCpuBenchmark.c and name output file mybenchmark.
- Once compiled there are two scripts run1.sh and run2.sh can be used to post jobs on slurm parallelly.
- Run1.sh can be used for QP and HP operations and Run2.sh can be used for SP and DP operations. These scripts use sbatch command to run config files .
- There is individual config file for each operations and for corresponding thread. E.g config_DP_4thread.slurm
- Each config file has #SBATCH argument and runs the executable.
- Dat file and output_cpu_benchmark.dat are used as input to the executable, Dat files has entry for each operation .
- After the execution output is written into output_cpu_benchmark.dat in formatted way.

2.Steps to run Linpack:

- Extract the TAR file and open the specified folder:
- /exports/home/psingh52/PA1/Linpack/benchmarks_2018/linux/mkl/benchmarks/linpack
- Modify lininput_xeon64 to modify no of test , problem size, and no of time to run the test

```
File Edit View Search Terminal Help
Shared-memory version of Intel(R) Distribution for LINPACK* Benchmark. *Other names and brands may be claimed as the property of others.
Sample data file lininput_xeon64.
# number of tests
300000 200000 10000 # problem sizes
300000 200000 10000 # leading dimensions
2 2 2 # times to run a test
15 15 15 # alignment values (in KBytes)
~
~
~
~
~
~
~
```

- Create a net script file to execute linpack.

```
File Edit View Search Terminal Help
#!/bin/bash
export OMP_NUM_THREADS=1
./runme_xeon64
~
~
~
~
~
~
~
~
```

-
- Run linpack-run.sh file.
- It generates output file in the same folder named slurm-12389.out

This is a SAMPLE run script for running a shared-memory version of Intel(R) Distribution for LINPACK* Benchmark. Change it to reflect the correct number of CPUs/threads, problem input files, etc..

*Other names and brands may be claimed as the property of others.

./runme_xeon64: 35: [: -gt: unexpected operator

Sat Mar 24 17:11:38 UTC 2018

Sample data file lininput_xeon64.

Current date/time: Sat Mar 24 17:11:38 2018

CPU frequency: 3.088 GHz

Number of CPUs: 2

Number of cores: 2

Number of threads: 1

Parameters are set to:

Number of tests: 3

Number of equations to solve (problem size) : 300000 200000 10000

Leading dimension of array : 300000 200000 10000

Number of trials to run : 2 2 2

Data alignment value (in Kbytes) : 15 15 15

Maximum memory requested that can be used=800215360, at the size=10000

===== Timing linear equation system solver =====

Size	LDA	Align.	Time(s)	GFlops	Residual	Residual(norm)	Check
10000	10000	15	17.469	38.1740	1.041791e-10	3.673460e-02	pass
10000	10000	15	16.951	39.3409	1.041791e-10	3.673460e-02	pass

Performance Summary (GFlops)

Size	LDA	Align.	Average	Maximal
10000	10000	15	38.7574	39.3409

Residual checks PASSED

End of tests

Done: Sat Mar 24 17:12:18 UTC 2018

~

~

~

"slurm-12940.out" 41L, 1368C

We can take Gflops values from the output file and store it in table .

Memory Benchmark:

1. Steps to execute Mem Benchmark:

- Use Makefile to compile MyMemBenchmark.c and name output file mymembenchmark.
- Once compiled there are two scripts run1.sh, run2.sh and run3.sh can be used to post jobs on slurm parallelly. Run each script individually one at a time for RWRand RWS
- These scripts use sbatch command to run config files.
- There is individual config file for each operations and for corresponding thread. E.g config_RWR_1000_1thread.slurm this can be created using mem-config-creator.sh
- Each config file has #SBATCH argument and runs the executable.
- Dat file and output_mem_benchmark.dat are used as input to the executable, Dat files has entry for each operation .
- After the execution output is written into output_mem_benchmark.dat in formatted way.

2. Steps to execute Pmbw:

- Create a script to execute pmbw name pmbw1.sh

```
File Edit View Search Terminal Help
~/bin/bash
pmbw -p 1 -P 1 -s 1K -S 1K -M 1G -f PermRead32SimpleLoop
~
~
~
~
~
~
~
~
~
~
```

- Execute the script it runs the pmbw for all the parameter listed -s for size and -p for threads Permread32Simpleloop is for random read.
- Pmbw creates output file in same folder named as e.g slurm-15343.slurm.

```

Running benchmarks with at least 4 threads.
Running benchmarks with up to 4 threads.
Running benchmarks with array size at least 1024.
Running benchmarks with array size up to 1024.
Setting memory limit to 1073741824.
Running only functions containing 'PermRead64SimpleLoop'
CPUID: mmx sse avx
Detected 3951 MiB physical RAM and 2 CPUs.

Allocating 512 MiB for testing.
Skipping ScanWrite64PtrSimpleLoop tests
Skipping ScanWrite64PtrUnrollLoop tests
Skipping ScanRead64PtrSimpleLoop tests
Skipping ScanRead64PtrUnrollLoop tests
Skipping ScanWrite64IndexSimpleLoop tests
Skipping ScanWrite64IndexUnrollLoop tests
Skipping ScanRead64IndexSimpleLoop tests
Skipping ScanRead64IndexUnrollLoop tests
Skipping ScanWrite128PtrSimpleLoop tests
Skipping ScanWrite128PtrUnrollLoop tests
Skipping ScanRead128PtrSimpleLoop tests
Skipping ScanRead128PtrUnrollLoop tests
Skipping ScanWrite256PtrSimpleLoop tests
Skipping ScanWrite256PtrUnrollLoop tests
Skipping ScanRead256PtrSimpleLoop tests
Skipping ScanRead256PtrUnrollLoop tests
Skipping ScanWrite32PtrSimpleLoop tests
Skipping ScanWrite32PtrUnrollLoop tests
Skipping ScanRead32PtrSimpleLoop tests
Skipping ScanRead32PtrUnrollLoop tests
Running nthreads=4 factor=1073741824 areaset=1024 thrsize=256 testsize=1024 repeats=4194304 testvol=4294967296 testaccess=536870912
Make permutation: filling permuting cycle=32 filling permuting cycle=32 filling permuting cycle=32 filling permuting cycle=32
run time = 0.203650 -> rerunning test with repeat factor=7908367640
Running nthreads=4 factor=7908367640 areaset=1024 thrsize=256 testsize=1024 repeats=30892062 testvol=31633471488 testaccess=3954183936
Make permutation: filling permuting cycle=32 filling permuting cycle=32 filling permuting cycle=32 filling permuting cycle=32
run time = 1.4459 -> next test with repeat factor=8204241627
RESULT dateline=2018-03-25 04:15:40 host=greencompute-7 version=0.6.2 funcname=PermRead64SimpleLoop nthreads=4 areaset=1024 threads=256 testsize=1024 repeats=30892062 testvol=31633471488
33471488 testaccess=3954183936 areal=1.4459047346205070569 bandwidth=21877977672.870388031 rate=3.6566451066088875846e-10
Skipping PermRead64SimpleLoop test with 2048 maximum array size due to -S 1024.
Skipping PermRead64SimpleLoop test with 3872 maximum array size due to -S 1024.
Skipping PermRead64SimpleLoop test with 4896 maximum array size due to -S 1024.
Skipping PermRead64SimpleLoop test with 6144 maximum array size due to -S 1024.
Skipping PermRead64SimpleLoop test with 8192 maximum array size due to -S 1024.
Skipping PermRead64SimpleLoop test with 12288 maximum array size due to -S 1024.
"slurm-15521.out" 100L, 7334C

```

7,1 Top

This file provide bandwidth value in bytes/sec convert to GB/sec

Disk Benchmark:

1. Steps to execute Disk Benchmark:

- Use Makefile to compile MyDiskBenchmark.c and name output file mydiskbenchmark.
- Once compiled there are five scripts run1.sh, run2.sh, run3.sh, run4.sh and run5.sh can be used to post jobs on slurm parallelly. Run each script individually one at a time for RR, RS, WS, WR operations.
- Run6.sh can be used to trigger jobs for measuring latency i.e RR and WR with 1KB data.
- These scripts use sbatch command to run config files.
- There is individual config file for each operations and for corresponding thread. E.g config_RR_1000_1thread.slurm this can be created using disk-config-creator.sh and disk-config-creator-latency.sh
- Each config file has #SBATCH argument and runs the executable.
- Dat file and output_disk_Latency.dat and output_disk_throughput.dat are used as input to the executable, Dat files has entry for each operation .
- After the execution output is written into output_disk_latency.dat and output_disk_latency.dat in formatted way.

- Steps to execute IOZONE:
- Create script to execute iozone named IOzone.sh
- Several parameters can be used to execute -s indicates file size, -r blocksize, -i for type or operations .

Network Benchmark:

1.Steps to execute Network benchmark:

- Use Makefile to compile MynetbenchTCP.c and MynetbenchUDP.c name output file mynetbenchtcp and netbenchudp.
- Once compiled there are three scripts run1.sh, run2.sh, run3.sh can be used to post jobs on slurm parallelly. Run each script individually one at a time to run TCP and UDP both operations.
- Run3.sh can be used to trigger jobs for measuring latency on 1B data.
- These scripts use sbatch command to run config files.
- There is individual config file for each operations and for corresponding thread. E.g config_TCP_1000_1thread.slurm this can be created using TCP-config-creator.sh and UDP-config-creator.sh.
- Each config file uses tcp-run.sh and udp-run.sh to execute the executables.
- Each config file has #SBATCH argument and runs the executable.
- Dat file and output_net_throughput.dat are used as input to the executable, Dat files has entry for each operation .
- After the execution output is written into output_net_throughput.dat in formatted way.

2.Steps to execute Iperf:

- Create iperf-run.sh and iperf-config.slurm
- Iperf-config.slurm uses iperf-run.sh to execute to iperf in server and client mode both.

```
#!/bin/bash

name=$(echo $1 | cut -d '-' -f1 -)
node1=$(echo $1 | cut -d '-' -f2 - | tr -d '[')
node2=$(echo $1 | cut -d '-' -f3 - | tr -d ']')
if [ "$(hostname)" == "$name-$node1" ]
then
    iperf3 -s
else
    sleep 20
    iperf3 -c $name-$node1 -P 2
fi
~
~
~
~
```

And

```
File Edit View Search Terminal Help
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks=2
#SBATCH --output=output-2.out
#SBATCH --wait-all-nodes=1
srun iperf-tcp-2run.sh $SLURM_JOB_NODELIST
~
~
~
~
~
~
~
```

This creates output file in same folder

File	Edit	View	Search	Terminal	Help					
[6]	1.00-2.00	sec	75.0	MBytes	629	Mbits/sec	42	257	KBytes	
[SUM]	1.00-2.00	sec	165	MBytes	1.39	Gbits/sec	46			
[4]	2.00-3.00	sec	84.2	MBytes	706	Mbits/sec	3	255	KBytes	
[6]	2.00-3.00	sec	85.1	MBytes	714	Mbits/sec	5	361	KBytes	
[SUM]	2.00-3.00	sec	169	MBytes	1.42	Gbits/sec	8			
[4]	3.00-4.00	sec	73.1	MBytes	613	Mbits/sec	20	252	KBytes	
[6]	3.00-4.00	sec	97.3	MBytes	816	Mbits/sec	3	438	KBytes	
[SUM]	3.00-4.00	sec	170	MBytes	1.43	Gbits/sec	23			
[4]	4.00-5.00	sec	80.4	MBytes	674	Mbits/sec	18	334	KBytes	
[6]	4.00-5.00	sec	90.3	MBytes	757	Mbits/sec	39	283	KBytes	
[SUM]	4.00-5.00	sec	171	MBytes	1.43	Gbits/sec	57			
[4]	5.00-6.00	sec	102	MBytes	853	Mbits/sec	7	298	KBytes	
[6]	5.00-6.00	sec	68.1	MBytes	572	Mbits/sec	4	315	KBytes	
[SUM]	5.00-6.00	sec	170	MBytes	1.42	Gbits/sec	11			
[4]	6.00-7.00	sec	61.7	MBytes	518	Mbits/sec	23	242	KBytes	
[6]	6.00-7.00	sec	110	MBytes	924	Mbits/sec	22	404	KBytes	
[SUM]	6.00-7.00	sec	172	MBytes	1.44	Gbits/sec	45			
[4]	7.00-8.00	sec	52.5	MBytes	441	Mbits/sec	12	281	KBytes	
[6]	7.00-8.00	sec	113	MBytes	950	Mbits/sec	44	324	KBytes	
[SUM]	7.00-8.00	sec	166	MBytes	1.39	Gbits/sec	56			
[4]	8.00-9.00	sec	78.4	MBytes	657	Mbits/sec	31	358	KBytes	
[6]	8.00-9.00	sec	86.4	MBytes	725	Mbits/sec	19	307	KBytes	
[SUM]	8.00-9.00	sec	165	MBytes	1.38	Gbits/sec	50			
[4]	9.00-10.00	sec	90.2	MBytes	757	Mbits/sec	37	304	KBytes	
[6]	9.00-10.00	sec	77.9	MBytes	653	Mbits/sec	12	211	KBytes	
[SUM]	9.00-10.00	sec	168	MBytes	1.41	Gbits/sec	49			
[ID]	Interval		Transfer		Bandwidth		Retr			
[4]	0.00-10.00	sec	799	MBytes	670	Mbits/sec	209		sender	
[4]	0.00-10.00	sec	797	MBytes	669	Mbits/sec			receiver	
[6]	0.00-10.00	sec	881	MBytes	739	Mbits/sec	237		sender	
[6]	0.00-10.00	sec	879	MBytes	737	Mbits/sec			receiver	
[SUM]	0.00-10.00	sec	1.64	GBytes	1.41	Gbits/sec	446		sender	
[SUM]	0.00-10.00	sec	1.64	GBytes	1.41	Gbits/sec			receiver	

iperf Done.