DSSC - Multicore and multinode

Grimaldi Marco Alberto - Presta Alberto

December 2018

Introduction

In this report we will describe three different exercises:

1. In the first one we will use Intel MPI benchmarks PingPong in order to estimate latency and bandwidth. We will do it both intranode and internode.

2. In the second one we will use the "Sustainable Memory Bandwidth Benchmark" (STREAM) in order to evaluate the Bandwidth in terms of MB/s.

3. We want to calculate nodeperf.c benchmark, which calculate a theoretical peak performance in terms of $GFlops$

MPI internode and intranode

MPI intranode

In this section we will use the MPI benchmark PingPong to measure bandwidth and the latency, both between cores in the same socket and in different sockets (always in the same node). How does PingPong work? It just sends a message from one core to another, and It estimates the latency and the bandwidth. We have written the following script in bash in figure 1.
First of all, we have loaded *impi* and then we have set a loop where we use PingPong for three different situations:

1. *core:0 core:1*: We want to calculate bandwidth and latency between two close cores.

2. *core:0 core:7*: We want to calculate bandwidth and latency between two cores which are not close,but which are in the same socket.

3. *core:0 core:13*: We want to calculate bandwidth and latency between two cores which are in different sockets in the same node.

```bash
#!/bin/bash

cd HPC/P1.2_seed/AssignmentsDSSC/A5-multicore-multinode/mpi

module load impi-trial/5.0.1.035

if [ -f results.txt ]; then rm results* ; fi

for j in {1..5..1}
    do
    sleep 1
    mpirun -np 2 hwloc-bind core:0 core:1 /u/shared/programs/x86_64/intel/impi_5.0.1/bin64/IMB-MPI1 PingPong >> results.txt
done
for j in {1..5..1}
    do
    sleep 1
    mpirun -np 2 hwloc-bind core:0 core:7 /u/shared/programs/x86_64/intel/impi_5.0.1/bin64/IMB-MPI1 PingPong >> results_07.txt
done
for j in {1..5..1}
    do
    sleep 1
    mpirun -np 2 hwloc-bind core:0 core:13 /u/shared/programs/x86_64/intel/impi_5.0.1/bin64/IMB-MPI1 PingPong >> results_13.txt
done

sed -i '/benchmark/d' res*
sed -i '/# /d' res*
```

Figure 1: Bash script for the PingPong Benchmark.

In order to do so, we have used *hwloc*, but what is hwloc precisely? It stands for Portable Hardware Locality and It is a software package which provides a portable abstraction of the hierarchical topology of modern architectures. In this specific exercise, we have used the command *hloc-bind* which Launches a command that is bound to specific processors and/or memory: in this case we force to do the PingPong benchmark between two specific cores. We have repeated the benchmark 5 times and we observe that results don't change too much and so we can take the mean of our results. Let's see the results in tables in figure 2.

With these kind of data, we observe that there is no such a huge difference between the first and the second case; indeed neither Bandwidth nor Latency changes a lot. Pretty different is the third case, where the latency grows up to 686.35 μsec and the bandwidth basically halves.

#bytes	#repetitio...	t[usec]	Mbytes/sec
0	1000	0.19	0.00
1	1000	0.19	5.03
2	1000	0.19	10.06
4	1000	0.19	20.08
8	1000	0.19	39.85
16	1000	0.22	70.95
32	1000	0.22	136.83
64	1000	0.25	247.58
128	1000	0.26	465.03
256	1000	0.28	871.86
512	1000	0.34	1440.73
1024	1000	0.40	2423.67
2048	1000	0.55	3544.79
4096	1000	0.83	4717.54
8192	1000	1.42	5499.83
16384	1000	2.58	6060.85
32768	1000	4.55	6865.10
65536	640	7.77	8041.81
131072	320	14.27	8758.43
262144	160	27.10	9225.09
524288	80	49.96	10007.29
1048576	40	95.74	10445.28
2097152	20	182.32	10969.44
4194304	10	352.20	11357.06

#bytes	#repetitions	t[usec]	Mbytes/sec
0	1000	0.20	0.00
1	1000	0.20	4.70
2	1000	0.20	9.32
4	1000	0.20	18.66
8	1000	0.20	38.93
16	1000	0.20	78.05
32	1000	0.25	121.79
64	1000	0.29	208.64
128	1000	0.28	435.93
256	1000	0.29	833.20
512	1000	0.35	1410.95
1024	1000	0.42	2338.57
2048	1000	0.60	3279.42
4096	1000	0.87	4469.18
8192	1000	1.49	5234.50
16384	1000	2.58	6056.37
32768	1000	4.54	6885.66
65536	640	7.69	8130.07
131072	320	14.07	8885.06
262144	160	26.88	9301.30
524288	80	49.51	10098.54
1048576	40	94.45	10587.67
2097152	20	180.77	11063.48
4194304	10	349.00	11461.41

#bytes	#repetitions	t[usec]	Mbytes/sec
0	1000	0.60	0.00
1	1000	0.66	1.45
2	1000	0.61	3.12
4	1000	0.62	6.20
8	1000	0.61	12.49
16	1000	0.62	24.51
32	1000	0.62	49.42
64	1000	0.68	89.89
128	1000	0.66	184.80
256	1000	0.66	372.70
512	1000	0.87	561.87
1024	1000	1.01	962.18
2048	1000	1.32	1474.57
4096	1000	2.03	1927.08
8192	1000	3.36	2324.80
16384	1000	5.64	2769.15
32768	1000	10.95	2854.54
65536	640	12.58	4968.97
131072	320	22.63	5523.64
262144	160	40.68	6145.73
524288	80	83.28	6003.76
1048576	40	169.20	5910.18
2097152	20	341.68	5853.47
4194304	10	686.35	5827.95

Figure 2: In these three tables we can observe data concerned the bandwidth (in $Mbytes/s$) and the latency(in μsec) using the PingPong Benchmark: In the left table, we have data in the case of "close cores" (case *core:0*, *core:1*). In the central table there is data related to the second case we have considered where cores are not close, but there are in the same socket (case *core:0*, *core:7*).Finally, In the right table there is data concerned the case where cores are in two different socket in the same node.

MPI Internode

The last thing that we want to do is to estimate Bandwidth and Latency between cores in different nodes: in order to do so, we have requested two different nodes in "interactive mode":

$$qsub \ -l \ nodes = 2 : ppn = 2, walltime = 2 : 00 : 00 \ -I$$

$$module\ load\ intel/14.0 \ \ impi - trial/5.0.1.035$$

And then we run the Benchmark "PingPong" with this command[1]:

```
apresta@cn01-08 ~]$ mpirun -np 2 -ppn 1 -hosts cn01-08,cn01-12 /u/shared/programs/x86_64/intel/impi_5.0.1/bin64/IMB-MPI1 PingPong
```

In the following table we have the results:

[1] we notice that we have two different nodes, but in the same rack; if we had been nodes in different racket, results would have been much worse!

#bytes	#repetitions	t[usec]	Mbytes/sec
0	1000	4.02	0.00
1	1000	4.07	0.23
2	1000	4.06	0.47
4	1000	4.04	0.94
8	1000	4.06	1.88
16	1000	4.08	3.74
32	1000	4.17	7.33
64	1000	4.16	14.66
128	1000	4.26	28.65
256	1000	4.37	55.82
512	1000	4.61	105.85
1024	1000	5.13	190.53
2048	1000	6.20	315.07
4096	1000	7.42	526.17
8192	1000	9.70	805.82
16384	1000	19.42	804.52
32768	1000	32.41	964.23
65536	640	56.71	1102.06
131072	320	112.00	1116.03
262144	160	115.10	2172.08
524288	80	197.12	2536.53
1048576	40	356.31	2806.52
2097152	20	675.95	2958.79
4194304	10	1349.31	2964.49

Finally, We can compare our results with grahps shown in figure 3 and figure 4:

1. In the first, Bandwidth is shown in the different situations we have studied.
2. In the second, we evaluate the latency in the different situations we have studied.

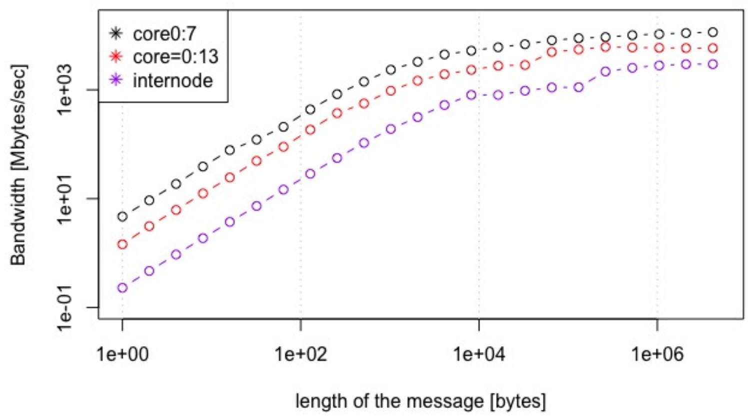

Figure 3: This figure shows how the bandwidth changes if we choose cores at different distances. the values of the abscissas and the ordinate are in a logarithmic scale.

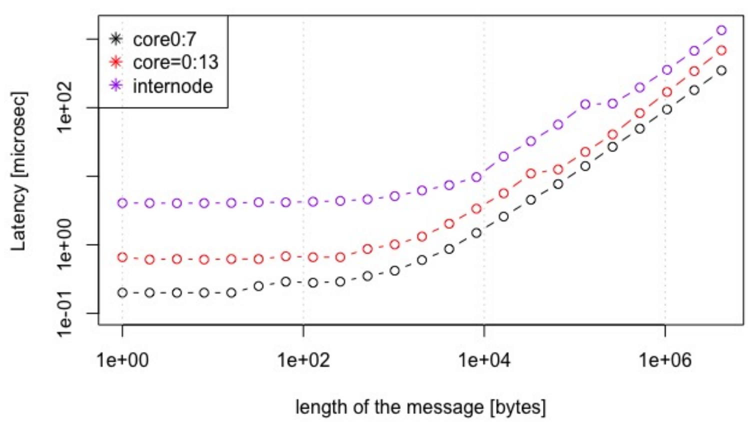

Figure 4: this figure shows how the latency changes when we choose cores at different distances. The values of the abscissas and the ordinate are in a logarithmic scale.

Stream

The STREAM benchmark is a simple synthetic benchmark program that measures sustainable memory bandwidth (in MB/s) and the corresponding computation rate for simple vector kernels, separately timed:

1. Copy : $A[i] = B[i]$.
2. Scale : $A[i] = s \cdot B[i]$.
3. Add : $C[i] = A[i] + B[i]$.
4. Triad : $A[i] = B[i] + s \cdot C[i]$.

Our main purpose is to evaluate Bandwidth, using also *Numactl* in order to force the position of the core and the memory; indeed Numactl runs processes with a specific NUMA scheduling or memory placement policy. Since we already have the code, after having compiled it with the "make" command, we have written 4 different script in order to evaluate 4 different situation:

1. With the command $numactl \ --cpunodebind\ 0 \quad --membind\ 0$ we force to process and use memory of the same socket (we also swap the sockets). So in this case we want to compute bandwidths for one single cores reading from the memory to their own socket.
2. With the command $numactl \ --cpunodebind\ 0 \quad --membind\ 1$ where threads process in one socket, with memory allocated in the other one (also here we exchanged socket roles).So in this case we want to compute bandwidth for one single cores reading from the memory to distant socket.

In figure 5 we can compare the two different situation (We have used the Bandwidth related to the Triad kernels) .

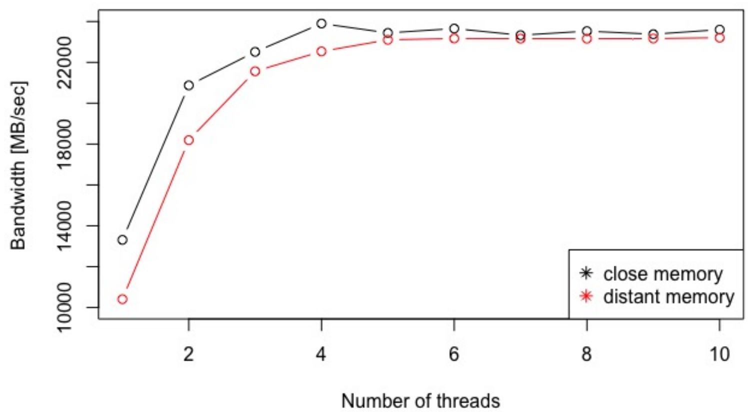

Figure 5: As we can see, while the difference between distant memory and closed memory is quite large with few threads: this difference decreases with the increase of the number of these one.

Nodeperf

Nodeperf computes matrix-matrix multiplication in order to calculate the peak performance in terms of Gflops. In order to make this benchmark work we have replaced *mkl_malloc* with normal malloc in the code and we have written the following script bash:

```
#!/bin/bash
#PBS -l nodes=1:ppn=20
#PBS -l walltime=01:00:00

cd $PBS_O_WORKDIR

module load impi-trial
module load intel/14.0
module load mkl

mpiicc -O3 -xHost -fopenmp -mkl nodeperf.c -o nodeperf.x

export OPM_NUM_THREADS=20
export OMP_PLACES=cores

./nodeperf.x >> res.txt
```

Using Intel compiler, we have the result shown in figure 6:

```
Multi-threaded MPI detected

The time/date of the run...  at Mon Dec  3 14:22:37 2018

This driver was compiled with:
        -DITER=4 -DLINUX -DNOACCUR -DPREC=double
Malloc done.  Used 1846080096 bytes
(0 of 1): NN lda=15000 ldb= 192 ldc=15000 0 0 0 446141.301 cn06-12
```

Figure 6: Obtained results running Intel compiler

If we instead use *gcc compiler* (mpicc) we deal with much more worse results (figure 7):

```
Multi-threaded MPI detected

The time/date of the run...  at Sat Dec 15 13:42:57 2018

This driver was compiled with:
        -DITER=4 -DLINUX -DNOACCUR -DPREC=double
Malloc done.  Used 1846080096 bytes
(0 of 1): NN lda=15000 ldb= 192 ldc=15000 0 0 0 26995.435 cn02-37
```

Figure 7: Obtained results with gcc compiler.