Multicore Multinode - A5

Marco Franzon

December, 2018

Abstract

In this exercise, the aim is to investigate and give a measure of the latency and bandwidth between cores in the same or in different sockets or NUMA nodes of the Ulysses cluster.

1 Introduction - Ping Pong

Ping Pong is an Intel benchmark used to measure latency and bandwidth of a message sent between two cores. The message size keeps increasing, because initial small messages are used to estimate latency, while bigger messages are used to estimate bandwidth. We performed three tests, to measure respectively latency and bandwidth for inter-socket, intra-socket and intra-node communications between cores of Ulisse cluster.

2 MPI

Once loaded the module *impi-trial/5.0.1.035* and *hwloc* we ran the code using *mpirun*. Using *hwloc* we have ensured that only two cores from the same socket were used for the first run and two from two different sockets in the second. It should be noted that the results obtained for small amount of data passed are significant for the latency while the result for large data transfer are significant for the bandwidth. We loaded *impi* and *hwloc* and launched the benchmark with the following command:

```
mpirun -np 2 path-to-executable/IMB-MPI1 PingPong -iter 10000 PingPong
```

The output of the MPI benchmark is shown in Figure 1. As it can be seen the time t is constant for packets of size up to and including 16/32 bits, the average of these can be used as an estimate of the latency of the channel. On the other side, the bandwidth keeps increasing as the packet size increases, but reaches a plateau for big packet sizes. In order to have an estimate of the bandwidth, the average of the last three entries is taken. The following tables are the output of the ping pong program. First one regards two cores in the same socket. Second one is the output of two cores from different sockets.

```
#bytes #repetitions      t[usec]   Mbytes/sec
      0      1000000         0.66         0.00
      1      1000000         0.74         1.30
      2      1000000         0.73         2.60
      4      1000000         0.74         5.19
      8      1000000         0.73        10.39
     16      1000000         0.73        20.81
     32      1000000         0.74        41.00
     64       655360         0.78       77.99
    128       327680         0.79       155.04
    256       163840         0.86       282.39
    512        81920         1.03       473.08
   1024        40960         1.28       764.15
   2048        20480         1.61      1210.23
   4096        10240         2.34      1668.02
   8192         5120         3.83      2042.23
  16384         2560         6.89      2267.25
  32768         1280        12.84      2433.76
  65536          640        15.57      4013.21
 131072          320        28.13      4444.24
 262144          160        50.26      4974.56
 524288           80       106.58      4691.48
1048576           40       219.41      4557.60
2097152           20       445.95      4484.81
4194304           10       908.65      4402.14
```

```
   #bytes #repetitions      t[usec]   Mbytes/sec
      0       903011         5.55         0.00
      1       884632         5.66         0.17
      2       884632         5.65         0.34
      4       884632         5.65         0.67
      8       884408         5.65         1.35
     16       884408         5.64         2.71
     32       879273         5.67         5.39
     64       655360         5.69        10.73
    128       327680         5.70        21.43
    256       163840         5.76        42.41
    512        81920         5.75        84.90
   1024        40960         5.81       168.19
   2048        20480         5.91       330.59
   4096        10240         6.15       635.34
   8192         5120         6.54      1195.29
  16384         2560         7.38      2116.41
  32768         1280         9.25      3377.40
```

65536	640	901.39	69.34
131072	320	914.72	136.65
262144	160	922.85	270.90
524288	80	2770.42	180.48
1048576	40	6358.53	157.27
2097152	20	14007.58	142.78
4194304	10	28846.80	138.66

It should be noted that the results obtained for small amount of data passed are significant for the latency while the result for large data transfer are significant for the bandwidth. It's clear from the tables that the bandwith is bigger for cores from the same socket while the latency is smaller.

3 Stream

Stream stand for Sustainable Memory Bandwidth Benchmark and uses four vector kernels to measure sustainable memory bandwidth (in MB/s). We set the variable OMP NUM THREADS=1 in order to perform this test on a single core, with the following results:

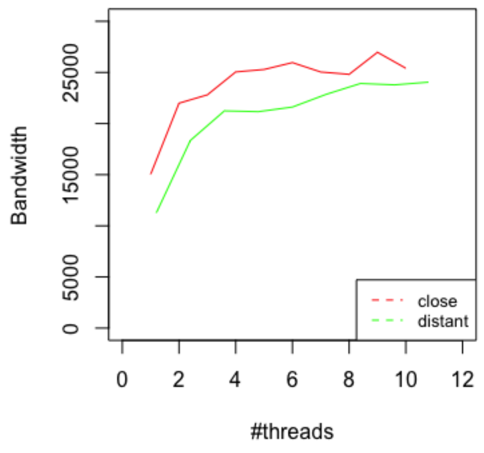

Figure 1: bandwidth for different numbers of cores, using both the memory of the same socket of the cores and the one of the other socket.

In the graph we can see that the cost on bandwidth of using a distant memory region is relevant in any case and should be avoided for better performance.

4 Nodeperf

In this test we run a benchmark that measure peak performance through matrix-matrix multiplication. In particular, we want to investigate the number of FLOPS attainable using the `nodeperf` benchmark tool. All the 20 cores are used in order to get an estimate of the performance. For this assignment we were supposed to compile the program nodeperf.c using the intel compiler but the suggested command didn't work, at first we changed the flag -qopenmp in -fopenmp, and the program compiled but while running something wrong in memory happened everytime and the cluster gave segmentation fault, in order to solve the problem we changed also the code commenting the MKL malloc line and decommenting the line with the normal C malloc. Using 20 cores the result obtained is 448.35 GFlops, that reaching almost perfectly the peak performance.