A5: Multicore-Multinode

Francesca Cairoli

Introduction

The architecture on modern computer is quite complex, we have multiple nodes which contain multiple sockets, each of which contains several cores. The communication between these components introduces the concepts of latency, i.e., the time required to open a communication channel, and bandwidth, i.e., the number of Mbytes per seconds that travel through the channel. Message Passing Interface (MPI) is a message passing standard which works on a wide variety of parallel computing architectures.

Theoretical Peak Performance (TPP)

The theoretical peak performance indicates the maximum possible number of floating point operations per second (FLOPS) which could be achieved inside a system. From a formal point of view it is defined as:

$$\text{Flops} = (\sharp \text{ sockets}) \times (\sharp \text{ cores}) \times (\text{Flops / cycle}) \times \text{ clock rate} = 2 \times 10 \times 8 \times (2.8 \times 10^9) = 448 \text{ GFlops}.$$

The values inserted above are relative to Ulysses' nodes, used for executing all the following benchmarks.

PingPong: communication performance

PingPong is a benchmark created by Intel to measure both latency and throughput of a single message sent between two processes. We run this program to measure latency and bandwidth of Ulysses's nodes. The very basic idea is to let processors exchange messages one to another. The two processors may be on different cores but on the same socket, where we expect the lowest latency. Then, we test the technology that connect sockets by computing latency between processors on different sockets but on the same node (intranode connection). Finally, we test the connection between nodes.

Latency and bandwidth

Latency indicates the amount of delay (or time) it takes to send information from one point to the next. In order to measure it, we consider the time needed

to ping-pong the smallest messages.

On the other hand, bandwidth is the maximum rate of data transferable across a given path. In order to measure it, we look at the largest messages, i.e. when the bandwidth tends to flat.

Tables below show the results of three consecutive runs in order to obtain more robust and precise estimates.

Intranode and internode

As starter, we would like to estimate the latency and bandwidth among cores that belong to the same socket and among cores that are located on different sockets. We start by considering a Ulysses node with 20 processors, where cores with id from 0 to 9 belong to socket 0 and cores with id from 10 to 19 belong to socket 1. First of all, in order to estimate latency among cores belonging to the same socket, we run the PingPong benchmark between core 0 and core 7, which belong to socket 0. Second of all, in order to evaluate latency and bandwidth among cores belonging to different sockets, we we run the PingPong benchmark between core 0, belonging to socket 0, and core 13, belonging to socket 1.

```
# cores on same socket
mpirun -np 2 hwloc-bind core:0 core:7 /u/shared/programs/x86_64/
    intel/impi_5.0.1/bin64/IMB-MPI1 -iter 1000000 PingPong

# cores on different socket
mpirun -np 2 hwloc-bind core:0 core:13 /u/shared/programs/x86_64/
    intel/impi_5.0.1/bin64/IMB-MPI1 -iter 1000000 PingPong
```

Finally, in order to estimate the latency and bandwidth between cores that are on different nodes we run the PingPong benchmark with the following command:

```
#Internode execution
# -npernode was deprecated in favor of --map-by ppr : n : node
mpirun -np 2 --map-by ppr:1:node /u/shared/programs/x86_64/intel/
    impi_5.0.1/bin64/IMB-MPI1 -iter 1000000 PingPong
```

All the estimates where executed on on interactive node.

0.0.1 Intranode results

Cores belonging to the same socket

- Latency:

bytes	run 1 [μs]	run 2 [μs]	run 3 [μs]
0	0.21	0.19	0.19
1	0.20	0.19	0.22
2	0.20	0.19	0.22
4	0.21	0.20	0.20
8	0.20	0.19	0.21
16	0.20	0.19	0.21

- Bandwidth:

bytes	run 1 [MB/s]	run 2 [MB/s]	run 3 [MB/s]
524288	10057.68	10088.22	10080.64
1048576	10583.33	10587.67	10579.32
2097152	11080.29	11114.05	11086.14
4194304	11412.30	11422.01	11058.74

Cores belonging to different sockets

- Latency:

bytes	run 1 [μs]	run 2 [μs]	run 3 [μs]
0	0.61	0.59	0.56
1	0.66	0.65	0.61
2	0.66	0.64	0.60
4	0.64	0.64	0.61
8	0.65	0.63	0.60
16	0.66	0.63	0.61

- Bandwidth:

bytes	run 1 [MB/s]	run 2 [MB/s]	run 3 [MB/s]
524288	5853.06	6197.26	6080.90
1048576	5738.55	6036.38	5979.05
2097152	5709.35	6146.74	5925.94
4194304	5658.90	6178.09	5887.16

0.0.2 Internode results

Cores belonging to different nodes

- Latency:

bytes	run 1 [μs]	run 2 [μs]	run 3 [μs]
0	0.58	0.66	0.61
1	0.64	0.68	0.70
2	0.64	0.69	0.63
4	0.64	0.64	0.59
8	0.64	0.63	0.60
16	0.63	0.64	0.58

- Bandwidth:

bytes	run 1 [MB/s]	run 2 [MB/s]	run 3 [MB/s]
524288	5798.84	5921.13	5698.40
1048576	5604.17	5775.39	5519.91
2097152	5517.28	5676.13	5389.06
4194304	5469.79	5632.21	5330.16

When the cores belong to the same socket the average latency is around $0.20\mu s$ and the average bandwidth is around 10.72 Gbps. On the other hand when the intersocket scenarios shows a significant increase in latency and a significant decrease in bandwidth. In particular, when the cores belong to different sockets inside the same node we have an average latency of $0.625\mu s$, when they belong to different nodes the average latency is $0.634\mu s$. The average value of bandwidth is around 6 Gbps for the intersocket scenario and around 5.6 Gbps in the internode scenario, nearly a half of the intrasocket one.

As expected, since the distance between nodes is the longest considered so far, values of bandwidth between cores on different nodes are smaller than the ones computed considering intranode cores and viceversa for the latency. In conclusion, we can state that the increase in distance between components translates in a degradation of communication performances including both latency and bandwidth.

STREAM: memory bandwidth

Since the CPUs are increasing their speed much more quickly than the memory systems, it is likely that a program's performance will be limited by its memory bandwidth rather than the performance of the CPU. Therefore, it is important to measure the performance of the memory system of a machine. In particular, the Stream benchmark measures sustainable memory bandwidth in MB/s.

Recall that, under a NUMA (Non-Uniform Memory Access) architecture, a processor can access its own local memory faster than non-local memory. Since Ulysses present a NUMA architecture we are able to compare the bandwidth of a core reading from memory associated to its own socket (close memory) against a core reading from memory associated to another socket (distant memory).

Hence, the measurements were conducted under two different scenarios: at first both the process and the memory were bind to socket 1 and then we bind the process to socket 1 and memory to socket 0.

0.0.3 Results

We compile the benchmark `stream.c` using the provided Makefile. We were provided with both a serial and MPI implementation. Since we want to vary the number of threads we are going to use the latter (executable file `stream_omp`). Then, we loop over ten threads, using numactl to bind the process to socket and to memory.

```
# same socket memory
for i in `seq 1 10`; do
echo "$i threads" >> samesocket_res.txt
 OMP_NUM_THREADS=$i numactl --cpunodebind 1 --membind 1 ./
    stream_omp.x | grep "Triad:" | awk '{print $2}' >>
    samesocket_res.txt
 done

# different socket memory
```

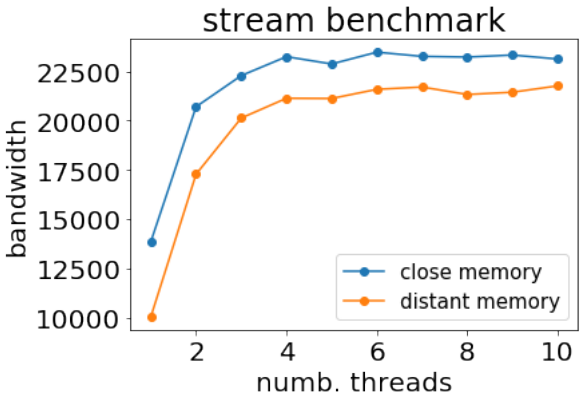

Figure 1: Comparison of memory bandwidth between close and distant memory

```
for i in `seq 1 10`; do
echo "$i threads" >> diffsocket_res.txt
OMP_NUM_THREADS=$i numactl --cpunodebind 1 --membind 0 ./stream_omp
    .x | grep "Triad:" | awk '{print $2}' >> diffsocket_res.txt
done
```

Figure 1 shows the numerical results. Recall that each point indicates the best bandwidth obtained after 50 iterations given a number of threads. We observe, as expected, that local access is faster that non-local access. The gap, however, seems to decrease as the number of threads increases.

Nodeperf

Nodeperf is a program that aims at evaluating the performance of the whole node using the matrix-matrix multiplication. This routine is also the core of the HPL test, that we are going to discuss in next chapter. We compare the performances of the Intel version against the gnu version.

```
# compilation with Intel compiler
mpiicc -O2 -xHost -fopenmp -mkl nodeperf.c -o nodeperf_intel.x

# compilation with gnu compiler
mpicc -fopenmp -o3 nodeperf.c -m64 -I${MKLROOT}/include -o
    nodeperf_gnu.x -L${MKLROOT}/lib/intel64 -Wl,--no-as-needed -
    lmkl_intel_lp64 -lmkl_sequential -lmkl_core -lpthread -lm -ldl
```

Using the Intel version, running in the interactive mode on `cn07-25`, we reached a peak performance of 444.61236 GFlops, corresponding to the 99.24% of the TPP. On the other hand, on the very same node, we obtained a peak performance of 26.989855 GFlops, corresponding only to the 6.0245% of the TPP. The difference in the performance is extremely large. This is due to the fact that nodeperf is highly optimized in order to leverage tricks provided by the Intel compiler only.