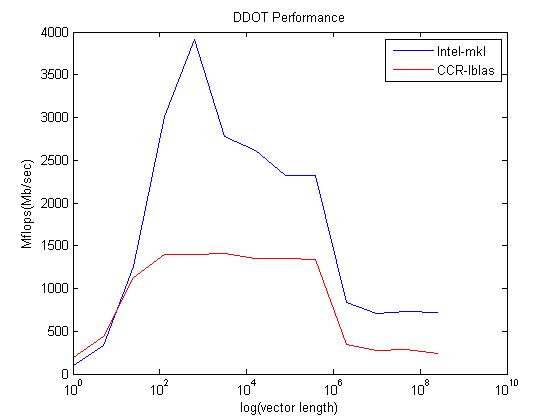
# Question 1

The graph for the performance of lblas and intel-mkl for DDOT\_ is given in the figure below:



The graph clearly shows a superior performance of Intel’s MKL library, which is what should be expected. The Intel MKL library also shows peaks and 2 kind of steps which shows the memory usage. For example the first peak performance is L1 cache, 2nd small bump is the L2, followed by L3 and finally into the main memory. The CCR lblas also shows peak and that is also due to L1 cache. If the code is made to run for a longer period of time, it will be a plateau which is shown around 10^8 vector size.

The code is attached in the appendix. For compiling the code in C, following specific commands were used for –lblas and intel mkl

1. **–lblas**

$gcc –o vector vector.c –lblas

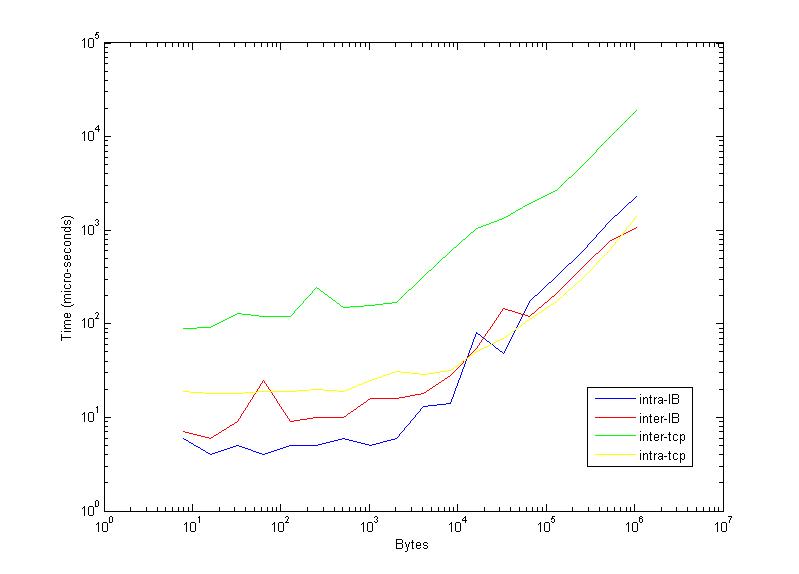
1. **–intel mkl**

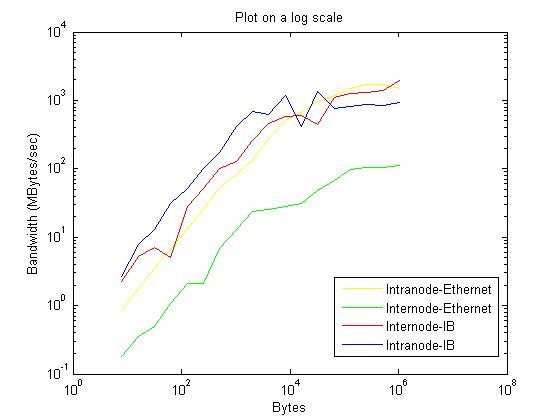
$gcc –o vector vector.c –L$MKLROOT/lin/intel64 –lmkl\_intel\_lp64 –lmkl\_sequential –lmkl\_core –lpthread –lm

Results were printed out in the terminal and exported to MATLAB to plot the final figure

# Question 2

The code for this question was downloaded from the internet whose reference is given along with the code in appendix. The results obtained were as follows:





Quite clearly, intra node performance is much better than inter node, which is something that should be expected. Also the Ethernet internode performance is the worst as seen in the figure.

A sample slurm file :

#!/bin/sh

##SBATCH --partition=debug

#SBATCH --time=00:15:00

#SBATCH --nodes=2

#SBATCH --ntasks-per-node=1

##SBATCH --mem=24000

# Memory per node specification is in MB. It is optional.

# The default limit is 3GB per core.

#SBATCH --job-name="ping\_pong"

#SBATCH --output=ping\_pong\_internode\_tcp.out

#SBATCH --mail-user=npaliwal@buffalo.edu

#SBATCH --mail-type=END

##SBATCH --requeue

#Specifies that the job will be requeued after a node failure.

#The default is that the job will not be requeued.

echo "SLURM\_JOBID="$SLURM\_JOBID

echo "SLURM\_JOB\_NODELIST"=$SLURM\_JOB\_NODELIST

echo "SLURM\_NNODES"=$SLURM\_NNODES

echo "SLURMTMPDIR="$SLURMTMPDIR

cd $SLURM\_SUBMIT\_DIR

echo "working directory = "$SLURM\_SUBMIT\_DIR

module load intel/13.0

module load intel-mpi/4.1.0

module list

ulimit -s unlimited

#

export I\_MPI\_PMI\_LIBRARY=/usr/lib64/libpmi.so

export I\_MPI\_DEBUG=4

export I\_MPI\_FABRICS="tcp"

srun ./mpi\_pong

echo "All Done!"

The line export I\_MPI\_FABRICS=”tcp” takes the code into Ethernet, and if this is commented out, the signaling is done via infiniband. The specific nodes for all 4 cases are:

1. Internode infiniband: d07n05s02, d07n06s02
2. Intranode infiniband: d07n05s02
3. Internode TCP: d07n05s02, d07n06,s02
4. Intranode TCP: d07n05s02

By looking at these nodes I am assuming that these nodes are specific for debug, and the node information is given just to make sure that intranode gives out only one node and internode gives out 2 nodes.

|  |  |
| --- | --- |
| Type | Minimum Latency (µsec) |
| Internode infiniband | 3.099 |
| Intranode infiniband | 0.953 |
| Internode TCP | 25.987 |
| Intranode TCP | 5.483 |

Clearly intranode infiniband latency is the lowest, which makes sense.

# Appendix 1

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <sys/time.h>

#include <math.h>

int main(int argc, char\*\* argv)

{

int n,i,j;

double vec,\*a,\*b,dot\_pdt;

struct timeval t0,t1;

for(vec=1;vec<1000000;vec\*5){

n=(10^7)/(2\*vec+1);

a=malloc(sizeof(double)\*(vec));

b=malloc(sizeof(double)\*vec);

for(i=1;i<vec;i++){

a[i]=i+1;

b[i]=i+1;

}

gettimeofday(&t0,0);

for(j=1;j<n;j++){

dot\_pdt=ddot\_(vec,a,1,b,1);

}

}

gettimeofday(&t1,0);

long elapsed = (t1.tv\_sec-t0.tv\_sec)\*1000000 + t1.tv\_usec-t0.tv\_usec;

double t=(elapsed/((float)1000000))/((float)n);

double Mflops=(2\*vec+1)/t;

printf("%d\n",Mflops);

printf("%i\n",vec);

return 0;

}

# Appendix 2

The reference link to the code is:

<http://www.scl.ameslab.gov/Projects/mpi_introduction/para_pingpong.html>

/\* pong.c Generic Benchmark code

\* Dave Turner - Ames Lab - July of 1994+++

\*

\* Most Unix timers can't be trusted for very short times, so take this

\* into account when looking at the results. This code also only times

\* a single message passing event for each size, so the results may vary

\* between runs. For more accurate measurements, grab NetPIPE from

\* http://www.scl.ameslab.gov/ .

\*/

#include "mpi.h"

#include <stdio.h>

#include <stdlib.h>

void

main (int argc, char \*\*argv)

{

int myproc, size, other\_proc, nprocs, i, last;

double t0, t1, time;

double \*a, \*b;

double max\_rate = 0.0, min\_latency = 10e6;

MPI\_Request request, request\_a, request\_b;

MPI\_Status status;

#if defined (\_CRAYT3E)

a = (double \*) shmalloc (132000 \* sizeof (double));

b = (double \*) shmalloc (132000 \* sizeof (double));

#else

a = (double \*) malloc (132000 \* sizeof (double));

b = (double \*) malloc (132000 \* sizeof (double));

#endif

for (i = 0; i < 132000; i++) {

a[i] = (double) i;

b[i] = 0.0;

}

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &nprocs);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myproc);

/\* if (nprocs != 2) exit (1);\*/

other\_proc = (myproc + 1) % 2;

printf("Hello from %d of %d\n", myproc, nprocs);

MPI\_Barrier(MPI\_COMM\_WORLD);

/\* Timer accuracy test \*/

t0 = MPI\_Wtime();

t1 = MPI\_Wtime();

while (t1 == t0) t1 = MPI\_Wtime();

if (myproc == 0)

printf("Timer accuracy of ~%f usecs\n\n", (t1 - t0) \* 1000000);

/\* Communications between nodes

\* - Blocking sends and recvs

\* - No guarantee of prepost, so might pass through comm buffer

\*/

for (size = 8; size <= 1048576; size \*= 2) {

for (i = 0; i < size / 8; i++) {

a[i] = (double) i;

b[i] = 0.0;

}

last = size / 8 - 1;

MPI\_Barrier(MPI\_COMM\_WORLD);

t0 = MPI\_Wtime();

if (myproc == 0) {

MPI\_Send(a, size/8, MPI\_DOUBLE, other\_proc, 0, MPI\_COMM\_WORLD);

MPI\_Recv(b, size/8, MPI\_DOUBLE, other\_proc, 0, MPI\_COMM\_WORLD, &status);

} else {

MPI\_Recv(b, size/8, MPI\_DOUBLE, other\_proc, 0, MPI\_COMM\_WORLD, &status);

b[0] += 1.0;

if (last != 0)

b[last] += 1.0;

MPI\_Send(b, size/8, MPI\_DOUBLE, other\_proc, 0, MPI\_COMM\_WORLD);

}

t1 = MPI\_Wtime();

time = 1.e6 \* (t1 - t0);

MPI\_Barrier(MPI\_COMM\_WORLD);

if ((b[0] != 1.0 || b[last] != last + 1)) {

printf("ERROR - b[0] = %f b[%d] = %f\n", b[0], last, b[last]);

exit (1);

}

for (i = 1; i < last - 1; i++)

if (b[i] != (double) i)

printf("ERROR - b[%d] = %f\n", i, b[i]);

if (myproc == 0 && time > 0.000001) {

printf(" %7d bytes took %9.0f usec (%8.3f MB/sec)\n",

size, time, 2.0 \* size / time);

if (2 \* size / time > max\_rate) max\_rate = 2 \* size / time;

if (time / 2 < min\_latency) min\_latency = time / 2;

} else if (myproc == 0) {

printf(" %7d bytes took less than the timer accuracy\n", size);

}

}

/\* Async communications

\* - Prepost receives to guarantee bypassing the comm buffer

\*/

MPI\_Barrier(MPI\_COMM\_WORLD);

if (myproc == 0) printf("\n Asynchronous ping-pong\n\n");

for (size = 8; size <= 1048576; size \*= 2) {

for (i = 0; i < size / 8; i++) {

a[i] = (double) i;

b[i] = 0.0;

}

last = size / 8 - 1;

MPI\_Irecv(b, size/8, MPI\_DOUBLE, other\_proc, 0, MPI\_COMM\_WORLD, &request);

MPI\_Barrier(MPI\_COMM\_WORLD);

t0 = MPI\_Wtime();

if (myproc == 0) {

MPI\_Send(a, size/8, MPI\_DOUBLE, other\_proc, 0, MPI\_COMM\_WORLD);

MPI\_Wait(&request, &status);

} else {

MPI\_Wait(&request, &status);

b[0] += 1.0;

if (last != 0)

b[last] += 1.0;

MPI\_Send(b, size/8, MPI\_DOUBLE, other\_proc, 0, MPI\_COMM\_WORLD);

}

t1 = MPI\_Wtime();

time = 1.e6 \* (t1 - t0);

MPI\_Barrier(MPI\_COMM\_WORLD);

if ((b[0] != 1.0 || b[last] != last + 1))

printf("ERROR - b[0] = %f b[%d] = %f\n", b[0], last, b[last]);

for (i = 1; i < last - 1; i++)

if (b[i] != (double) i)

printf("ERROR - b[%d] = %f\n", i, b[i]);

if (myproc == 0 && time > 0.000001) {

printf(" %7d bytes took %9.0f usec (%8.3f MB/sec)\n",

size, time, 2.0 \* size / time);

if (2 \* size / time > max\_rate) max\_rate = 2 \* size / time;

if (time / 2 < min\_latency) min\_latency = time / 2;

} else if (myproc == 0) {

printf(" %7d bytes took less than the timer accuracy\n", size);

}

}

/\* Bidirectional communications

\* - Prepost receives to guarantee bypassing the comm buffer

\*/

MPI\_Barrier(MPI\_COMM\_WORLD);

if (myproc == 0) printf("\n Bi-directional asynchronous ping-pong\n\n");

for (size = 8; size <= 1048576; size \*= 2) {

for (i = 0; i < size / 8; i++) {

a[i] = (double) i;

b[i] = 0.0;

}

last = size / 8 - 1;

MPI\_Irecv(b, size/8, MPI\_DOUBLE, other\_proc, 0, MPI\_COMM\_WORLD, &request\_b);

MPI\_Irecv(a, size/8, MPI\_DOUBLE, other\_proc, 0, MPI\_COMM\_WORLD, &request\_a);

MPI\_Barrier(MPI\_COMM\_WORLD);

t0 = MPI\_Wtime();

MPI\_Send(a, size/8, MPI\_DOUBLE, other\_proc, 0, MPI\_COMM\_WORLD);

MPI\_Wait(&request\_b, &status);

b[0] += 1.0;

if (last != 0)

b[last] += 1.0;

MPI\_Send(b, size/8, MPI\_DOUBLE, other\_proc, 0, MPI\_COMM\_WORLD);

MPI\_Wait(&request\_a, &status);

t1 = MPI\_Wtime();

time = 1.e6 \* (t1 - t0);

MPI\_Barrier(MPI\_COMM\_WORLD);

if ((a[0] != 1.0 || a[last] != last + 1))

printf("ERROR - a[0] = %f a[%d] = %f\n", a[0], last, a[last]);

for (i = 1; i < last - 1; i++)

if (a[i] != (double) i)

printf("ERROR - a[%d] = %f\n", i, a[i]);

if (myproc == 0 && time > 0.000001) {

printf(" %7d bytes took %9.0f usec (%8.3f MB/sec)\n",

size, time, 2.0 \* size / time);

if (2 \* size / time > max\_rate) max\_rate = 2 \* size / time;

if (time / 2 < min\_latency) min\_latency = time / 2;

} else if (myproc == 0) {

printf(" %7d bytes took less than the timer accuracy\n", size);

}

}

if (myproc == 0)

printf("\n Max rate = %f MB/sec Min latency = %f usec\n",

max\_rate, min\_latency);

MPI\_Finalize();

}