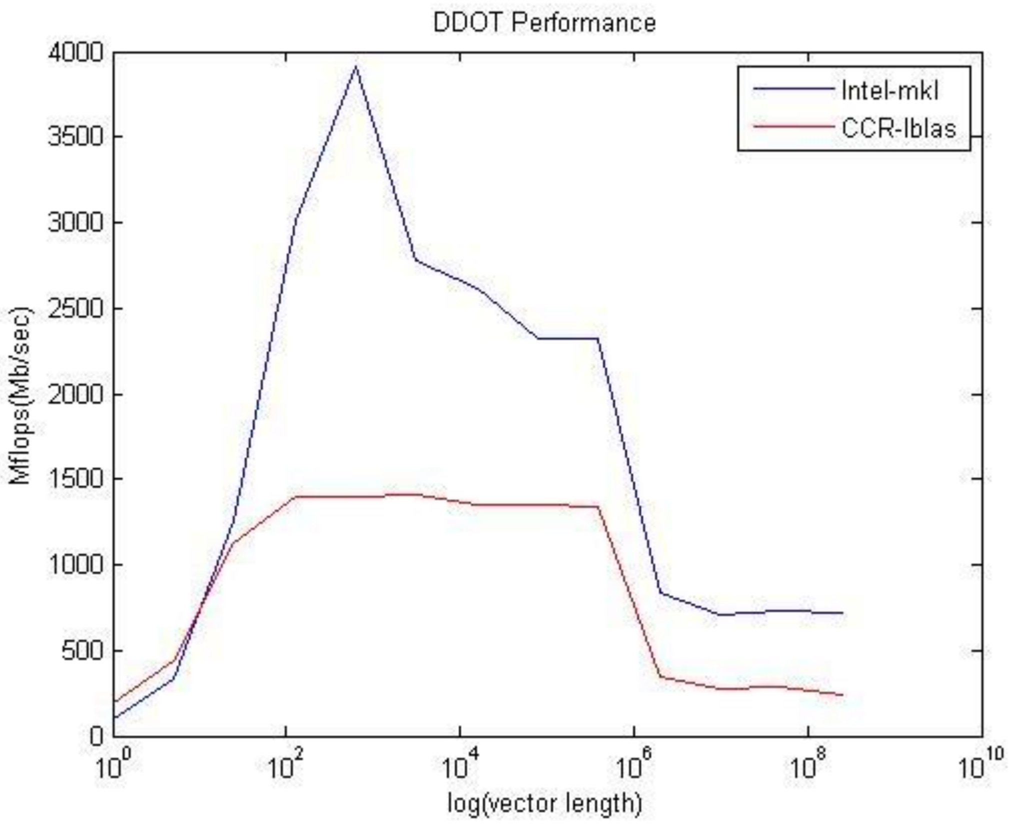
Question 1

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
```

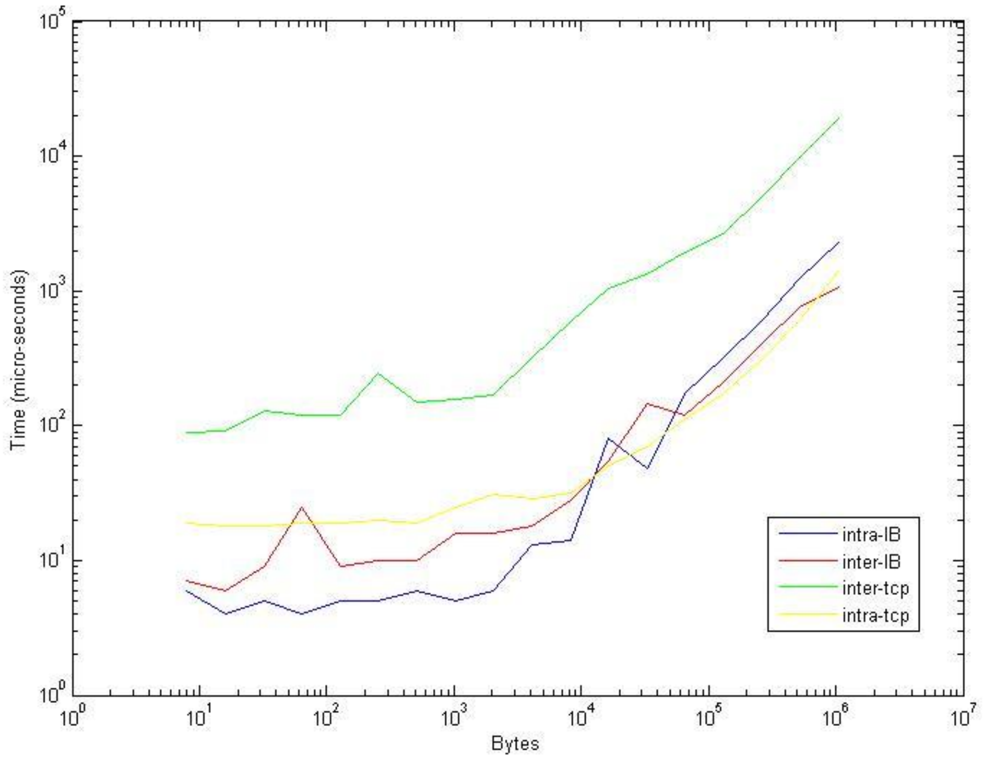
2. **–intel mkl**

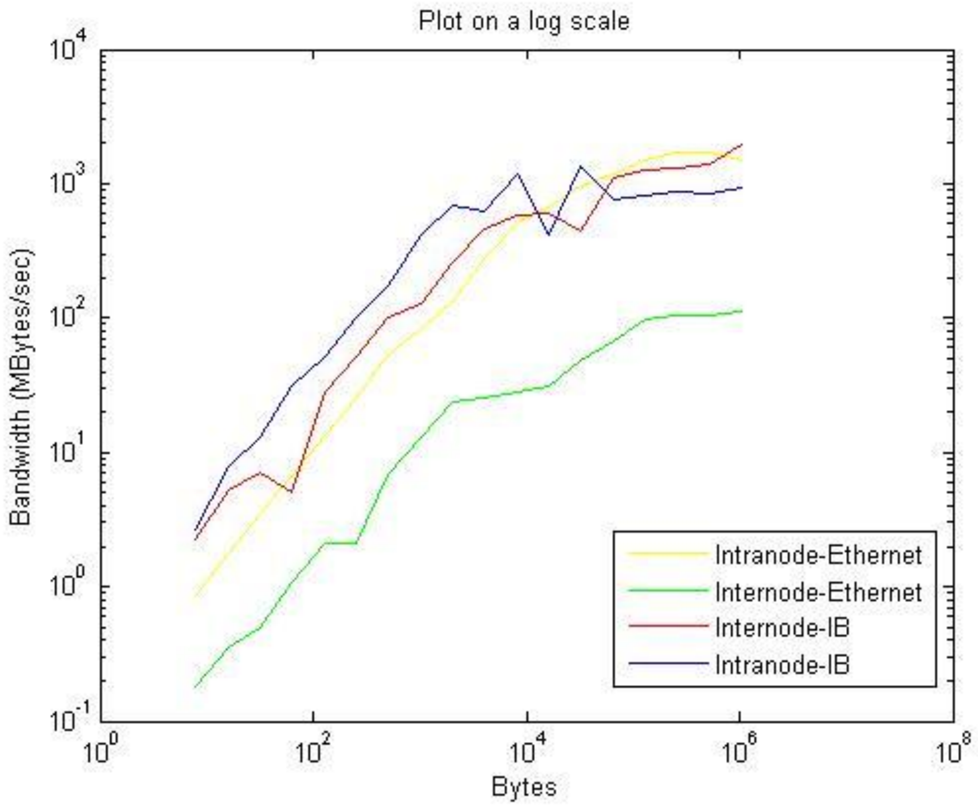
```
$gcc –o vector vector.c –L$MKLROOT/lin/intel64 –lmkl_intel_lp64 –lmkl_sequential –lmkl_core –lpthread –lm
```

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();
}
```