Homework 2 – Mohammad Atif Faiz Afzal

HPC1

Date- 10/2/2014

Problem 1:

(a)

The code for the dot product evaluation is attached in the appendix 1.

(b)

The below plot shows performance between different nodes (on CCR) when complied with BLAS and MKL.

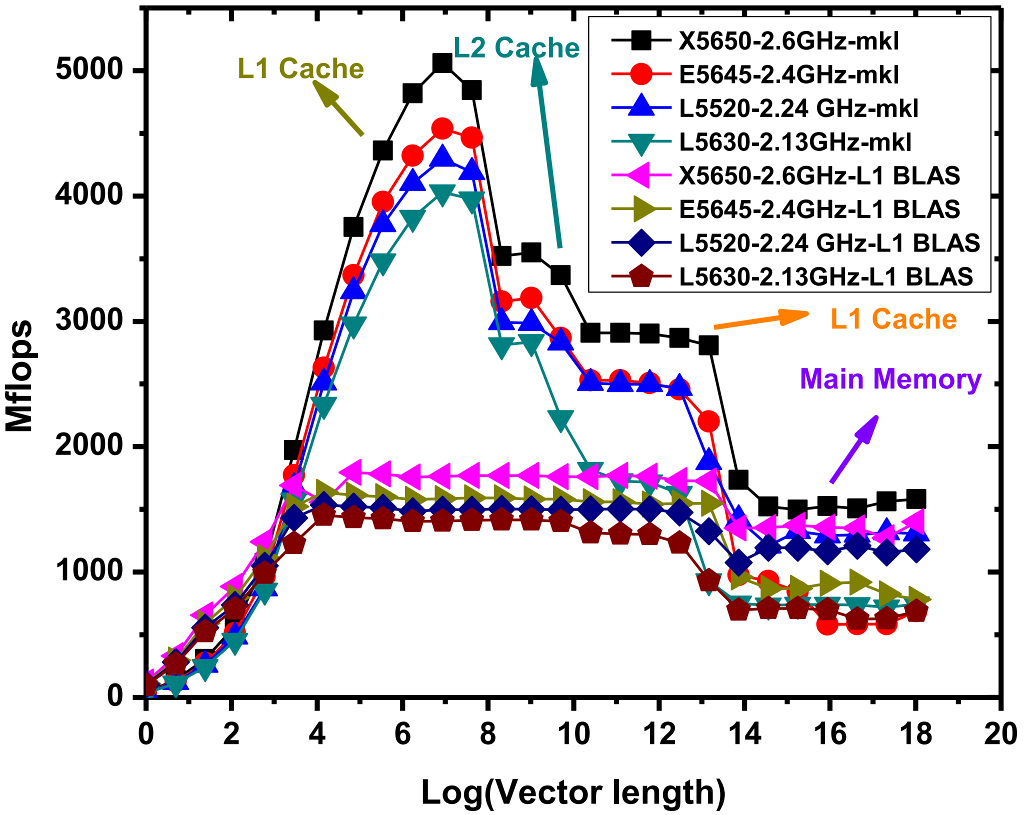

Figure 1– Performance between nodes when compiled with BLAS and MKL

L1 BLAS was used by running the following command

```
$gcc –o dotp dotp.c –lblas
```

whereas the MKL was used by running the following command (the extension was obtained from the intel link)

```
-$gcc –o dotp dotp.c -L$MKLROOT/lib/intel64 -lmkl_intel_lp64 -lmkl_core -lmkl_sequential -lpthread -lm
```

For further details on the code, please check the Appendix 1.

A slurm script was written to run the code on different nodes. The slurm script is attached in the Appendix 1.

Observations

- It can be seen from the plot that the performance of mkl compiler is better than BLAS compiler
- Smaller vectors use small memory caches and thus have better performance. Bigger vectors are too big for smaller memory caches and therefore use higher memory caches at the cost of performance. It can be seen that L1 cache performance is the best followed by L2,L3 and main memory
- As the nodes are changed, the performance also changes. Higher the clock speed, the better is the performance. From the plot, it can be seen that in both compilers the performance increases as the clock speed is increasing.

The below plot shows performance between different compiler flags

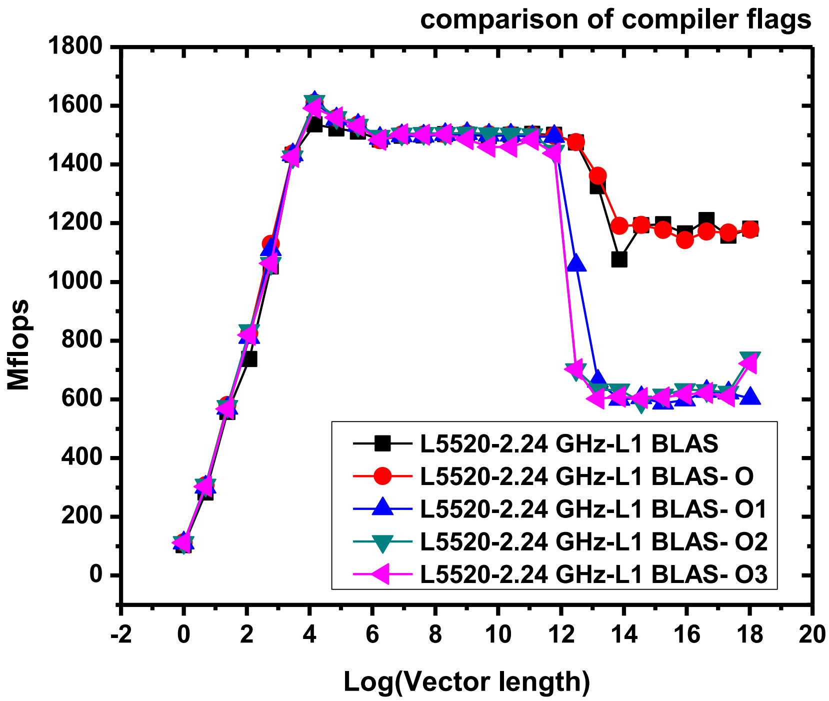

Figure 2– Performance between different compiler flags

Observations

- It can be seen from the plot that the performance does not change for small vectors
- Significant decrease in the performance can be seen for large vectors when the flags O1,O2 and O3 are applied
- As this decrease is seen only for large vectors, it can be said that this performance decrease occurs when main memory is being used

Problem 2:

(a) Ping pong code is obtained from the following link

http://levlafayette.com/node/464

The code is attached in Appendix 2. The code also outputs asynchronous and bi-directional asynchronous ping-pong outputs too (which are neglected in my analysis).

A slurm script is written to run the run the code on CCR. The code has been run on CPU- L5520 for all the runs made to make sure that the comparison can be made right.

Below plot shows the bandwidth variation with increase in buffer size

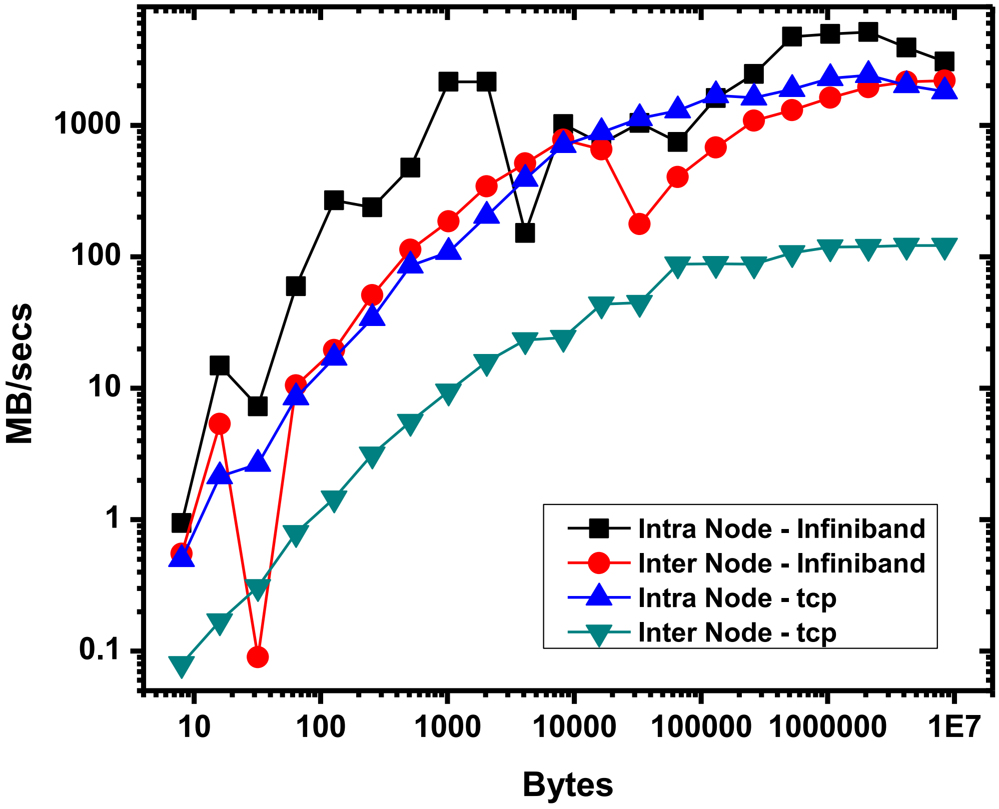

Figure 3– Bandwidth variation with buffer size

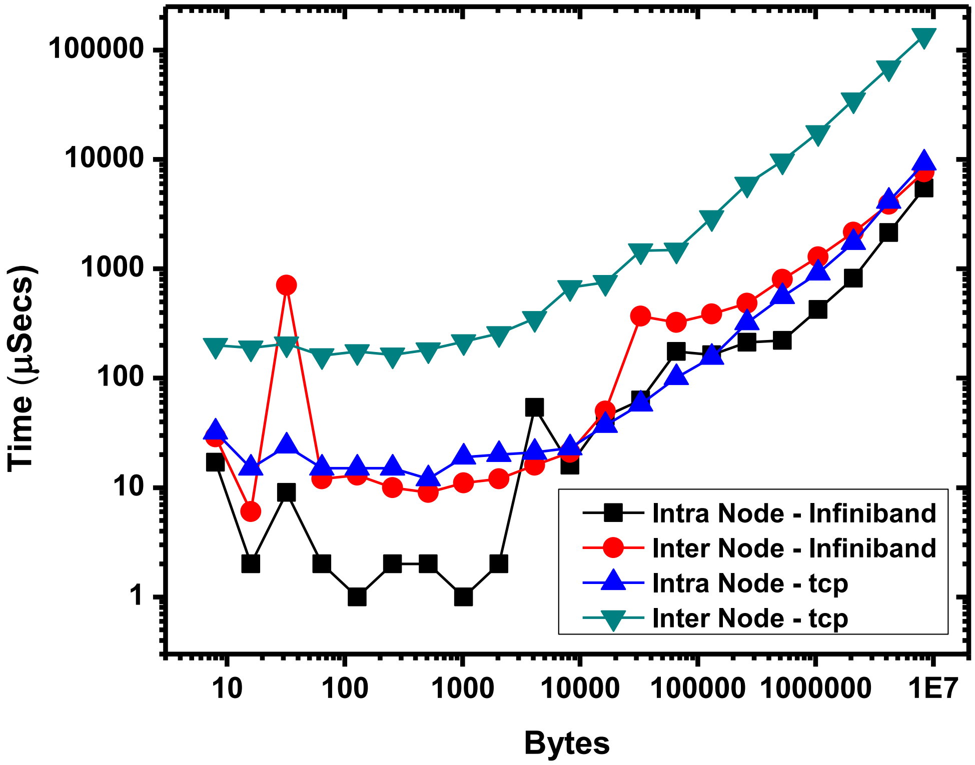

Figure 4– Message times with buffer size

Latencies

Type	Node/nodes used	Latency
Intra node - Infiniband	d07n33s01	0.476837 µsec
Inter node - Infiniband	d07n33s01, d07n33s02	2.980232 µsec
Intra node – TCP/IP	d07n33s01	5.960464 µsec
Inter node – TCP/IP	d07n33s01, d07n33s02	46.968460 µsec

Observations

- ➔ The comparison between infiniband and TCP/IP shows that the Infiniband is superior.
- ➔ In case of TCP/IP the intra node is highly superior when compared to intra node
- ➔ In case of Infiniband it is not clear whether inter node or intra node is better. From observation, it can be seen that that intra node is better than inter node for small buffer size
- ➔ The latency of inter node - TCP/IP is the highest , whereas, the latency of intra node - infiniband is the lowest

Appendix 1

Code for dot product (written in C language)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>
#include <math.h>

int main(int argc, char** argv)
{
  int maxlength= 100000000, i, n, veclength, k=1, tflops=1000000000;
  double  *vec1,*vec2,dotp,logval;
        struct timeval ti,tf;
        for (veclength=1; veclength<maxlength; veclength=veclength*5) {
          vec1=malloc(sizeof(double)*veclength);
          vec2=malloc(sizeof(double)*veclength);
          for(i=0;i<veclength;i++){
            vec1[i]=1.8+i;
            vec2[i]=1.3+i*0.4;
          }
          n=tflops/(2*veclength-1);
           gettimeofday(&ti,0);
           for (i=1;i<=n;i++) {
             dotp=ddot_(&veclength,vec1,&k,vec2,&k);
           }
           gettimeofday(&tf,0);
           double timeelapsed = ((tf.tv_sec - ti.tv_sec)*1000000 + (tf.tv_usec - ti.tv_usec));

           double Mflops=((2*(double)veclength-1)/timeelapsed)*(double)n;
           logval=log(veclength);
           printf("%f\t",logval);
           printf("%f\n",Mflops);
        }
        return 0;
}
```

Sample slurm script

```
#!/bin/sh
#SBATCH --partition=general-compute
#SBATCH --time=00:15:00
#SBATCH --job-name="dotp_E5645"
#SBATCH --output=dotp_E5645.out
#SBATCH --mail-user=m27@buffalo.edu
#SBATCH --mail-type=ALL

#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
echo "2.4GHz"
#SBATCH --constraint=CPU-E5645

tic=`date +%s`
echo "Start Time = "`date`
echo "Loading modules ..."

echo "Loading modules ..."
module load mkl

echo "SLURM job ID        = "$SLURM_JOB_ID
echo "Working Dir         = "`pwd`
echo "Compute Nodes       = "`nodeset -e $SLURM_NODELIST`

gcc -o dotp_E5645 dotp.c -lblas

srun ./dotp_E5645

echo "All Done!"

echo "End Time = "`date`
toc=`date +%s`

elapsedTime=`expr $toc - $tic`
echo "Elapsed Time = $elapsedTime seconds"
```

Appendix 2

Slurm script used to run the ping pong benchmark code

```
#!/bin/sh
#SBATCH --partition=debug
#SBATCH --time=00:15:00
#SBATCH --job-name="pingpong_21_infi"
#SBATCH --output=pingpong_21_infi.out
#SBATCH --mail-user=m27@buffalo.edu
#SBATCH --mail-type=ALL
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1

tic=`date +%s`
echo "Start Time = "`date`

echo "Loading modules ..."
module load intel
module load intel-mpi
ulimit -s unlimited
module list

echo "SLURM job ID        = "$SLURM_JOB_ID
echo "Working Dir         = "`pwd`

echo "Compute Nodes       = "`nodeset -e $SLURM_NODELIST`
echo "Number of Processors = "$SLURM_NPROCS
echo "Number of Nodes     = "$SLURM_NNODES
echo "mpirun command      = "`which mpirun`

export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
#export I_MPI_FABRICS="tcp"

mpicc -o pingpong_21_infi pingpong.c
srun ./pingpong_21_infi

echo "All Done!"

echo "End Time = "`date`
toc=`date +%s`

elapsedTime=`expr $toc - $tic`
echo "Elapsed Time = $elapsedTime seconds"
```

Ping pong benchmark code obtained from the website

http://levlafayette.com/node/464

I changed the size range in the code to cover the range from 8 Bytes to 8 MBytes

```
/* mpi-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>
#include <time.h>
#include <sys/time.h>
#include <math.h>

int 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*8 * sizeof (double));
b = (double *) shmalloc (132000*8 * sizeof (double));
#else
a = (double *) malloc (132000*8 * sizeof (double));
b = (double *) malloc (132000*8 * sizeof (double));
#endif
for (i = 0; i < 132000*8; 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) {
```

```
printf("Error: You don't have two processors available.");
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*8; 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();
return (0);
}
```