CS-301
High Performance Computing

Assignment 6

Amarnath Karthi 201501005
Chahak Mehta 201501422

November 20, 2017

1 Hardware and Compiler specifications

Architecture	x86_64
CPU op-mode(s)	32-bit, 64-bit
Byte Order	Little Endian
CPU(s)	12
On-line CPU(s) list	0-11
Thread(s) per core	1
Core(s) per socket	6
Socket(s)	2
NUMA node(s)	2
Vendor ID	GenuineIntel
CPU family	6
Model	63
Model name	Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz
Stepping	2
CPU MHz	1211.531
BogoMIPS	4804.10
Virtualization	VT-x
L1d cache	32K
L1i cache	32K
L2 cache	256K
L3 cache	15360K
NUMA node0 CPU(s)	0-5
NUMA node1 CPU(s)	6-11

Language	C
Compiler	mpicc

2 Trapezoidal Integration

Write and analyze serial code and parallel code for the following using OpenMP.

1. **Integration of a function using trapezoidal rule. Verify your code by using it to calculate the value of π.**
2. **Calculation of π using series.**
3. **Summation of two vectors**

3 Solution

3.1 Integration using trapezoidal rule

3.1.1 Theory

We use the following formula to compute the value of π :

$$\int_0^1 \frac{4}{1+x^2} dx \tag{1}$$

3.1.2 Approach

For the serial code, we use simple numerical integration over 0 to 1 using n segments for trapezoidal integration. For parallelizing we just have to divide these n segments between p processors and run the serial algorithm on each processor to get a local integration. After this, each processor sends the local integration answer to the master processor which adds each answer to get the global answer.
This is achieved using MPI.

3.2 Sending and Receiving schemes

We can use 2 send receive models in MPI -

1. **MPI_Send() , MPI_Recv()**: Using this model, the master sends the data n to each slave processor iteratively and then gets the data from each in the same manner. This is not efficient if p i.e. the number of processors is large.
2. **MPI_Bcast(), MPI_Reduce()**: Using this model, the master sends the data to some processors. As soon as they get the data they forward this to other processors also. This way each processor is helping in sending data, thereby reducing the communication time and increasing the bandwidth utilization. This is good if we have large number of processors.

3.2.1 Results and Observations

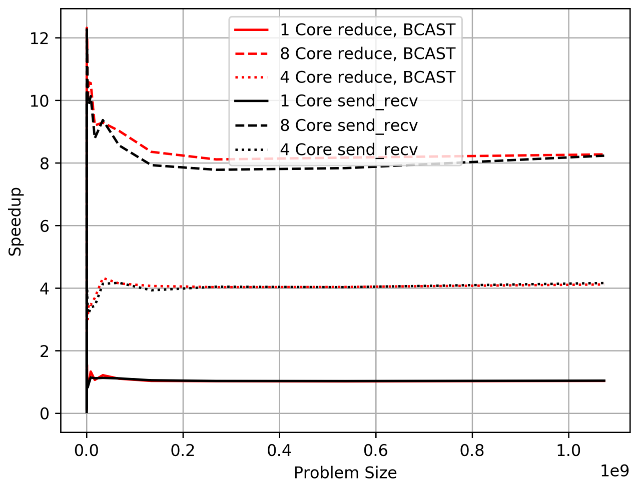

Figure 1: speedup vs problem size

The above plot shows speedup as a function of number of the problem size. Notice that as we increase the number of processors, the **MPI_Bcast()** model performs better due to the reasons mentioned in the previous section.
Also there is a slight superlinear speedup so the efficiency becomes slightly greater than 1.

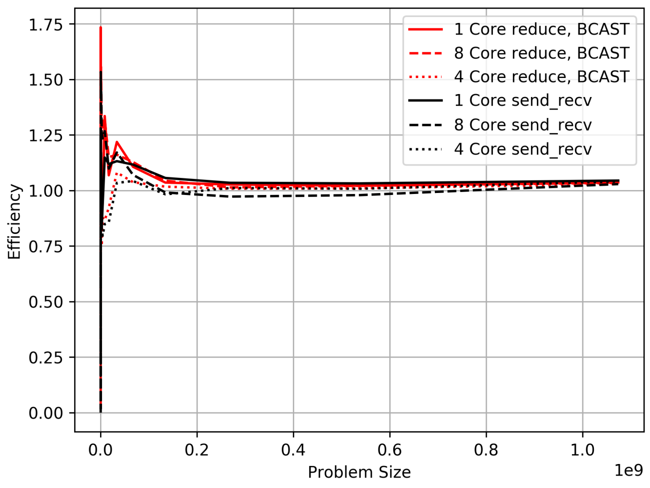

Figure 2: speedup vs problem size

4 Matrix Multiplication

Perform matrix multiplication using MPI

4.1 Approach

For the following operation :

$$AB = C$$

The serial approach to evaluate C is naive matrix multiplication which runs in $O(n^3)$ time complexity. To parallelize the code, we broadcast A to all processors and then decompose B and distrbute it amongst p processors.

4.2 Results and observations

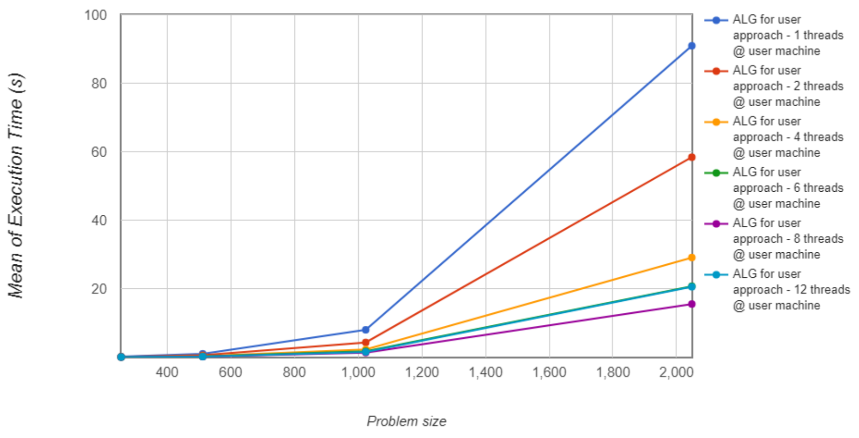

Figure 3: Problem size vs time taken

The time curve is shown above for 1, 2, 4, 8 and 12 threads. Notice that 8 thread model is faster than the 12 thread model. This isbecause the inter processor communication cost is higher in 12 threads and is not compensated by the computational ease provided by the same. This can also be seen from the speedup curves shown below.

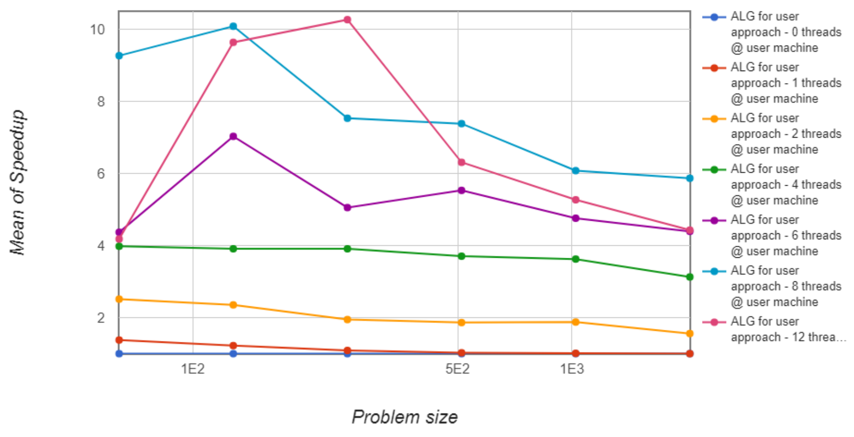

Figure 4: Problem size vs speedup