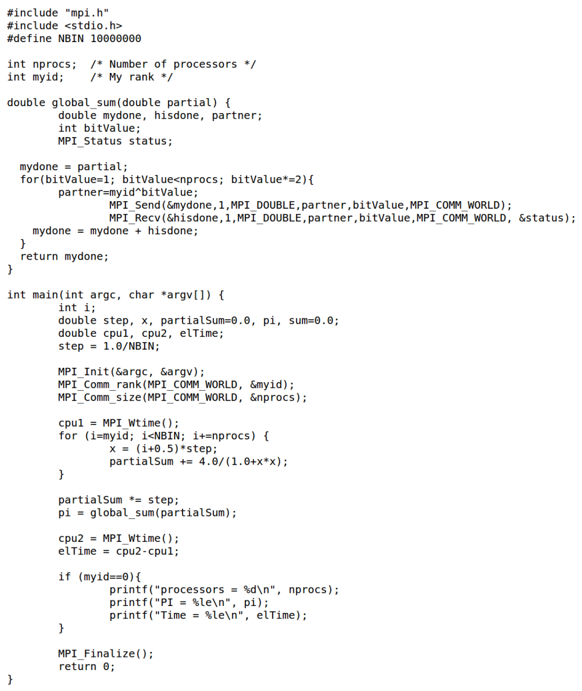
Assignment 3- Parallel Computation of Pi
CSCI 596: Scientific Computing & Visualization

Anup V Kanale

September 25, 2017

This purpose of this assignment is to understand the meaning of scalability analysis for parallel programs using a simple example– parallel computation of π.

1 MPI program to compute π



```
#include "mpi.h"
#include <stdio.h>
#define NBIN 10000000

int nprocs;  /* Number of processors */
int myid;    /* My rank */

double global_sum(double partial) {
        double mydone, hisdone, partner;
        int bitValue;
        MPI_Status status;

  mydone = partial;
  for(bitValue=1; bitValue<nprocs; bitValue*=2){
        partner=myid^bitValue;
                MPI_Send(&mydone,1,MPI_DOUBLE,partner,bitValue,MPI_COMM_WORLD);
                MPI_Recv(&hisdone,1,MPI_DOUBLE,partner,bitValue,MPI_COMM_WORLD, &status);
    mydone = mydone + hisdone;
  }
  return mydone;
}

int main(int argc, char *argv[]) {
        int i;
        double step, x, partialSum=0.0, pi, sum=0.0;
        double cpu1, cpu2, elTime;
        step = 1.0/NBIN;

        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &myid);
        MPI_Comm_size(MPI_COMM_WORLD, &nprocs);

        cpu1 = MPI_Wtime();
        for (i=myid; i<NBIN; i+=nprocs) {
                x = (i+0.5)*step;
                partialSum += 4.0/(1.0+x*x);
        }

        partialSum *= step;
        pi = global_sum(partialSum);

        cpu2 = MPI_Wtime();
        elTime = cpu2-cpu1;

        if (myid==0){
                printf("processors = %d\n", nprocs);
                printf("PI = %le\n", pi);
                printf("Time = %le\n", elTime);
        }

        MPI_Finalize();
        return 0;
}
```

Figure 1: C-program to compute pi parallely

2 Scalability

Let us now quantify the scalability of `global.pi.c`. There are two metrics here– fixed-problem size scaling and the isogranular scaling.

2.1 Fixed problem-size scaling

In this case, we keep the size of the problem constant, i.e., we run the program `global_pi.c` for a fixed number of quadrature points, `NBIN` $= 10^7$, for different number of compute nodes (or processors). The below plots summarizes the results.

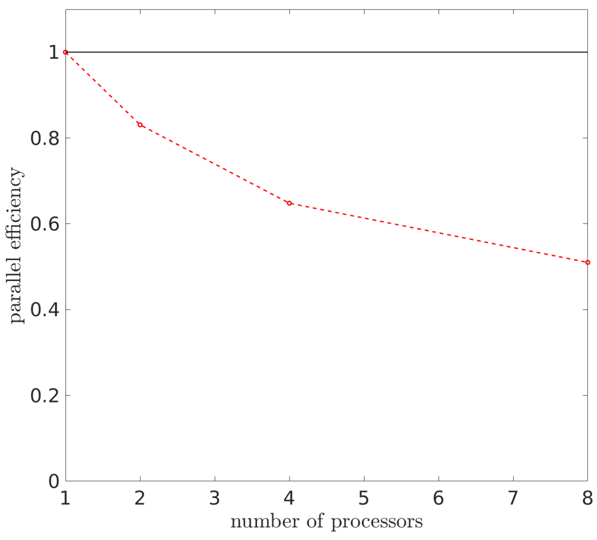


Figure 2: Fixed problem-size parallel efficiency

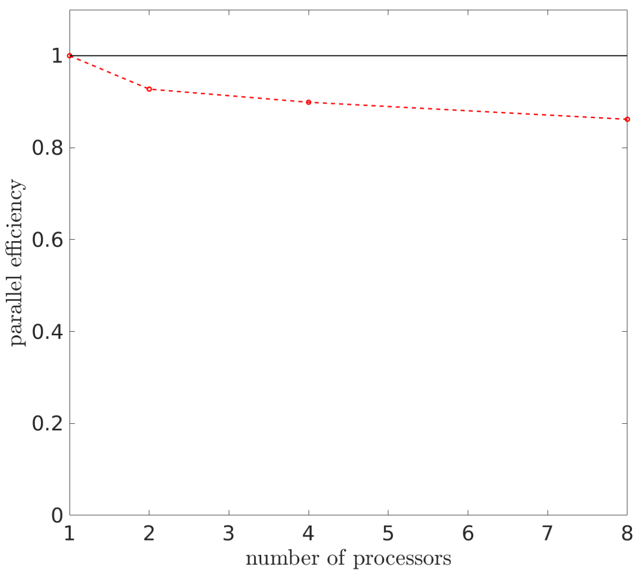


Figure 3: Isogranular parallel efficiency

2.2 Fixed problem-size scaling

In this case, the size of the problem scales with the number of processors, i.e., we run the program `global_pi.c` for a `NBIN` $= 10^7$. *per processor.* So for 1 processor, `NBIN` $= 10^7$, for 2 processors, `NBIN` $= 2 \times 10^7$, and for P processors, `NBIN` $= P \times 10^7$. The below plot summarises the results.

3 Conclusions

As seen from the above results, parallel computation can get very inefficient as the number of processors are increased for the same problem size. This is because more time is lost in communication between processors than for the actual computation. However, isogranular parallel efficiency decays at a much slower rate.

4 Potential Issues

1. Run time variance
   - Heterogeneous cluster–> run all P in one PBS file
   - Shared network–> run multiple times and take min
2. Effficiency>1
   - Measurement noise
   - Cache effect (only fixed problem size). won't happened for this HW.