1. The output results are
There are 50847534 primes less than or equal to 1000000000
largest prime under 1000000000 is: 999999937

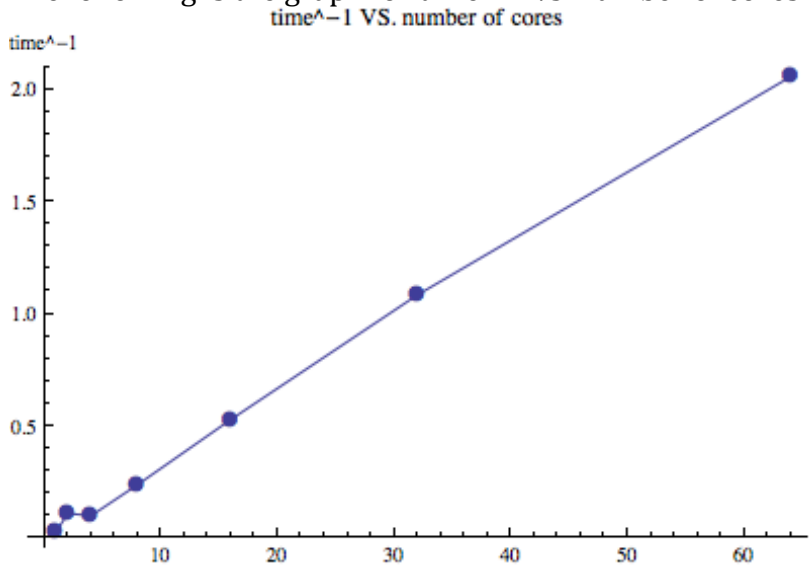
for all the cores, I get the same results.
Full results are attached at the end.

When I used different job request methods, I get different results.
One method is #PBS –l nodes=64:ppn=1
One method is #PBS –l nodes=6:ppn=12

The performance for nodes=64:ppn=1

# of cores	Time for calculation	Time^-1
1	33.837660	0.0295529
2	8.926943	0.11202
4	9.832782	0.101701
8	4.159676	0.240403
16	1.883718	0.530865
32	0.920974	1.08581
64	0.485961	2.05778

The following is the graph for time^-1 vs. number of cores

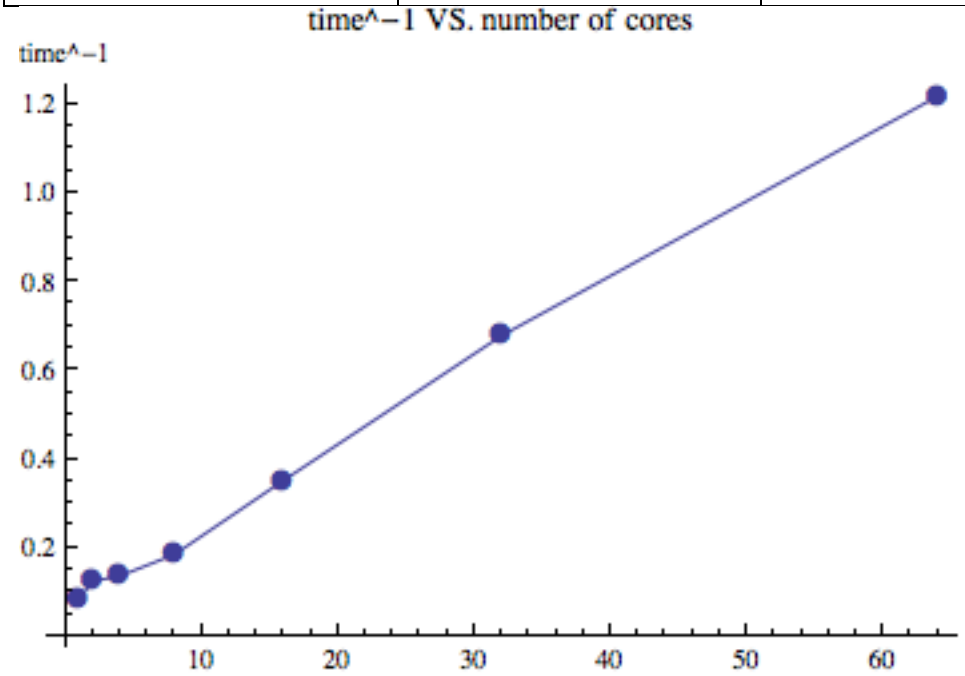


it seems that number of cores and performance have linear relationship.

Performance for nodes= nodes=6:ppn=12

# of cores	Time for calculation	Time^-1
1	11.775532	0.0849219
2	8.025935	0.124596
4	7.340342	0.136233

8	5.410584	0.184823
16	2.841941	0.351872
32	1.474556	0.67817
64	0.821028	1.21799



It also seems that number of cores and performance have linear relationship. But the performance of this version is not as good as the first one.

2.

For this problem, I permutated ji loop to ij loop in order to parallel i loop and make it possible to mpi reduce, send, bcase, receive multiple elements (e.g. MPI_Send(&hzp[nxp],ny,MPI_DOUBLE,id+1,10,MPI_COMM_WORLD);) of array during the calculation.

the output results are
Minhz= -3.055476047; Maxhz = 117.505802697; Mean= 0.866054434, RMS = 8.271862954

mpi_Minhz= -3.055476047; mpi_Maxhz = 117.505802697; mpi_Mean= 0.866054434, mpi_RMS = 8.271862954

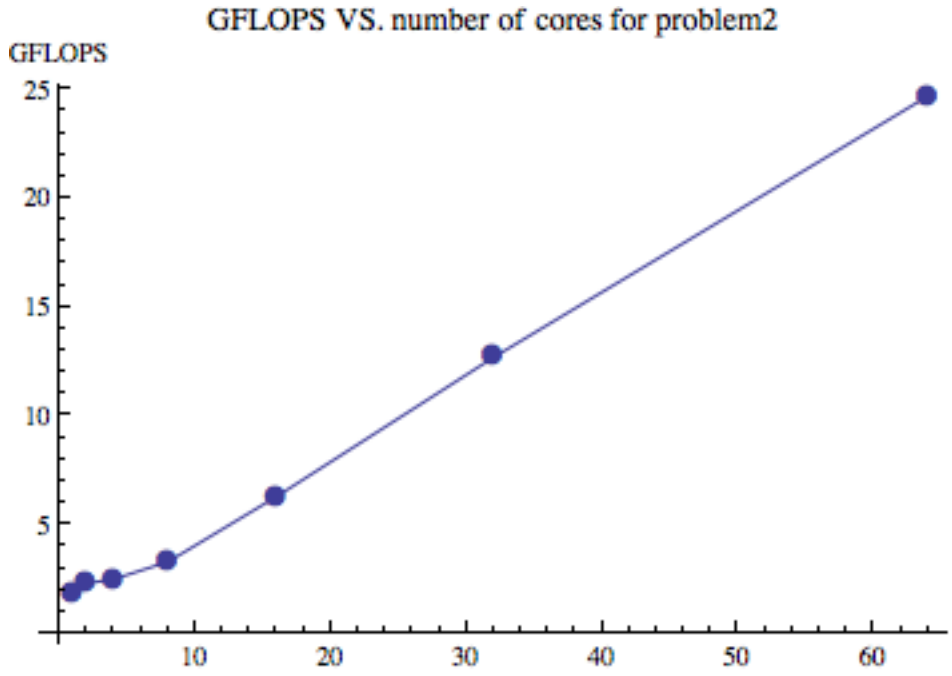
the results using all different number of cores are the same as the sequential codes. MPI code is correct!

The full results and the code are attached at the end.

The performance

Number of cores	GFLOPS	Time
Seq. reference	1.7	9.3
1	1.8	8.9
2	2.3	6.9
4	2.5	6.5
8	3.3	4.9
16	6.3	2.5
32	12.7	1.3
64	24.7	0.6

The following is the graph for GFLOPS vs. number of cores.



Code for problem 2
"fdtd-par-final.c"

```
#include <unistd.h>
#include <stdio.h>
#include <math.h>
#include <sys/time.h>
#include "mpi.h"
//#include "../MyMPI.h"
#define nx 4000
//#define nx 500
```

```
#define ny 4000
//#define ny 500
#define tmax 100
//#define tmax 1000
#define coeff1 0.5
#define coeff2 0.7

double ex[nx][ny];
double ey[nx][ny];
double hz[nx][ny];
double g_hz[nx][ny];
double gg_hz[nx][ny];
void check();
void dump();
void mpi_check();

int main(int argc, char *argv[]){
  double rtclock();

  int tt,i,j,nt;
  double clkbegin, clkend;
  double t, maxdiff;
    double g_t;

    int id;              // process ID number
    int p;               // number of processes
    MPI_Init (&argc, &argv);
    MPI_Comm_rank (MPI_COMM_WORLD, &id);
    MPI_Comm_size (MPI_COMM_WORLD, &p);
    MPI_Barrier(MPI_COMM_WORLD);

// Initialize arrays
    if(id==p-1){
    for (i=0; i<nx; i++){
    for (j=0; j<ny; j++) {
          ex[i][j] = sin(i)*(1-sin(j));
    }
    }
    for (i=0; i<nx; i++) {
    for (j=0; j<ny; j++) {
          ey[i][j] = cos(i)*(1-cos(j));
    }
    }
     printf("\n");
    for (i=0; i<nx; i++) {
    for (j=0; j<ny; j++) {
          hz[i][j] = sin(i)*(1-cos(j));
    }
    }
    clkbegin = rtclock();
```

```
    for (tt=0; tt<tmax; tt++){
      for (j=0; j<ny; j++)
      ey[0][j] = tt;

     for (j=0; j<ny; j++) {
          for (i=1; i<nx; i++){
               ey[i][j] = ey[i][j] - coeff1*(hz[i][j]-hz[i-
1][j]);
               }
          }
     for (j=1; j<ny; j++){
          for (i=0; i<nx; i++){
               ex[i][j] = ex[i][j] - coeff1*(hz[i][j]-hz[i][j-
1]);
               }
          }
     for (j=0; j<ny-1; j++){
          for (i=0; i<nx-1; i++){
               hz[i][j] =  hz[i][j] -
               coeff2*(ex[i][j+1]-ex[i][j]+ey[i+1][j]-
ey[i][j]);
               }
          }
    }
    clkend = rtclock();
    t = clkend-clkbegin;
    printf ("Sequential GFLOPS: %.1f, Time: %.1f\n",
10.0*nx*ny*tmax/t/1e9,t);
    check();
    }

    if (id==p-1){
     printf("seq end!\n");
    }
    MPI_Barrier(MPI_COMM_WORLD);
    int nxp;
    int m = nx%p;
    MPI_Status status;
// mpi initialize
    for(i=0;i<nx;i++)
     for(j=0;j<ny;j++){
          g_hz[i][j]=0.0;
          gg_hz[i][j]=0.0;
     }
    if (m==0)
     nxp = nx/p;
    else
     {
          if (id<m) nxp = nx/p+1;
          else  nxp = nx/p;
```

```
      }
  double exp[nxp+2][ny];
    double eyp[nxp+2][ny];
    double hzp[nxp+2][ny];
    int offset;
    if (id>=m) offset=m;
    else offset=0;
  for (i=1; i<nxp+1; i++){
     for (j=0; j<ny; j++) {
     exp[i][j] = sin(i-1+id*nxp+offset)*(1-sin(j));
     }
    }
    for (i=1; i<nxp+1; i++) {
    for (j=0; j<ny; j++) {
     eyp[i][j] = cos(i-1+id*nxp+offset)*(1-cos(j));
    }
    }
    for (i=1; i<nxp+1; i++) {
     for (j=0; j<ny; j++) {
     hzp[i][j] = sin(i-1+id*nxp+offset)*(1-cos(j));
    }
    }
    MPI_Barrier(MPI_COMM_WORLD);
//mpi calculate
  clkbegin = rtclock();
    int is, ie;
    if (id==0) is=2;
    else is=1;
    if (id==p-1) ie=nxp-1;
    else ie=nxp;
    //     MPI_Barrier(MPI_COMM_WORLD);
     if(id<p-1)
MPI_Send(&hzp[nxp],ny,MPI_DOUBLE,id+1,10,MPI_COMM_WORLD);
     if(id>0) MPI_Recv(&hzp[0],ny,MPI_DOUBLE,id-
1,10,MPI_COMM_WORLD,&status);
     if(id>0) MPI_Send(&eyp[1],ny,MPI_DOUBLE,id-
1,20,MPI_COMM_WORLD);
     if(id<p-1)
MPI_Recv(&eyp[nxp+1],ny,MPI_DOUBLE,id+1,20,MPI_COMM_WORLD,&status
);
     MPI_Barrier(MPI_COMM_WORLD);
  for (tt=0; tt<tmax; tt++){
    if(id ==0)
          for (j=0; j<ny; j++)
          eyp[1][j] = tt;

    for (i=is; i<=nxp; i++)
     {
     for (j=0; j<ny; j++){
          eyp[i][j] = eyp[i][j] - coeff1*(hzp[i][j]-hzp[i-
```

```
1][j]);
            }
      }
     //     MPI_Barrier(MPI_COMM_WORLD);
      if(id>0) MPI_Send(&eyp[1],ny,MPI_DOUBLE,id-
1,tt+tmax,MPI_COMM_WORLD);
      if(id<p-1)
MPI_Recv(&eyp[nxp+1],ny,MPI_DOUBLE,id+1,tt+tmax,MPI_COMM_WORLD,&s
tatus);
      MPI_Barrier(MPI_COMM_WORLD);

    for (i=1; i<=nxp; i++){
      for (j=1; j<ny; j++){
            exp[i][j] = exp[i][j] - coeff1*(hzp[i][j]-hzp[i][j-
1]);
            }
      }
    for (i=1; i<=ie; i++){
      for (j=0; j<ny-1; j++){
            hzp[i][j] =  hzp[i][j] -
              coeff2*(exp[i][j+1]-exp[i][j]+eyp[i+1][j]-
eyp[i][j]);
            }
      }
     //     MPI_Barrier(MPI_COMM_WORLD);
      if(id<p-1)
MPI_Send(&hzp[nxp],ny,MPI_DOUBLE,id+1,tt,MPI_COMM_WORLD);
      if(id>0) MPI_Recv(&hzp[0],ny,MPI_DOUBLE,id-
1,tt,MPI_COMM_WORLD,&status);
      MPI_Barrier(MPI_COMM_WORLD);
     }
     if(id<p-1)
MPI_Send(&hzp[nxp],ny,MPI_DOUBLE,id+1,tt,MPI_COMM_WORLD);
     if(id>0) MPI_Recv(&hzp[0],ny,MPI_DOUBLE,id-
1,tt,MPI_COMM_WORLD,&status);


  clkend = rtclock();
    MPI_Barrier(MPI_COMM_WORLD);
    t = clkend-clkbegin;
    MPI_Reduce(&t,&g_t,1,MPI_DOUBLE,MPI_MAX,0,MPI_COMM_WORLD);


// check
    MPI_Barrier(MPI_COMM_WORLD);
    for (i=1; i<=nxp; i++){
      for (j=0; j<ny; j++)
            g_hz[i-1+id*nxp+offset][j]=hzp[i][j];
    }
    MPI_Barrier(MPI_COMM_WORLD);
```

```
    for (i=0; i<nx; i++){

    MPI_Reduce(&g_hz[i],&gg_hz[i],ny,MPI_DOUBLE,MPI_SUM,0,MPI_COM
M_WORLD);
    }
    MPI_Barrier(MPI_COMM_WORLD);
    if(id==0) {
    printf ("MPI parellel GFLOPS: %.1f, Time: %.1f\n",
10.0*nx*ny*tmax/g_t/1e9,g_t);
     mpi_check();
    }
    MPI_Finalize();
}


double rtclock()
{
  struct timezone Tzp;
  struct timeval Tp;
  int stat;
  stat = gettimeofday (&Tp, &Tzp);
  if (stat != 0) printf("Error return from gettimeofday:
%d",stat);
  return(Tp.tv_sec + Tp.tv_usec*1.0e-6);
}

void check()
{
double maxval, minval, mean, rms;
int i,j;
  minval = maxval = hz[0][0];
  mean = rms = 0.0;
  for (i=0;i<nx;i++)
   for (j=0;j<ny;j++)
    {
     if (hz[i][j] < minval) minval = hz[i][j];
     if (hz[i][j] > maxval) maxval = hz[i][j];
     mean += hz[i][j]/(1.0*nx*ny);
     rms += hz[i][j]*hz[i][j]/(1.0*nx*ny);
    }
  rms = sqrt(rms);
  printf("Minhz= %18.9f; Maxhz = %18.9f; Mean= %18.9f, RMS =
%18.9f\n", minval,maxval,mean,rms);
}


void mpi_check()
{
double maxval, minval, mean, rms;
int i,j;
```

```
  minval = maxval = hz[0][0];
  mean = rms = 0.0;
  for (i=0;i<nx;i++)
   for (j=0;j<ny;j++)
    {
     if (gg_hz[i][j] < minval) minval = gg_hz[i][j];
     if (gg_hz[i][j] > maxval) maxval = gg_hz[i][j];
     mean += gg_hz[i][j]/(1.0*nx*ny);
     rms += gg_hz[i][j]*gg_hz[i][j]/(1.0*nx*ny);
    }
  rms = sqrt(rms);
  printf("mpi_Minhz= %15.9f; mpi_Maxhz = %15.9f; mpi_Mean=
%15.9f, mpi_RMS = %15.9f\n", minval,maxval,mean,rms);

}
```

For One method is #PBS –l nodes=64:ppn=1

Programming Assignment 4, Problem 1

--- 1 core ---
There are 50847534 primes less than or equal to 1000000000
SIEVE (1) 33.837660
largest prime under 1000000000 is: 999999937

--- 2 core ---
largest prime under 1000000000 is: 999999937
There are 50847534 primes less than or equal to 1000000000
SIEVE (2) 8.926943

--- 4 core ---
largest prime under 1000000000 is: 999999937
There are 50847534 primes less than or equal to 1000000000
SIEVE (4) 9.832782

--- 8 core ---
largest prime under 1000000000 is: 999999937
There are 50847534 primes less than or equal to 1000000000
SIEVE (8) 4.159676

--- 16 core ---
largest prime under 1000000000 is: 999999937

There are 50847534 primes less than or equal to 1000000000
SIEVE (16) 1.883718

--- 32 core ---
largest prime under 1000000000 is: 999999937
There are 50847534 primes less than or equal to 1000000000
SIEVE (32) 0.920974

--- 64 core ---
There are 50847534 primes less than or equal to 1000000000
largest prime under 1000000000 is: 999999937
SIEVE (64) 0.485961

One method is #PBS –l nodes=6:ppn=12

Programming Assignment 4, Sieve, 1 processes

There are 50847534 primes less than or equal to 1000000000
SIEVE (1) 15.660846
largest prime under 1000000000 is: 999999937

Programming Assignment 4, Sieve, 2 processes
---

largest prime under 1000000000 is: 999999937
There are 50847534 primes less than or equal to 1000000000
SIEVE (2) 10.039494

Programming Assignment 4, Sieve, 4 processes
---

largest prime under 1000000000 is: 999999937
There are 50847534 primes less than or equal to 1000000000
SIEVE (4) 9.351460

Programming Assignment 4, Sieve, 8 processes
---

largest prime under 1000000000 is: 999999937
There are 50847534 primes less than or equal to 1000000000
SIEVE (8) 7.867141

Programming Assignment 4, Sieve, 16 processes
---

largest prime under 1000000000 is: 999999937
There are 50847534 primes less than or equal to 1000000000
SIEVE (16) 5.385425

Programming Assignment 4, Sieve, 32 processes
---

largest prime under 1000000000 is: 999999937
There are 50847534 primes less than or equal to 1000000000
SIEVE (32) 2.527521

Programming Assignment 4, Sieve, 64 processes

There are 50847534 primes less than or equal to 1000000000
SIEVE (64) 1.389446
largest prime under 1000000000 is: 999999937

Programming Assignment 4, problem2 fdtd, 1 processes

Sequential GFLOPS: 1.7, Time: 9.3
Minhz= -3.055476047; Maxhz = 117.505802697; Mean= 0.866054434, RMS = 8.271862954
seq end!
MPI parellel GFLOPS: 1.8, Time: 8.9
mpi_Minhz= -3.055476047; mpi_Maxhz = 117.505802697; mpi_Mean= 0.866054434, mpi_RMS = 8.271862954

```
------------------------------------------------------------
Programming Assignment 4, problem2 fdtd, 2 processes
------------------------------------------------------------

Sequential GFLOPS: 1.7, Time: 9.3
Minhz=      -3.055476047; Maxhz =      117.505802697; Mean=       0.866054434,
RMS =       8.271862954
seq end!
MPI parellel GFLOPS: 2.3, Time: 6.9
mpi_Minhz=    -3.055476047; mpi_Maxhz =   117.505802697; mpi_Mean=
0.866054434, mpi_RMS =     8.271862954

------------------------------------------------------------
Programming Assignment 4, problem2 fdtd, 4 processes
------------------------------------------------------------

Sequential GFLOPS: 1.6, Time: 9.8
Minhz=      -3.055476047; Maxhz =      117.505802697; Mean=       0.866054434,
RMS =       8.271862954
seq end!
MPI parellel GFLOPS: 2.5, Time: 6.5
mpi_Minhz=    -3.055476047; mpi_Maxhz =   117.505802697; mpi_Mean=
0.866054434, mpi_RMS =     8.271862954

------------------------------------------------------------
Programming Assignment 4, problem2 fdtd, 8 processes
------------------------------------------------------------

Sequential GFLOPS: 1.7, Time: 9.3
Minhz=      -3.055476047; Maxhz =      117.505802697; Mean=       0.866054434,
RMS =       8.271862954
seq end!
MPI parellel GFLOPS: 3.3, Time: 4.9
mpi_Minhz=    -3.055476047; mpi_Maxhz =   117.505802697; mpi_Mean=
0.866054434, mpi_RMS =     8.271862954

------------------------------------------------------------
Programming Assignment 4, problem2 fdtd, 16 processes
------------------------------------------------------------

Sequential GFLOPS: 1.6, Time: 9.8
```

Minhz= -3.055476047; Maxhz = 117.505802697; Mean= 0.866054434, RMS = 8.271862954
seq end!
MPI parellel GFLOPS: 6.3, Time: 2.5
mpi_Minhz= -3.055476047; mpi_Maxhz = 117.505802697; mpi_Mean= 0.866054434, mpi_RMS = 8.271862954

Programming Assignment 4, problem2 fdtd, 32 processes

Sequential GFLOPS: 1.7, Time: 9.4
Minhz= -3.055476047; Maxhz = 117.505802697; Mean= 0.866054434, RMS = 8.271862954
seq end!
MPI parellel GFLOPS: 12.7, Time: 1.3
mpi_Minhz= -3.055476047; mpi_Maxhz = 117.505802697; mpi_Mean= 0.866054434, mpi_RMS = 8.271862954

Programming Assignment 4, problem2 fdtd, 64 processes

Sequential GFLOPS: 1.6, Time: 9.8
Minhz= -3.055476047; Maxhz = 117.505802697; Mean= 0.866054434, RMS = 8.271862954
seq end!
MPI parellel GFLOPS: 24.7, Time: 0.6
mpi_Minhz= -3.055476047; mpi_Maxhz = 117.505802697; mpi_Mean= 0.866054434, mpi_RMS = 8.271862954