CS 240   Homework 2

Lan Liu 7956394
Pritha D N 9727371

Experiment 1: Strong scaling analysis

1. Set k=864, as a common multiple of p=1,4,8,12,16,24,48,72.
2. Use compute node for each running.
3. Stop condition: relres < 1e-8.
k=864 is fairly large, based on hw2harness.h, *If you want to verify correctly on large problems (k > 400 or so), then reduce the norm of the residual requirement by an order of magnitude or two.*
4. After 2198 iterations, cgsolve stops and the norm of residual is 3.77e-7.

Remark: Here we did not use value p=32 and p=64, instead, we use p=48 and 72, which are multiple of 24. We have two reasons to do this:
Reason 1: save SUs.
For each compute node, the total processor number is 24, and no matter how many processors we actually use, the SUs consumed will always be counted as 24. For p=64, it requires 4 compute nodes with 16 processor on each node, and need 4*24=96 SUs. But for p=72, we only need 3 nodes with 24 processors on each node, and need 3*24=72 SUs. And multiple of 24 will make full use of processor and save SUs. [This was told by Burak]

Reason2: save running time.
From the result below, we can see that p=64 takes longer time than p=48, 24, 16. It is because communication within a node is faster than communication between nodes, and it is also cheaper. So we should set p be multiples of 24, i.e. 24, 48 and 72.

Number of processors	Time taken	Efficiency
1	24.056	
4	6.979	0.862
8	4.084	0.736
12	2.839	0.706
16	2.118	0.710
24	1.47	0.682
48	0.823	0.609
72	0.680	0.491
32	1.117	0.673
64	2.910	0.129

Form1: Result for k=864, iteration steps =2198, residual=3.77e-7

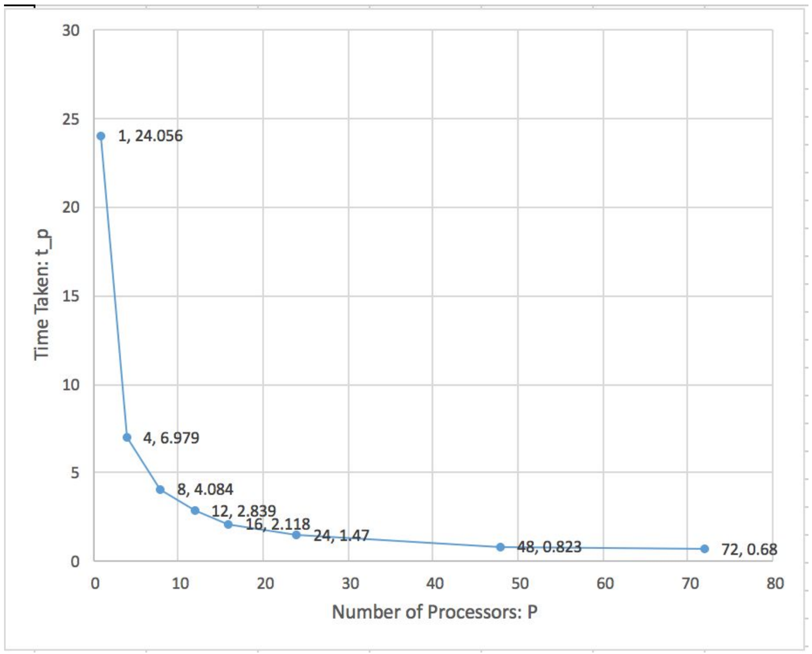


Figure 1: Running time versus number of processors P

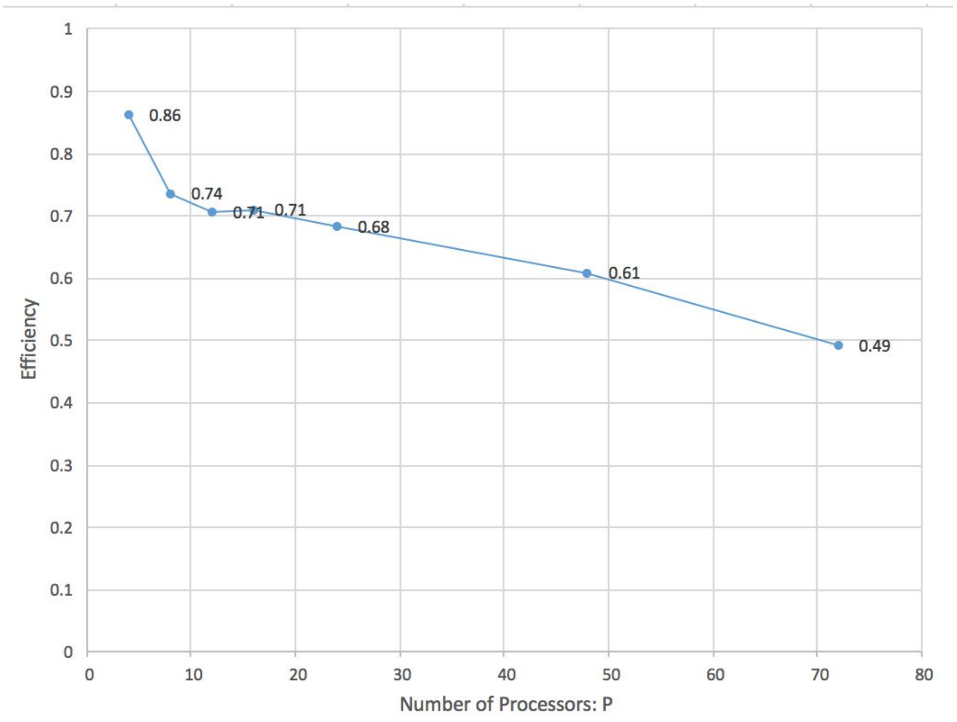


Figure 2: Efficiency versus number of processors P

As number of processors increase, the efficiency will decrease, this match with Latency/ Bandwidth Model. The concrete analysis will be on the last part.

Experiment 2: Weak scaling analysis

K	Number of processors	Time taken	Efficiency
864	1	1.088321	
1728	4	1.178593	0.231
2440	8	1.417219	0.096
2988	12	1.539454	0.059
3456	16	1.818838	0.0373
4224	24	3.114038	0.0145
6000	48	3.648195	0.00621
7344	72	Times out	

Form 2: K is the closest number proportional to sqrt(p) and also multiple of p. Do 100 iterations.

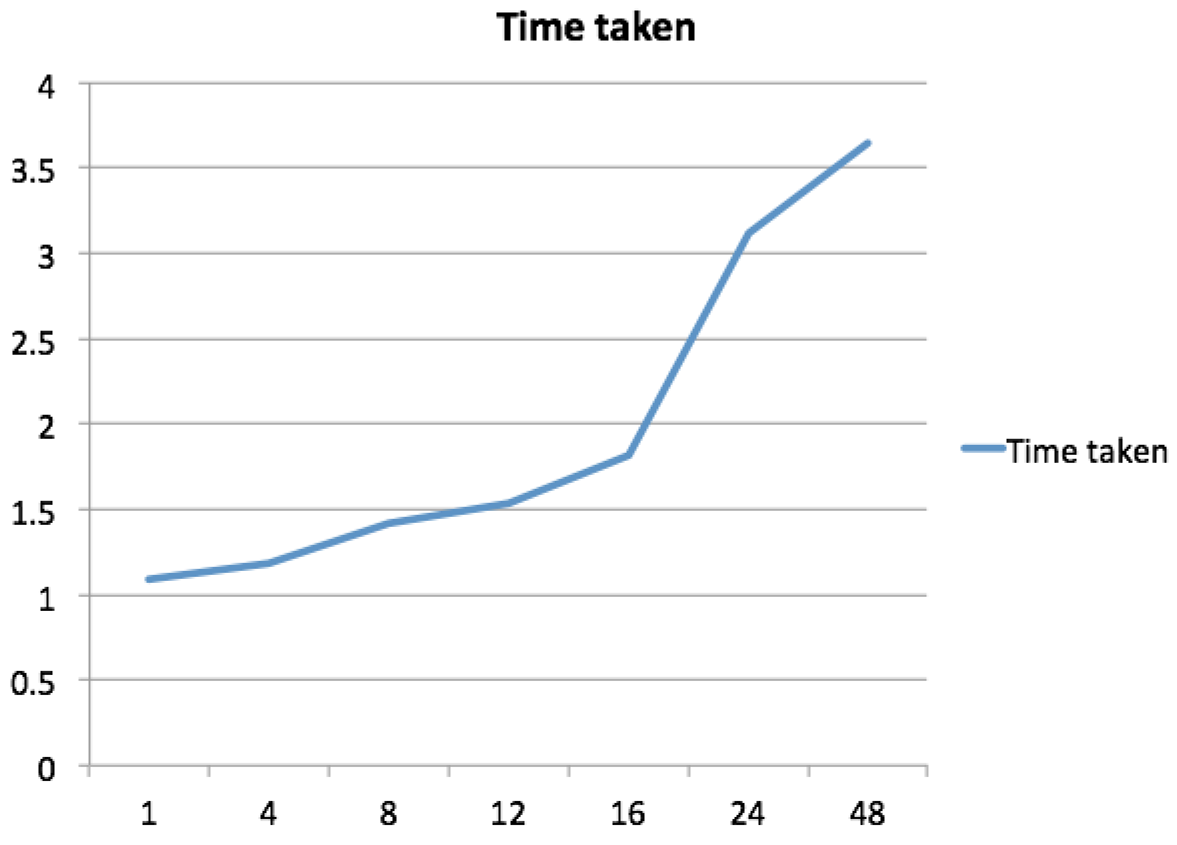


Figure 3: Running time versus number of processors P

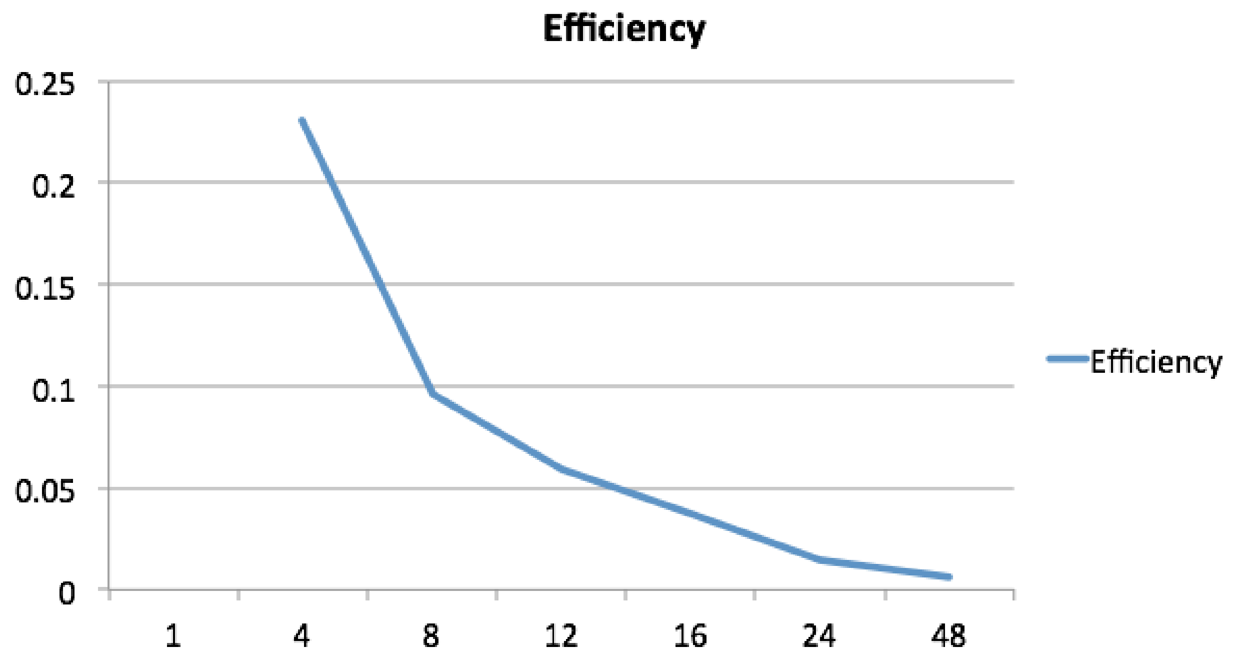


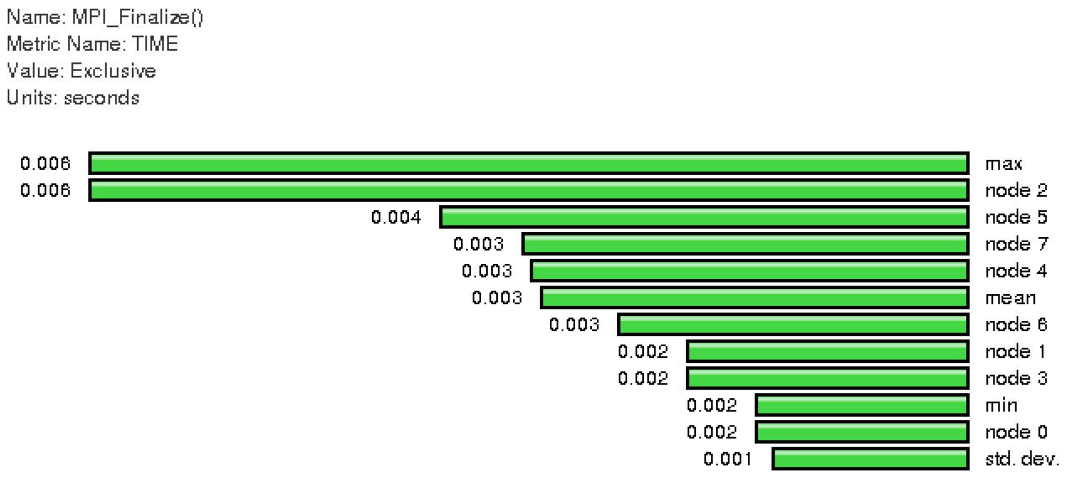
Figure 4: Efficiency versus number of processors P

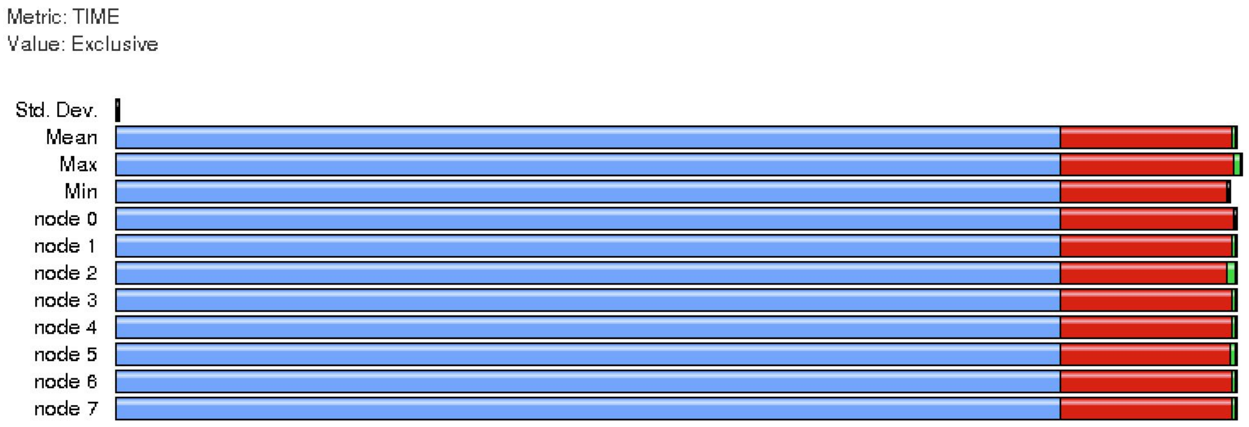
Experiment 3: TAU analysis of the performance

K = 144; p = 8; maxiterations = 10

```
NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.741          940           1          77     940066 int main(int, char **) C
 84.2          791          791           1           0     791795 MPI_Init()
 15.4          145          145           1           0     145113 void save_vec(int, double *) C
  0.2            1            1           1           0       1544 MPI_Finalize()
  0.0         0.35         0.35          30           0         12 MPI_Bcast()
  0.0        0.273        0.273          21           0         13 MPI_Reduce()
  0.0        0.162        0.162           1           0        162 MPI_Gather()
  0.0        0.064        0.064          10           0          6 MPI_Recv()
  0.0        0.023        0.023          10           0          2 MPI_Send()
  0.0        0.001        0.001           1           0          1 MPI_Comm_size()
  0.0            0            0           1           0          0 MPI_Comm_rank()
---------------------------------------------------------------------------------------

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
        30          8          8          8          0  Message size for broadcast
         1  2.074E+04  2.074E+04  2.074E+04          0  Message size for gather
        21          8          8          8          0  Message size for reduce
---------------------------------------------------------------------------------------
```

```
NODE 2;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.721          939           1          97     939992 int main(int, char **) C
 84.2          791          791           1           0     791775 MPI_Init()
 14.9          140          140           1           0     140416 void save_vec(int, double *) C
  0.7            6            6           1           0       6350 MPI_Finalize()
  0.0        0.458        0.458          30           0         15 MPI_Bcast()
  0.0        0.155        0.155          21           0          7 MPI_Reduce()
  0.0        0.059        0.059          20           0          3 MPI_Recv()
  0.0        0.041        0.041          20           0          2 MPI_Send()
  0.0        0.016        0.016           1           0         16 MPI_Gather()
  0.0        0.001        0.001           1           0          1 MPI_Comm_size()
  0.0            0            0           1           0          0 MPI_Comm_rank()
---------------------------------------------------------------------------------------

USER EVENTS Profile :NODE 2, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
        30          8          8          8          0  Message size for broadcast
         0          0          0          0          0  Message size for gather
        21          8          8          8          0  Message size for reduce
---------------------------------------------------------------------------------------


FUNCTION SUMMARY (total):
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0            5        7,520           8         736     940029 int main(int, char **) C
 84.2        6,334        6,334           8           0     791763 MPI_Init()
 15.3        1,149        1,149           8           0     143682 void save_vec(int, double *) C
  0.3           24           24           8           0       3092 MPI_Finalize()
  0.0            3            3         240           0         15 MPI_Bcast()
  0.0            1            1         168           0          7 MPI_Reduce()
  0.0        0.727        0.727         140           0          5 MPI_Recv()
  0.0        0.345        0.345         140           0          2 MPI_Send()
  0.0        0.293        0.293           8           0         37 MPI_Gather()
  0.0        0.006        0.006           8           0          1 MPI_Comm_size()
  0.0        0.002        0.002           8           0          0 MPI_Comm_rank()

FUNCTION SUMMARY (mean):
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.738          940           1          92     940029 int main(int, char **) C
 84.2          791          791           1           0     791763 MPI_Init()
 15.3          143          143           1           0     143682 void save_vec(int, double *) C
  0.3            3            3           1           0       3092 MPI_Finalize()
  0.0        0.445        0.445          30           0         15 MPI_Bcast()
  0.0        0.138        0.138          21           0          7 MPI_Reduce()
  0.0       0.0909       0.0909        17.5           0          5 MPI_Recv()
  0.0       0.0431       0.0431        17.5           0          2 MPI_Send()
  0.0       0.0366       0.0366           1           0         37 MPI_Gather()
  0.0      0.00075      0.00075           1           0          1 MPI_Comm_size()
  0.0      0.00025      0.00025           1           0          0 MPI_Comm_rank()
```





Blue - MPI_Init; Red - save_vec; Green - MPI_Finalize

Per iteration:

1. Total number of send/receive = 2(p-1), since first and last processor only send one set of k elements. Other processors send 2k elements.
2. Total broadcasts = 3p (alpha, beta, relres)
3. Total reduce = 2p [We initialize rtr before the loop begins which counts as p computations, independent of iterations]
4. Total gather = p

4. Analysis with Communication Volume Model

4.1 Sequential Matlab Code

```
while relres > 1e-6  &&  niters < maxiters
    niters = niters+1;
    Ad = matvec(d,n);          % MATVEC
    alpha = rtr / (d'*Ad);     % DDOT
    x = x + alpha * d;         % SAXPY
    r = r - alpha * Ad;        % SAXPY
    rtrold = rtr;
    rtr = r'*r;                % DDOT
    beta = rtr / rtrold;
    d = r + beta * d;          % SAXPY
    relres = sqrt(rtr) / normb;
end;
```

P=1, for each iteration step, the communication cost is 0, and computation cost is (only count product and division, ignore lower order):

1 MATVEC: 5n

3 SAXPY: 3n

2 DDOT: 2n

Total cost per step is 10n.

4.2 Parallel Code

```
while(relres > 1e-8 && *niters < maxiterations)
{
   *niters = *niters+1;
    Ad = matvec(d, n, rank, p);
    dot_product = ddot(d,Ad,n,p);
    MPI_Reduce(&dot_product,&dAd,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
    if(rank==0)
    {
        alpha = rtr / dAd;
    }
    MPI_Bcast(&alpha,1, MPI_DOUBLE,0,MPI_COMM_WORLD);
    x =saxpy(alpha,x,d,n,p);
    r =saxpy(-1*alpha,r,Ad,n,p);
    if(rank==0)
    {
        rtrold = rtr;
    }
    dot_product=ddot(r,r,n,p);
    MPI_Reduce(&dot_product,&rtr,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);

    if(rank==0)
    {
        beta = rtr / rtrold;
    }
    MPI_Bcast(&beta,1, MPI_DOUBLE,0,MPI_COMM_WORLD);
    d = saxpy(beta,r,d,n,p);
    if(rank==0)
    {
        relres = sqrt(rtr) / normb;
    }
    MPI_Bcast(&relres,1, MPI_DOUBLE,0,MPI_COMM_WORLD);

}
```

	computation cost		communication cost
1 MATVEC	5n/p	~2p MPI_Send/Receive	2k*(p-2)+k*2
3 SAXPY	3n/p	3 MPI_Bcast	3(p-1)
2 DDOT	2n/p	2 MPI_Reduce	2(p-1)
Total	**10n/p**	1 MPI_Gather	n
		Total	**2kp+n**

Note: For Matvec function, each processor other than first and last should send twice k elements and receive twice k elements, and the first and last processor should send and receive once k elements. So the total cost is 2k*(p-2)+k*2

As k increase, the computation cost will decrease and the communication cost will increase. For experiment 2, if we set k be about proportional to sqrt(p), then computation cost n/p=k^2/p is about a constant, but the communication cost will increase as k and p increase, so we will see a pattern that as p and k increase, the running time is increasing.