1.)

One Machine

Number of Processes	Residual Norm	Runtime (seconds)	Speedup
1	.000010	12.5912292004	1
2	.000010	6.8196249008	1.84632283792074
4	.000010	4.0626068115	3.09929800854911
8	0.00010	3.3171279430	3.795822596162068

Distinct Machines

Number of Processes	Residual Norm	Runtime (seconds)	Speedup
1	.000010	12.5905439854	1
2	.000010	7.9924309254	1.5753084515734963
4	.000010	6.4020240307	1.9666505350532626
8	.000010	6.1409161091	2.050271288797209

Discussion:
When running the experiment on one machine the run-times and speedups are consistent with what is expected. The run-times decrease and and speedups increase as the number of processes go up. However, the difference between the number of processes and speedup declines as the number of processes go up. This is expected as the use of more processes requires the use of more process communication.

When running the experiment on distinct machines the run-times decrease and the speedups increase as the number of processes used grows. However, in comparison to running on one machine the run-times are much larger and the speedups are much smaller. This occurs because in both cases as the number of processes used grows the amount of required process communication grows as well. However, the price of process communication on one machine is significantly less expensive than the price on distinct machines.

2.)

Run-time Analysis:

One of the first steps of the program is to calculate the matvec multiplication. The code for this can be seen on lines 142-149. Since the nnz for each process is assumed to be shared equally the run-time for this calculation is:

$$3 * (nnz(A)/P)$$

The next step of the program is to calculate the dot product of <rk, rk> and <pk, sk>. These values are used as the numerator and denominator of the alpha calculation. This code can be found on lines 152-155. The run-time for this operation is:

$$4 * (n/P)$$

Once the dot products have been found for the local values the value of sk must be gathered so that all processes can calculate their appropriate alphas. This code can be found on lines 156-157. The run-time for this communication is (where the message length of a double is considered to be 1):

$$2 * \log_2(P)(t_s + 1*t_w)$$

Now the local values for the next iteration of x and r can be calculated. The code for this can be seen on lines 161-164. The run-time for this calculation is:

$$4*(n/P)$$

The local maximum value of the residual vector r is then calculated. This will be used later to determine the convergence criteria. The code for this is on lines 166-170. The run-time is:

$$(n/P)$$

Now the maximum value of the residual vector must be calculated for all processes. To do this a reduction is used to find the maximum value for all the local maximums. This can be seen on line 172 in the code. The runtime for this communication is (where the message length of a double is considered to be 1):

$$\log_2(P)(t_s + 1 * t_w)$$

Now the new beta value must be calculated. Since the beta calculation's denominator is a previously found value only the numerator has to be calculated. This allows for the saving of a reduction. The code for this numerator calculation is on lines 179-181 and the run-time is:

$$2 * (n/P)$$

As in the alpha calculation this numerator needs to be combined globally using a sum reduction. The code for this is on line 182. The run-time for this communication is (where the message length of a double is considered to be 1):

$$\log_2(P)(t_s + 1 * t_w)$$

Now the local p value has to be updated for the next iteration. This code for this calculation is on lines 185-187. The run-time is:

$$2 * (n/P)$$

Finally, each process needs to gather the local p array from each other process for the next iteration. This is performed in the code on line 188. This communication has run-time (where the message length of a double is considered to be 1):

$$\log_2(P)(t_s + (n/P) * t_w)$$

Therefore, the total run-time of the program can be simplified to:

$$(nnz(A)/P) + (n/P) + \log_2(P)(t_s + 1 * t_w) + \log_2(P)(t_s + (n/P) * t_w)$$

3.)

```
input: A, b, initial guess x_0, tol (tolerance)
output: final iterates of x, inf_norm (r).

Scatter the rows of A, distributing the the nnz's evenly

j = 0; {iteration index}
r_0 = b – A * x_0; {residual vector}
p_0 = r_0; {search direction to update solution}
Compute <r_0, r_0>;

while (j < = N) do {if not converged iterate to max no. iterations}
     s_j = A * p_j; {auxiliary vector for new search vector; Matvec}

     {Gather the values of local s_j into the s_j_global}
     AllGather(s_j, Num_or_Row/Num_of_Proc, double, s_j_global,
          Num_or_Row/Num_of_Proc, double, Comm)
     alpha_j = <r_j, r_j> / <s_j_global, p_j>; {step length}

     x_{j+1} = x_j + alpha_j p_j; {next iterate of the solution}
     r_{j+1} = r_j - alpha_j s_j_global; {next iterate of the
residual}

     {Check convergence}.
     If (inf_norm (r_{j+1}) = tol
          converged = true;
          break;
      endif

     {compute values needed in next iteration}
     beta_j = <r_{j+1}, r_{j+1}> / <r_j, r_j>;
     p_{j+1} = r_{j+1} + beta_j p_j; {update search direction}
     j = j+ 1; {update index for next iteration}
end While
end Procedure
```

Run-time Analysis:

The first calculation the algorithm must perform is the matvec multiplication. This will be the same as the previous method. Its run-time is:

$$(nnz(A)/P)$$

Now a gather must be performed to get all the values that belong to the vector s. This ensures that each process contains its own local copy of s. The run-time for this communication is:

$$\log_2(P)(t_s + (n/P) * t_w)$$

Now alpha must be calculated. Because all redundant calculations are performed on all processes the runtime of this calculation will be:

$$\text{some constant} * n$$

After alpha's calculation vectors x and r must be updated for the next iteration. Again, because redundant calculations are performed on every process the run-time will be:

$$\text{some constant} * n$$

Now beta most be calculated and vector p must be updated for the next iteration. Again, because redundant calculations are performed on every process the run-time will be:

$$\text{some constant} * n$$

Therefore, the total run-time of this algorithm will be:

$$(nnz(A)/P) + \log_2(P)(t_s + (n/P) * t_w) + n$$

Discussion:

This method would be slower than the one implemented in problem 1. While it saves time by performing less process communication, the time spent performing redundant operations makes the algorithm's run-time grow much faster. While the problem 1 algorithm performs the majority of it's calculations at a rate of (n/P) this algorithm performs its operations at a rate of n. Thus, not taking full advantage of the parallelization possible in the conjugate gradient algorithm.