First Revision of How the algorithm works

In a similar vein to all the functions, gaussJordan.c divides the number of rows done among the processes available.

```
void gaussJordanSolve(float *A, float *B, int numLines, int processNum){
    int j = 0;
    for (int i = 0; i < 3; i++){
        //Here, we take our pivot (make sure it's not zero) and divide the
whole row
        if (!processNum)
            divideRow(A, numLines, processNum, i);
        sweep(A, i, numLines, processNum);
    }
}
```

so This function is used to solved the linear system of equations (In this case, I'm testing using a 3x4 matrix)

Let's say I'm running three parallel processes. Each process will take a row, find it's pivot (Eg the second row of Matrix A has pivot a_{22}), and divide the entire row by the pivot, if the pivot is nonzero.

```
void divideRow(float *A, int numLines, int processNum, int i){
    float divisor = *(A + i + 4*i);
    if (!divisor)
        return;
    for (int j = i; j < 4; j++)
        A[4 * i + j] /= divisor;
}
```

It will then proceed to "sweep" the column that the pivot is on (Eg the pivot a_{22} for row 2 is also on column 2), thus turning all the elements in the columns to zero, other than the pivot of course. This action, due to how Gauss Jordan elimination works, turns the matrix in reduced echelon form, thus giving us the solution if there is one

```
void sweep(float *A, float *B, int pivot, int numLines, int processNum){
    int annihilationMultiple;
    int start = numLines * processNum;
    //this for loop sweeps the current column
    for (int  k = start; k < numLines; k++){
        if (k == pivot)
            continue;
        annihilationMultiple = -1 * A[pivot + 4 * k];
        for(int m = pivot; m < 4; m++){
            A[m + 4 * k] += A[m + 4 * pivot] * annihilationMultiple;
        }
    }
```

This algorithm is faulty because when you divide all the rows first and then proceed to sweep, you might end up with a race condition. You also would defeat the purpose of dividing the row by the pivots, because by sweeping a row that's already been divided, you potentially change the pivot from one, which will mess up the whole algorithm.

Revision 2

I have created an algorithm that solves 3x4 matrices using either one or two concurrent processes.

We can compare the average time of PPGJ vs SPGJ

Matrix size	PPGJ	SPGJ
3	0.000002sec	.000001sec
4	0.000003sec	.000001sec
5	.000044sec	.000003sec
6	0.051786sec	.000003sec
7	.075878sec	.000003sec
8	.096040sec	.000004sec

9	.117008sec	.000007sec
10	.144897sec	.000008sec

PPGJ performs worse that SPGJ, and the reason for this is unclear, although it may be linked to how the synch functions are utilized.