Tarea 5 - Computo Paralelo

Rubén Pérez Palacios - 19 Mayo 2022 - Profesor: Luis Daniel Blanco Cocom

Introducción

Se implemento los productos $Ax, x \cdot b$, donde $A \in M_{N \times N}, x, b \in R^N$.

Compilación

El algoritmo lee de un archivo *.mat* la matriz A (en su forma compresa por coordenadas), los vectores b, x. Cuyo nombre se tiene que pasar como parámetro.

Comando para compilar:

- mpicxx Tarea6.cpp -o Tarea6

Compando para ejecutar:

- mpiexec -n 4 Tarea6

Donde RUTA es la la dirección relativa donde se encuentra el archivo .mat

Código

Se implemento una clase para almacenar una Matriz en su forma "Compressed Row Storage":

```
class CRS
{
public:
    int N, M; //Dimensiones
    vector < vector < int > > indexJ; //Matriz de indices de columnas
    vector < vector < double > > vals; //Valores distintos a 0
    vector < double > diag; //Valores de la diagonal

    //Convertir una matriz comprimida por coordenadas a por renglones
    void fill_from_sparse(int N, int M, int nonzero, double* vals,
     double* indexI, double* indexJ)
    {
        this->N = N;
        this->M = M;
        this->indexJ.resize(N);
        this->vals.resize(N);
        diag.resize(N);

        for (int k = 0; k < nonzero - 1; k++)
        {
            this->indexJ[indexI[k] - 1].push_back(indexJ[k] - 1);
```

```
            this->vals[indexI[k] - 1].push_back(vals[k]);
            if (indexI[k] == indexJ[k])
                diag[indexI[k]-1] = vals[k];
        }
    }
};
```

Se implemento una función que divide las posiciones a trabajar de un arreglo acorde al número de procesos en paralelo.

```
void procesor_range( const int rank, const int nprocs, const int N, int& tl, int& tr )
{
    //...
}
```

Puesto que el producto de dos vectores está dado por

$$\sum_{i=1}^{N} x_i b_i, x = (x_1, \cdots, x_n), b = (b_1, \cdots, b_n)$$

Entonces si dividimos las componente entre todos los procesos, entonces cada uno se encarga una parte de esta suma. De esta forma se implemento la siguiente función que se encarga de hacer la suma respectiva al proceso en curso

```
double dot_prod( const int rank, const int nprocs, const vector < double > & a, const vecto
{
    const int N = a.size();
    int tl, tr;
    double local_sum = 0.0, sum = 0.0;
    procesor_range(rank, nprocs, N, tl, tr);

    for (int i = tl; i <= tr; i++)
        local_sum += a[i] * b[i];

    MPI_Reduce(&local_sum, &sum, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

    return sum;
}
```

En la cual la instrucción $MPI\_Reduce()$ se encarga de juntar cada suma.

Ahora como el producto de matriz vector está dado por

$$Ax_{i,j} = \sum_{i=k}^{N} A_{i,k} x_k$$

entonces los cálculos de los renglones de Ax puede ser divido entre todos los procesos. Se implemento la siguiente función que se encarga de hacer el cálculo de los renglones correpondientes a cada proceso

```
vector < double > mat_vec_prod( const int rank, const int nprocs, const CRS& A, const vecto
{
    const int N = A.N;
    int tl, tr, tsize;
    MPI_Status status;

    procesor_range(rank, nprocs, N, tl, tr);
    tsize = tr - tl + 1;

    vector < double > ans(N);
    for (int i = tl; i <= tr; i++)
        for (int j = 0; j < A.indexJ[i].size(); j++)
            ans[i] += A.vals[i][j] * x[(int)A.indexJ[i][j]];

    if (rank == 0)
    {
        for (int i = 1; i < nprocs; i++)
        {
            int local_tl, local_sz;
            MPI_Recv(&local_tl, 1, MPI_INT, i, 101, MPI_COMM_WORLD, &status);
            MPI_Recv(&local_sz, 1, MPI_INT, i, 102, MPI_COMM_WORLD, &status);
            MPI_Recv(&ans[local_tl], local_sz, MPI_DOUBLE, i, 103, MPI_COMM_WORLD, &status);
        }
    }
    else
    {
        MPI_Send(&tl, 1, MPI_INT, 0, 101, MPI_COMM_WORLD);
        MPI_Send(&tsize, 1, MPI_INT, 0, 102, MPI_COMM_WORLD);
        MPI_Send(&ans[tl], tsize, MPI_DOUBLE, 0, 103, MPI_COMM_WORLD);
    }

    return ans;
}
```

Donde las tres intrucciones MPI_Send se encargan de enviar apartir de que renglón tl hará el cálculo, cuántos $tsize$ renglones calculará, y donde guardar el resultado; y las tres intrucciones MPI_Recv se encargan de recibir apartir de que renglón tl se hizo el cálculo, cuántos $tsize$ renglones se calcularon y donde guardar el resultado.

Por último se uso la función $read\_system\_eq()$ implementada en la tarea pasada para leer la matriz A y los vectores x, b. Se inicio la paralelización de MPI con MPI_Init y se guardo el número de procesos MPI_Comm_size y el proceso

en curso MPI_Comm_rank. Se llamo a las repectivas funciones para hacer los calculos de Ax y $x \cdot b$, y en caso de ser el proceso 0 entonces se imprimio el resultado puesto que este es el que recibe todos los resultados. Finalmente se termino la paralelización de MPI con $MPI\_Finalize$.