TECHNISCHE UNIVERSITÄT BERGAKADEMIE FREIBERG

Die Ressourcenuniversität. Seit 1765.

Introduction to High Performance Computing and Optimisation

Project Winter Semester 2021/2022

Submitted by:

Harihara Rajan Ramaswamy

Matriculation Number: 65076

harihara-rajan.ramaswamy@student.tu-freiberg.de

Contents

1 Problem Statement

To implement a parallel dense matrices and vectors using Message Passing Interface (MPI) in C++ programming language. Further to compute the largest eigen value of matrix and corresponding eigen vector by parallel implementation of Power Iteration method: an algorithm that produce a number λ, which is the greatest eigen value of a matrix A and a non-zero vector v which is a corresponding eigen vector of λ i.e. Av =λv. This method is also called as Von-Mises iteration.

2 Brief Overview

2.1 Project Description

GooglePageRank We denote the importance of (page rank) of the i_{th} web-page by r_i, $i \in 1, ...n$. We define the entries of a Connectivity matrix or Link matrix L $\in \mathbb{R}^{n \times n} by$,

$$l_{ij} = \begin{cases} 1\text{, if a link from } j_{th} \text{ web-page to the } i_{th} \text{ web-page exists} \\ 0\text{, otherwise} \end{cases} \tag{1}$$

for the rank page vector $\boldsymbol{r} = (r_1, r_2, ...r_n)^T$ the following conditions hold,

$$r_i = \sum_{i=1}^{n} \frac{1}{n_j} l_{ij} r_j \tag{2}$$

$$\sum_{i=1}^{n} r_i = 1 \tag{3}$$

$$r_i \geq 0 \forall i = 1, ...n \tag{4}$$

where n_i denotes the number of links from i_{th} web-page.

The Matrix L represent the graph of web. To make sure the user of the web does not get stuck on a web-page that has no link to the other web-page, The L matrix is modified as following: We define a matrix Q by $Q_{ij} = \frac{1}{n_j} l_{ij}$ and denote the j_{th} column Q by Q_j. Furthermore, we define the vector $e = (1, ...1)^T$, the vector d by,

$$d_j = \begin{cases} 1\text{, Q}_j = 0 \\ 0\text{, otherwise} \end{cases} \tag{5}$$

and the matrix $P = Q + \frac{1}{n} e d^T$. In the columns of Q with zero entries are replaced by the vector $\frac{1}{n} e$. This models the case that for a web-page with no links to one another, the user visits the random web-page with uniform distribution. P is the *left stochastic matrix*, meaning that sum of each column will be equal to one.

The Eigen value problem $Pr = \lambda_{max} r$ can be solved by using Power Iteration method.

```
for k = 1,2,.... until convergence do,
    q(k) = P r(k-1)
    r(k) = q(k)/||q(k)||
end
```

The eigen value λ_{max} can be approximated by the *Rayleigh Quotient*

$$\theta_k = \frac{< r_k, Pr_k >}{r_k \times r_k} \tag{6}$$

2.2 Project Implementation

In order to have faster code and to efficiently utilise all cores in machine, This project majorly employs Message Passing Interface (MPI) functions like $MPI - Scatter, MPI - Gather, MPI - Allreduce\ and\ MPI - Bcast$ for a parallel implementation. The example of each MPI function is illustrated with an example from the implemented project below.

2.2.1 MPI-Scatter

MPI Scatter is used to send data from one process to other process or processes in a communicator. *MPI Scatter* sends chunks of array to different processes. In each of different processes a particular operation will be carried out parallelly. The function prototype of *MPI Scatter* looks like

```
MPI_Scatter(void* send_data, int send_count, MPI_Datatype send_datatype,
void* recv_data, int recv_count, MPI_Datatype recv_datatype, int root,
MPI_Comm communicator)
```

here send data is the one that needs to scattered into different processes. The second and third argument dictates the number of elements of a particular data type that needs to be scattered into each process. The first three arguments of a *MPI Scatter* belongs to the data that is being sent. The second three arguments of *MPI Scatter* belongs to the receiving parameters of the function prototype and it is identical to the first three parameters. recv data is a buffer in each process which collects the data that is scattered. Note that, the send count, receive count should be same also send data type and the receive data type should be same. The last two parameters, root and communicator indicates the root process that is scattering the elements of an array and the communicator

In this work, at many instances scatter operations have been performed to perform computations in different processes, one such instance is initialising the L matrix (connectivity matrix - section 2.1) with zeros and ones parallelly.

```
MPI_Scatter(&L, chunks*n, MPI_INT, &L_sub, chunks*n, MPI_INT, 0, MPI_COMM_WORLD);
for (int i=0; i<chunks; i++)
    {
        for (int j=0; j<n; j++)
        {
            L_sub[i][j] = rand() % 2
        }
    }
```

here L is connectivity matrix of shape [n][n] is scattered into L sub of shape [chunks][n] to different processes and initialise with random number (0 or 1) parallelly and assemble the matrix together into L matrix using Gather operation. Gather operation will be dicused in the following section

2.2.2 MPI-Gather

The inverse operation to *MPI Scatter* is *MPI Gather*. *MPI Gather* takes the element from all the process and gathers them into one single process. The function prototype of the *MPI Gather* looks like,

```
MPI_Gather(void* send_data, int send_count, MPI_Datatype send_datatype, void* recv_data,
int recv_count, MPI_Datatype recv_datatype, int root, MPI_Comm communicator)
```

like scatter operation, Gather operation have been performed at many instances. one such instances is gathering L sub from all the process into L matrix in root process again.

```
MPI_Gather(&L_sub,chunks*n, MPI_INT, &L, chunks*n, MPI_INT, 0, MPI_COMM_WORLD);
```

In section 2.2.1, L sub in different processes is assigned to zero or one. After which we need to gather L sub from different processes into L matrix. L sub argument in *MPI Gather* takes the L sub from the particular process and assign it back to the L matrix.

2.2.3 MPI-Allreduce

$MPI - Allreduce$takes an array of input elements on the root process and distribute the reduced outputs to all other process. The output elements contain the reduced result. The prototype for MPI-Allreduce looks like,

```
MPI_Allreduce(void* send_data, void* recv_data, int count, MPI_Datatype datatype,
MPI_Op op, MPI_Comm communicator)
```

In this project, to compute the links from j_{th} web page to i_{th} web page, we need to take the column sum of matrix L. for which we will scatter the matrix L into different processes and compute the sum of columns for each chunks and then finally use the All reduce function to compute the sum over all chunks. This can be achieved with following lines of code

```
MPI_Scatter(L, n*chunks, MPI_INT, L_sub, n*chunks, MPI_INT, 0, MPI_COMM_WORLD);
for (i=0; i<chunks; i++)
{
    for (j=0; j<n; j++)
    {
        n_links [j] += L_sub[i][j];
    }
}
MPI_Allreduce(& n_links, n_links_total, n, MPI_INT, MPI_SUM, MPI_COMM_WORLD );
```

Allreduce perform sum over n links from different process and send it to n links total and finally broadcast it to all other processes.

2.2.4 MPI-Bcast

MPI-Bcast is used to send the same data from the root process to all other existing processes in a communicator.In multiple occasions, MPI-Bcast have been used with in this project work.

```
MPI_Bcast(&res, n, MPI_INT, 0, MPI_COMM_WORLD);
```

here res is the rank vector updated during every power iteration which is of length n and of integer type. At the end every power iteration res i.e. rank vector is broadcasted from the root process to all other processes in the communicator.

3 Results

In order to assess the performance improvement with increase in processes, A random 10000 web page connections are created and tried to identify the rank of the every web-page using different number of processes. The result of which is shown below in the plot.

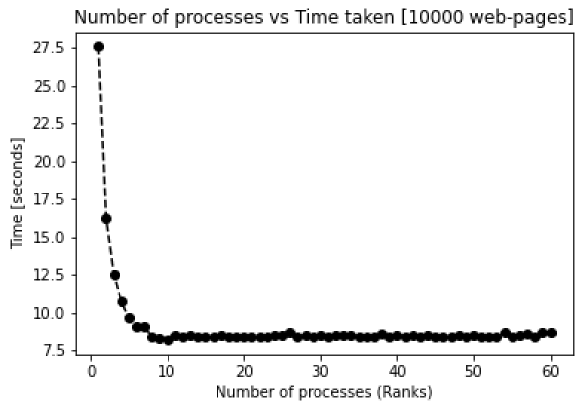

Figure 1: Number of process vs Time

Initially, there is drastic decrease in time for computing the rank of 10000 web-pages when increasing the process size. But after certain number of increment(in this case after 9 processes), the time required for computing the rank for 10000 web-pages gets saturated almost remains constant. This phenomenon is due to the fact that after certain number of increment, the time taken to interact between the processes gets increased which play a decisive role in over all time required to compute the rank.In order to get maximum time reduction and for an effective utilisation of processes, it is best to use 10 number of processes rather than using higher number of processes. Since from the plot it is evident that increasing the number of processes after certain increment will not yield better results in terms of time required for computation