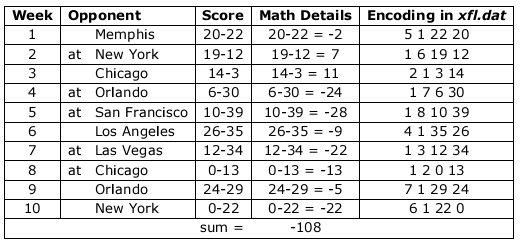
Russell Ratcliffe

Lab4 Football Power Rankings

The point of this lab was to learn about collective communication by building a program to estimate “power rankings” for teams based on their previous games. The power rating measures how good a team is compared to the other teams in the league. We assume that the equation to calculate the power rating is POWER\_home\_team - POWER\_visiting\_team = SCORE\_home\_team - SCORE\_visitng\_team. I utilised the Jacobi method to calculate the power ratings. The jacobi method is an algorithm for solving a system of linear equations by iteration. Once the system converges you can stop iterating.

To calculate the power rating, I read the teams from a xfl.dat file. I then assign each of the teams to a different slave process. I use 0 as my master process. My master process reads the game scores from a file, and stores them in an array. It then splits the game scores into arrays separated by each team and then sends the arrays to the slaves. In my first implementation, I used the XFL league which consists of 8 teams and they play a total of 10 games. The figure below represents all of the data in the XFL files.



For each process I want to create an equation similar to:

10\*Power(1)=2\*power(5)+power(3)...-108

This is used to find an approximate power rating for each team. For the first iteration I set all of the opposing team's power rating to 100. To accomplish this goal I had to make use of the MPI\_Gather and MPI\_Bcast operations. MPI\_Gather is used to fill an array from elements sent to the master process from its slaves. This array contains all of the power ranks. MPI\_Bcast is used to send its power ranking to every other process.

After several iterations I found the approximate power ratings of the teams. Below is a diagram of the final power ratings that I found.

|  |  |
| --- | --- |
| Team Name | Power Rank |
| Birmingham | 90.05 |
| Chicago | 99.22 |
| LasVegas | 103.11 |
| LosAngeles | 106.70 |
| Memphis | 101.03 |
| New York | 97.97 |
| Orlando | 101.39 |
| SanFrancisco | 100.53 |

Los Angeles, Las Vegas, Orlando, and Memphis had the highest power rankings. These teams are different than the teams who made the playoffs. Out of the teams that are in the playoffs, I expect Los Angeles and Orlando to win. This is strickly because they have the higher power ratings.

The power ratings for the 32 nfl teams are outputted by running my program. For my algorithm I had to take into account that teams play a different amount of games. Now, instead of dividing the team's power ranking by 10 each time, I calculate how many teams they play and divide by that number. Also, if one chooses to keep the process count constant then each process will be responsible for 4 times the amount of work. A mod operation will be used to assign the teams to the correct process.

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

int SIZE=40;

int PROCESSORS=9;

int GAMES=10;

int STATS=4;

void split\_arrays(int team, int scores[SIZE][STATS], int teams\_score[GAMES][STATS])

{

int i,j,count=0;

for(i = 0; i<SIZE; i++)

{

if(scores[i][0]==team || scores[i][1]==team )

{

for(j=0; j<STATS; j++)

{

teams\_score[count][j]=scores[i][j];

}

count++;

}

}

}

void init\_array(float master\_array[PROCESSORS])

{

int i=0;

for (i=0; i<PROCESSORS; i++)

master\_array[i]=100;

}

int main(int argc,char \*argv[])

{

int score[GAMES][STATS],teams\_score[GAMES][STATS],scores[SIZE][STATS];

int i,rank,size,j;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

float power\_rank,num,sum=0;

float master\_array[PROCESSORS];

init\_array(master\_array);

int opponent=0;

FILE \*fp;

if (rank==0) //init

{

fp = fopen("xfl.dat","r");

for(i = 0; i<SIZE; i++)

{

fscanf(fp,"%d %d %d %d",&scores[i][0],&scores[i][1],&scores[i][2],&scores[i][3]);

}

fclose(fp);

for(i=1; i<PROCESSORS;i++)

{

split\_arrays(i,scores,teams\_score);

MPI\_Send(teams\_score,SIZE,MPI\_INT,i,i,MPI\_COMM\_WORLD);

}

}

else //init

{

MPI\_Recv(score,SIZE,MPI\_INT,0,rank,MPI\_COMM\_WORLD,MPI\_STATUS\_IGNORE);

for(i=0; i<GAMES; i++)

{

if (score[i][1] == rank)

{

opponent = score[i][0];

num += master\_array[opponent];

sum = sum + score[i][3] - score[i][2];

}

else

{

opponent = score[i][1];

num+=master\_array[opponent];

sum = sum + score[i][2] - score[i][3];

}

}

power\_rank=(sum+num)/GAMES;

num+=master\_array[opponent];

MPI\_Gather(&power\_rank,1,MPI\_FLOAT,master\_array,1,MPI\_FLOAT,0,MPI\_COMM\_WORLD);

}

for (j=0; j<GAMES; j++)

{

if (rank==0)

{

for(i = 0; i<PROCESSORS; i++)

{

printf("%.2f ", master\_array[i]);

}

printf("\n");

MPI\_Gather(&power\_rank,1,MPI\_FLOAT,master\_array,1,MPI\_FLOAT,0,MPI\_COMM\_WORLD);

MPI\_Bcast(master\_array,PROCESSORS,MPI\_FLOAT,0,MPI\_COMM\_WORLD);

}

else

{

sum = 0;

num = 0;

MPI\_Bcast(master\_array,PROCESSORS,MPI\_FLOAT,0,MPI\_COMM\_WORLD);

for(i=0; i<GAMES; i++)

{

if (score[i][0]==rank)

{

opponent=score[i][1];

num+=master\_array[opponent];

sum+=score[i][2]-score[i][3];

}

else

{

opponent=score[i][0];

num+=master\_array[opponent];

sum+=score[i][3]-score[i][2];

}

}

power\_rank=(num+sum)/GAMES;

MPI\_Gather(&power\_rank,1,MPI\_FLOAT,master\_array,1,MPI\_FLOAT,0,MPI\_COMM\_WORLD);

}

}

MPI\_Finalize();

return 0;

}

Code for part 2

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

int SIZE=40;

int PROCESSORS=9;

int GAMES=10;

int STATS=4;

void split\_arrays(int team, int scores[SIZE][STATS], int teams\_score[GAMES][STATS])

{

int i,j,count=0;

for(i = 0; i<SIZE; i++)

{

if(scores[i][0]==team || scores[i][1]==team )

{

for(j=0; j<STATS; j++)

{

teams\_score[count][j]=scores[i][j];

}

count++;

}

}

}

void init\_array(float master\_array[PROCESSORS])

{

int i=0;

for (i=0; i<PROCESSORS; i++)

master\_array[i]=100;

}

int main(int argc,char \*argv[])

{

int TEAM\_SIZE = (PROCESSORS-1);

FILE \*fp;

char buff[255];

char \*teams[TEAM\_SIZE];

int score[GAMES][STATS],teams\_score[GAMES][STATS],scores[SIZE][STATS];

int i,rank,size,j;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

float power\_rank,num,sum=0;

float master\_array[PROCESSORS];

init\_array(master\_array);

int opponent=0;

if (rank==0) //init

{

fp = fopen("nfl.dat","r");

for(i = 0; i<SIZE; i++)

{

fscanf(fp,"%d %d %d %d",&scores[i][0],&scores[i][1],&scores[i][2],&scores[i][3]);

}

fclose(fp);

fp = fopen("nfltms.dat","r");

for(i=0; i<(PROCESSORS-1); i++)

{

fgets(buff,255,fp);

teams[i]=buff;

}

fclose(fp);

for(i=1; i<PROCESSORS;i++)

{

split\_arrays(i,scores,teams\_score);

MPI\_Send(teams\_score,SIZE,MPI\_INT,i,i,MPI\_COMM\_WORLD);

}

}

else //init

{

MPI\_Recv(score,SIZE,MPI\_INT,0,rank,MPI\_COMM\_WORLD,MPI\_STATUS\_IGNORE);

for(i=0; i<GAMES; i++)

{

if (score[i][1] == rank)

{

opponent = score[i][0];

num += master\_array[opponent];

sum = sum + score[i][3] - score[i][2];

}

else

{

opponent = score[i][1];

num+=master\_array[opponent];

sum = sum + score[i][2] - score[i][3];

}

}

power\_rank=(sum+num)/GAMES;

num+=master\_array[opponent];

MPI\_Gather(&power\_rank,1,MPI\_FLOAT,master\_array,1,MPI\_FLOAT,0,MPI\_COMM\_WORLD);

}

for (j=0; j<GAMES; j++)

{

if (rank==0)

{

for(i = 0; i<PROCESSORS; i++)

{

printf("%.2f ", master\_array[i]);

}

printf("\n");

MPI\_Gather(&power\_rank,1,MPI\_FLOAT,master\_array,1,MPI\_FLOAT,0,MPI\_COMM\_WORLD);

MPI\_Bcast(master\_array,PROCESSORS,MPI\_FLOAT,0,MPI\_COMM\_WORLD);

}

else

{

sum = 0;

num = 0;

MPI\_Bcast(master\_array,PROCESSORS,MPI\_FLOAT,0,MPI\_COMM\_WORLD);

for(i=0; i<GAMES; i++)

{

if (score[i][0]==rank)

{

opponent=score[i][1];

num+=master\_array[opponent];

sum+=score[i][2]-score[i][3];

}

else

{

opponent=score[i][0];

num+=master\_array[opponent];

sum+=score[i][3]-score[i][2];

}

}

power\_rank=(num+sum)/GAMES;

MPI\_Gather(&power\_rank,1,MPI\_FLOAT,master\_array,1,MPI\_FLOAT,0,MPI\_COMM\_WORLD);

}

}

MPI\_Finalize();

return 0;

}