Computación de alto rendimiento

Año: 2021

TP1

“Uso básico de MPI”

Salim Taleb, Nasim A.

Docente: Garelli, Luciano

Carrera: Lic. en Bioinformática

SEMINARIO DE CALCULO PARALELO

GUIA DE TRABAJOS PRACTICOS Nº1

USO BASICO DE MPI

1) Dados dos procesadores P0 y P1 el tiempo que tarda en enviarse un mensaje desde P0 hasta P1 es una función de la longitud del mensaje Tcomm = Tcomm(n), donde n es el número de bytes en el mensaje. Si aproximamos esta relación por una recta, entonces:

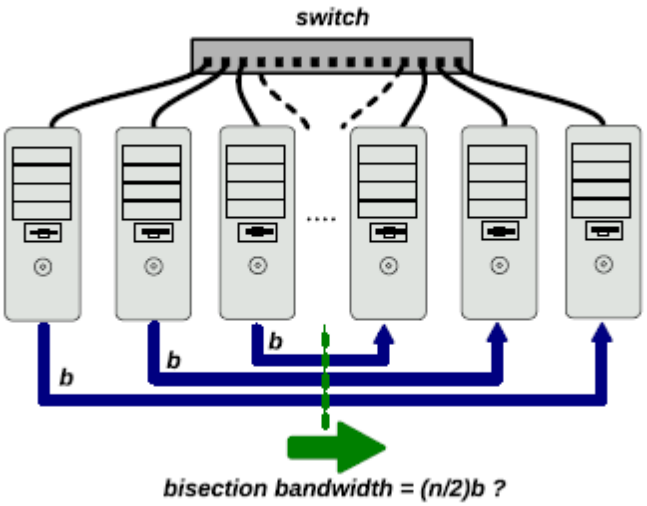
$$T_{comm} = l + \frac{n}{b}$$

donde **l** es la "latencia" y **b** es el "ancho de banda". Ambos dependen del hardware, software de red (capa de TCP/IP en Linux) y la librería de paso de mensajes usada (MPI).

Para determinar el ancho de banda y latencia de la red, escribir un programa en MPI que envíe paquetes de diferentes tamaños y realice una regresión lineal con los datos obtenidos. Obtener los parámetros para cualquier par de procesadores en el cluster. Comparar con los valores nominales de la red utilizada (por ejemplo, para Fast Ethernet: b≈100Mbit/sec, l = O(100μsec)).

2) Para un dado número de procesadores (n) realizar una matriz de transferencia a los fines de detectar posibles inhomogeneidades en la velocidad de transferencia de datos entre los procesadores. Analizar la matriz obtenida.

3) El "ancho de banda de disección" de un clúster es la velocidad con la cual se transfieren datos simultáneamente desde n/2 de los procesadores a los otros n/2 procesadores. Asumiendo que la red es switcheada y que todos los procesadores están conectados al mismo switch, la velocidad de transferencia debería ser $\left(\frac{n}{2}\right) * b$, pero podría ser que el switch tenga una máxima tasa de transferencia interna.

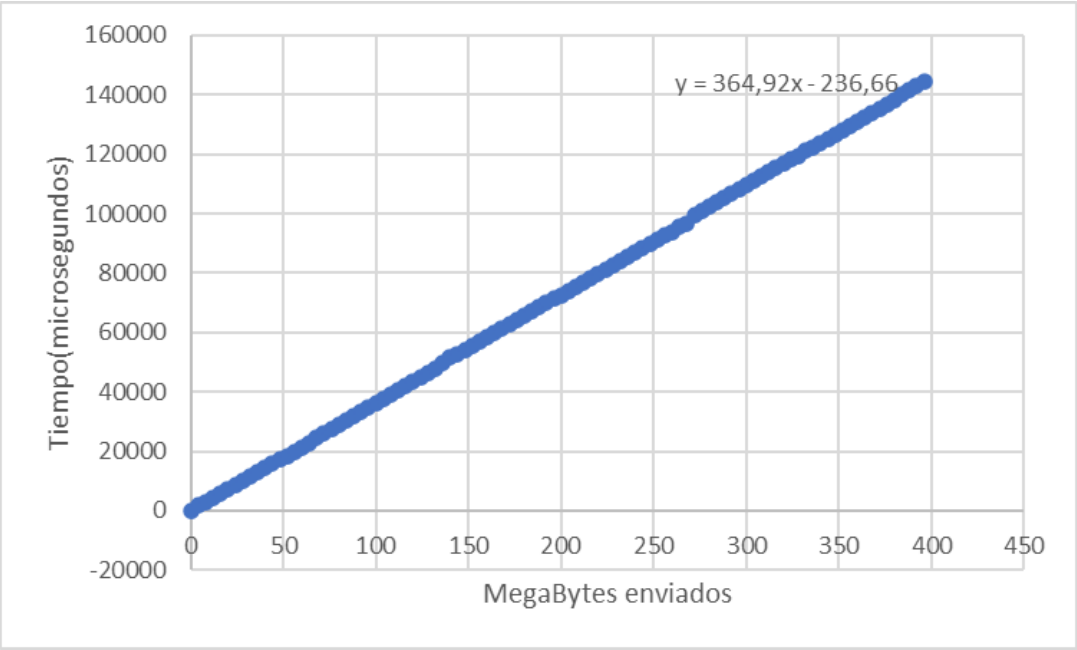


Tomar un número creciente de n procesadores, dividirlos arbitrariamente por la mitad y medir el ancho de banda para esa partición.
Comprobar si el ancho de banda de disección crece linealmente con n, es decir si existe algún límite interno para transferencia del switch.

4) Escribir un programa que dado el archivo homo_sapiens_chromosome_1.fasta, lo particione en **n** partes (donde **n** es el número total del procesadores), envíe cada una de esas porciones a los respectivos procesos y busque la cantidad de veces que aparece la base A (adenina), devolviendo al master la cantidad de veces que cada proceso contabilizó. Finalmente calcule el porcentaje de A en el cromosoma.

Desarrollo

1) Se ejecutó un código con 2 nodos en un clúster externo, enviando paquetes e incrementando el tamaño sucesivamente y a partir de los datos obtenidos se realizó una regresión lineal con los siguientes resultados:



La latencia de la red obviamente no puede ser un valor negativo, esto puede deberse a los valores elegidos para el muestreo y también a que la red posee una latencia muy cercana a 0, por simplificación se puede tomar a la latencia como el tiempo que tarda en enviar el paquete de menor tamaño, 11µs. Calculando el ancho de banda en cada dato y haciendo un promedio, este da como resultado 2755,97426 MB/s.

2) Se ejecutó un código con 30 nodos en un clúster externo, que consiste en enviar paquetes entre los distintos nodos y medir el tiempo que tarda dicho envío:

Dando como resultado la siguiente matriz:

Nodos	0	1	2	3	4	5	6	7	8	9	10
0	-	0.057375	0.053211	0.053766	0.053097	0.053609	0.053601	0.053575	0.052951	0.053525	0.053544
1	-	-	0.049804	0.049756	0.050219	0.049787	0.050299	0.050238	0.050203	0.049790	0.050240
2	-	-	-	0.049770	0.049753	0.050227	0.049756	0.050239	0.050222	0.050193	0.049757
3	-	-	-	-	0.049776	0.049785	0.050247	0.049765	0.050193	0.050191	0.050228
4	-	-	-	-	-	0.049800	0.049765	0.050209	0.049768	0.050213	0.050193
5	-	-	-	-	-	-	0.049779	0.049755	0.050199	0.049761	0.050172
6	-	-	-	-	-	-	-	0.049782	0.049772	0.050217	0.049773
7	-	-	-	-	-	-	-	-	0.049785	0.049768	0.050241
8	-	-	-	-	-	-	-	-	-	0.049787	0.049753
9	-	-	-	-	-	-	-	-	-	-	0.049784

Nodos	11	12	13	14	15	16	17	18	19	20
0	0.053587	0.053534	0.053574	0.053105	0.053474	0.053010	0.053542	0.053502	0.053290	0.056838
1	0.050206	0.050217	0.050197	0.050211	0.049778	0.050210	0.049776	0.050210	0.050190	0.050220
2	0.050207	0.050215	0.050207	0.050171	0.050189	0.049764	0.050232	0.049756	0.050209	0.050191
3	0.049761	0.050237	0.050183	0.050199	0.050195	0.050198	0.049786	0.050231	0.049761	0.050216
4	0.050406	0.049767	0.050201	0.050246	0.050358	0.050189	0.050228	0.049785	0.050370	0.049771
5	0.050180	0.050207	0.049761	0.050188	0.050184	0.050175	0.050193	0.050184	0.049778	0.050207
6	0.050215	0.050205	0.050187	0.049808	0.050227	0.050221	0.050185	0.050242	0.050212	0.049793
7	0.049773	0.050245	0.050186	0.050231	0.049774	0.050218	0.050196	0.050196	0.050236	0.050249
8	0.050938	0.049759	0.050213	0.050567	0.050216	0.049761	0.050268	0.050196	0.050354	0.050175
9	0.049772	0.050255	0.049777	0.050233	0.050219	0.050220	0.049776	0.050199	0.050194	0.050195
10	0.049778	0.049803	0.050260	0.049781	0.050194	0.050186	0.050186	0.049787	0.050183	0.050246
11	-	0.049792	0.049750	0.050193	0.049758	0.050536	0.050170	0.050179	0.049793	0.050210
12	-	-	0.049781	0.049771	0.050273	0.049768	0.050271	0.050175	0.050207	0.049766
13	-	-	-	0.049770	0.049768	0.050234	0.049770	0.050189	0.050214	0.050179
14	-	-	-	-	0.049768	0.049798	0.050208	0.049764	0.050197	0.050211
15	-	-	-	-	-	0.049783	0.049764	0.050208	0.049771	0.050206
16	-	-	-	-	-	-	0.049790	0.049769	0.050234	0.049771
17	-	-	-	-	-	-	-	0.049780	0.049768	0.050196
18	-	-	-	-	-	-	-	-	0.049954	0.049763
19	-	-	-	-	-	-	-	-	-	0.048703

Nodos	21	22	23	24	25	26	27	28	29
0	0.053660	0.053151	0.053567	0.053639	0.053596	0.053101	0.053604	0.056388	0.055330
1	0.050213	0.050225	0.049792	0.050208	0.050207	0.050218	0.049840	0.050577	0.049784
2	0.050220	0.050280	0.050203	0.049772	0.050181	0.050193	0.050248	0.049772	0.050200
3	0.050199	0.050199	0.050216	0.050189	0.049790	0.050221	0.050261	0.050248	0.049790
4	0.050238	0.050216	0.050219	0.050210	0.050201	0.049784	0.050216	0.050213	0.050215
5	0.049829	0.050184	0.050210	0.050186	0.050207	0.050218	0.049831	0.050184	0.050191
6	0.050290	0.049774	0.050225	0.050199	0.050274	0.050233	0.050256	0.049797	0.050209
7	0.049797	0.050206	0.049775	0.050231	0.050311	0.050253	0.050241	0.050303	0.049794
8	0.050193	0.049775	0.050179	0.050359	0.050205	0.050199	0.050191	0.050206	0.050243
9	0.050259	0.050323	0.049797	0.050207	0.049780	0.050402	0.050300	0.050208	0.050210
10	0.050200	0.050248	0.050187	0.049778	0.050691	0.049762	0.050266	0.050208	0.050177
11	0.050237	0.050214	0.050184	0.050202	0.049813	0.050179	0.049837	0.050174	0.050170
12	0.050194	0.050227	0.050196	0.050192	0.050305	0.049798	0.050277	0.049773	0.050198
13	0.049765	0.050190	0.050230	0.050177	0.050199	0.050200	0.049870	0.050182	0.049764
14	0.050194	0.049765	0.050197	0.050243	0.050240	0.050168	0.050272	0.049792	0.050184
15	0.050247	0.050216	0.049773	0.050201	0.050192	0.050239	0.050226	0.050195	0.049793
16	0.050198	0.050190	0.050188	0.049770	0.050204	0.050190	0.050270	0.050190	0.050227
17	0.049772	0.050195	0.050204	0.050213	0.049768	0.050222	0.050226	0.050200	0.050214
18	0.050229	0.049770	0.050179	0.050759	0.050191	0.049764	0.050183	0.050183	0.050215
19	0.048690	0.049130	0.048686	0.049127	0.049096	0.049102	0.048745	0.049102	0.049122
20	0.049759	0.049752	0.050244	0.050383	0.050161	0.050191	0.050205	0.049748	0.050250

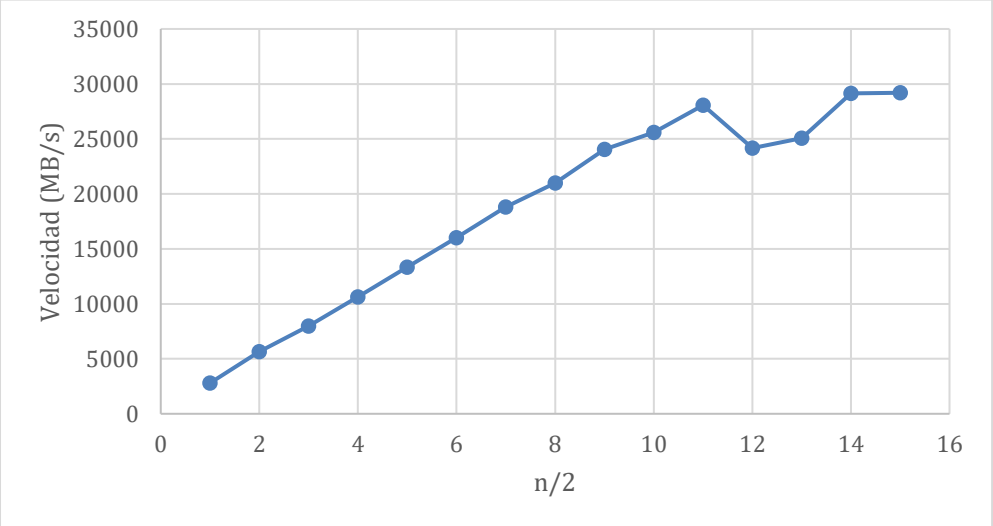
21	-	0.049755	0.049771	0.050865	0.049768	0.050173	0.050197	0.050169	0.049767
22	-	-	0.049770	0.050149	0.050266	0.049763	0.050198	0.050194	0.050232
23	-	-	-	0.049966	0.049765	0.050214	0.049767	0.050231	0.050191
24	-	-	-	-	0.049763	0.049765	0.050241	0.049747	0.050169
25	--	-	-	-	-	0.049779	0.049852	0.050190	0.049768
26	-	-	-	-	-	-	0.049836	0.049757	0.050190
27	-	-	-	-	-	-	-	0.049789	0.049775
28	-	-	-	-	-	-	-	-	0.049778
29	-	-	-	-	-	-	-	-	-

Nodo	Media de tiempo	Nodo	Media de tiempo	Nodo	Media de tiempo
0	0,05385331	10	0,05019331	20	0,05027039
1	0,05034448	11	0,05021428	21	0,05017696
2	0,05017731	12	0,05018648	22	0,05014711
3	0,05018472	13	0,05017166	23	0,05013132
4	0,050194	14	0,05018879	24	0,05024336
5	0,05017141	15	0,05017893	25	0,05017704
6	0,05019114	16	0,05018145	26	0,05013911
7	0,05019214	17	0,05017583	27	0,05018693
8	0,05022862	18	0,0501911	28	0,05027571
9	0,05019155	19	0,04980752	29	0,05021207

Mediana: 0,050187861
Se observa un pequeño retardo en las comunicaciones con el nodo 0, tiempos relativamente rápidos con el nodo 19 y resultados bastante homogéneos en el resto.

3) En esta actividad se utilizó el siguiente código en un clúster externo con 20 nodos, que consiste en enviar paquetes entre los nodos variando la cantidad de pares de nodos que envían y reciben simultáneamente:

Haciendo el tamaño del paquete (160MB) por la cantidad de envíos en simultaneo y dividiendo por el tiempo tardado, se obtiene como resultado la siguiente gráfica:



Al comienzo se aprecia que la velocidad crece linealmente a medida que se aumentan los envíos, después se observan valores extraños con 12 y 13 pero al final se ve que la velocidad de estabiliza en 30000 MB/s.

4) Para esta actividad se implementó un código que funciona con cualquier número de nodos mayor a 1 y se encarga de contar la cantidad de A para luego calcular la proporción sobre el total de bases, dando como resultado una proporción de A de 0.265611.

Códigos

Ejercicio1:

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char **argv) {
    int rank, size;
    double starttime, endtime, tiempo;
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&size);

    for (int n = 1; n <50000000; n+=500000) //Cantidad de doubles de 8
bytes a enviar
    {
            double *buff= new double[n];
            tiempo=0;
            for (int j=0;j<5;j++)
//Se envian 5 veces y se promedia el tiempo
            {
                MPI_Barrier( MPI_COMM_WORLD ) ;
                starttime = MPI_Wtime();
                if(rank==0)
                    MPI_Send(buff,n,MPI_DOUBLE,1,5,MPI_COMM_WORLD);
                if(rank==1)
                {
                        MPI_Status status;
                        MPI_Recv(buff,n,MPI_DOUBLE,MPI_ANY_SOURCE,5,
                                MPI_COMM_WORLD,&status);
                }
                MPI_Barrier( MPI_COMM_WORLD ) ;
                endtime = MPI_Wtime();
                tiempo+=endtime-starttime;
            }
            if(rank==0)printf("%d,%f;\n",n*8,tiempo/5);
            delete buff;
    }
    MPI_Finalize();
}
```

Ejercicio2:

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char **argv) {

    int rank, size;
    double starttime, endtime, tiempo=0;
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    double *buff= new double[20000000];
    MPI_Status status;

    for (int e=0;e<size;e++)
    {
        for(int r=e+1;r<size;r++)
        //Se recorre la matriz de nodos trinagular superior
        {
            for (int i=0;i<5;i++)
            //Se repiten los envios 5 veces y se promedian
            {
                MPI_Barrier( MPI_COMM_WORLD ) ;
                starttime = MPI_Wtime();

                if(rank==e)
                MPI_Send(buff,20000000,MPI_DOUBLE,r,5,MPI_COMM_WORLD);

                else if(rank==r)
                MPI_Recv(buff,20000000,MPI_DOUBLE,e,5,
                         MPI_COMM_WORLD,&status);

                MPI_Barrier( MPI_COMM_WORLD ) ;
                endtime = MPI_Wtime();
                tiempo+=endtime-starttime;
            }
            if (rank==r)printf("El tiempo de envio en la prueba desde
            %d a %d fue %f seg \n",e,r,tiempo/5);
            tiempo=0;
        }
    }
    MPI_Finalize();
}
```

Ejercicio3:

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char **argv) {

    int rank, size;
    double starttime, endtime, tiempo=0;
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    int n=1;
    while(n*2<=size)
    //Se relizara un envio a la particion siempre que exista
suficientes nodos en la corrida
    {
        double *buff= new double[20000000];
        MPI_Status status;
        for (int i=0;i<5;i++)
        //Se repiten los envios 5 veces y se promedian
        {
            MPI_Barrier( MPI_COMM_WORLD ) ;
            starttime = MPI_Wtime();

            if(rank<n)
           MPI_Send(buff,20000000,MPI_DOUBLE,rank+n,5,MPI_COMM_WORLD);

            else if(rank<n*2)
                MPI_Recv(buff,20000000,MPI_DOUBLE,
                         rank-n,5,MPI_COMM_WORLD,&status);

            MPI_Barrier( MPI_COMM_WORLD ) ;
            endtime = MPI_Wtime();
            tiempo+=endtime-starttime;
        }
        if (rank==0)printf("El tiempo de envio en la prueba con %d
        procesadores en cada particion fue de %f seg \n",n,tiempo/5);
        n++;
        tiempo=0;
    }
    MPI_Finalize();
}
```

Ejercicio4:

```
#include <stdio.h>
#include <mpi.h>
#include <fstream>
int main(int argc, char **argv)
{
    int rank, size;
    double starttime, endtime;
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    if(size>1)
    {
        int l=0;
        int n=0;
        int extra=0;
        int cantA=0;
        char*lectura;
        MPI_Status status;
        if(rank==0)
        {
            std::fstream archi;
            archi.open("homo_sapiens_chromosome_1.fasta",
                       std::fstream::in);
            archi.seekg(0,archi.end);
            l= archi.tellg();
            archi.seekg(0,archi.beg);
            archi.ignore(10000,'\n'); //Se ignora la cabecera
            l-=archi.tellg(); //Se resta la cabecera
            lectura= new char[l];
            archi.read(lectura,l);
            n=l/size;
            int extra= l-n*size;
            //Se envia a cada nodo el tamaño del arreglo que van a
              recibir y tambien el tamaño extra que recibira el ultimo
            for (int i=1;i<size;i++)
                MPI_Send(&n,1,MPI_INT,i,i,MPI_COMM_WORLD);
            MPI_Send(&extra,1,MPI_INT,size-1,20,MPI_COMM_WORLD);
        }
        else
            MPI_Recv(&n,1,MPI_INT,0,rank,MPI_COMM_WORLD,&status);
        if(rank==size-1)
        {
            MPI_Recv(&extra,1,MPI_INT,0,20,MPI_COMM_WORLD,&status);
            n+=extra;
        }
```

```
        if(rank==0)
        {
            //Se envian los datos a cada nodo
            for (int i = 1; i < size; i++)
                if(i==size-1)
                MPI_Send(lectura+n*i,n+extra,MPI_CHAR,
                         i,i,MPI_COMM_WORLD);
                else
                    MPI_Send(lectura+n*i,n,MPI_CHAR,
                             i,i,MPI_COMM_WORLD);
            //Se cuentan las A
            for (int i = 0; i < n; i++)
                if(lectura[i]=='A')
                    cantA+=1;

            int aux=0;
            for (int i = 1; i < size; i++)
            {
                MPI_Recv(&aux,1,MPI_INT,i,i,MPI_COMM_WORLD,&status);
                cantA+=aux;
            }

            printf("Cantidad de A:%d\n",cantA);

            double porA=double(cantA/double(l));
            printf("Procentaje de A:%f\n",porA);

        }
        else
            {
                lectura= new char[n];
                MPI_Recv(lectura,n,MPI_CHAR,0,rank,
                         MPI_COMM_WORLD,&status);
                //Se cuentan las A
                for (int i = 0; i < n; i++)
                    if(lectura[i]=='A')
                        cantA+=1;
                MPI_Send(&cantA,1,MPI_INT,0,rank,MPI_COMM_WORLD);
            }
    }
    MPI_Finalize();
}
```