Computación de alto rendimiento

Año: 2021

TP3

Salim Taleb, Nasim A.

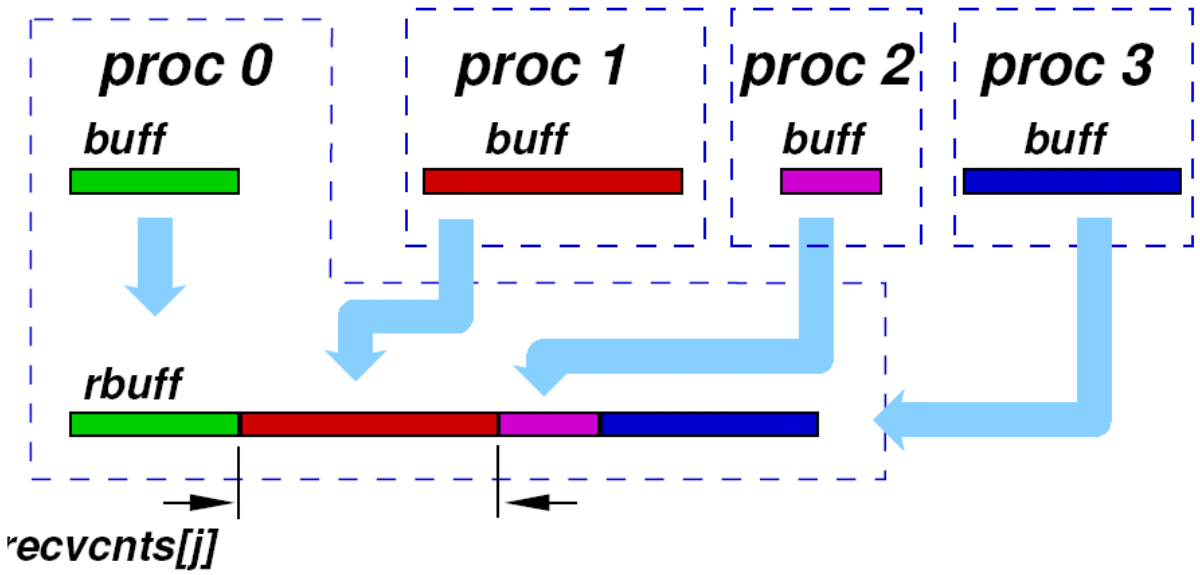
Docente: Garelli, Luciano

Carrera: Lic. en Bioinformática

SEMINARIO DE CALCULO PARALELO

GUIA DE TRABAJOS PRACTICOS Nº 3

1) Implementar una función utilizando MPI que permita mostrar el contenido de un determinado buffer, que será el resultado de concatenar varios buffers de tamaño variable (por procesador) ordenados según el proceso, como muestra la siguiente figura.
Presentar el código implementado conjuntamente con algún ejemplo de utilización de dicha función.



2) Implementar la versión paralela del Teorema de los Números Primos con distribución de carga estática.
Buscar los números primos hasta 1e7 empleando las siguientes particiones {1e3,1e4,1e5,1e6,2e6}, empleando 5 nodos. Realizar un análisis del balance de carga en los procesadores.
Obtener las distribuciones por procesador de tiempo consumido en cálculo y en comunicación/sincronización para cada partición.
Graficar el tiempo consumido en función de la partición. ¿Qué conclusiones puede sacar de la gráfica?

Desarrollo

1) El código presentado consiste en crear datos desde el “master”, que puede ser cualquier nodo, en este caso el 2, y enviarlos con distintos tamaños a los demás nodos, enviando una menor cantidad de datos a la mitad inferior de los nodos y compensando enviando el doble a la mitad superior, luego de realizar los cálculos estos se juntan en el nodo “master” y se muestran los resultados.

2) Después de ejecutar el código en un clúster externo se obtienen los siguientes datos y gráficos:

Chunk:1000

Rank	Comsynctime	Worktime
0	1,106522	0,001535
1	0,031329	1,077127
2	0,031066	1,077165
3	0,031268	1,077285
4	0,032058	1,076219

Chunk:10000

Rank	Comsynctime	Worktime
0	1,042339	0,00015
1	0,000911	1,038195
2	0,000889	1,037476
3	0,000857	1,041669
4	0,000903	1,040758

Chunk:100000

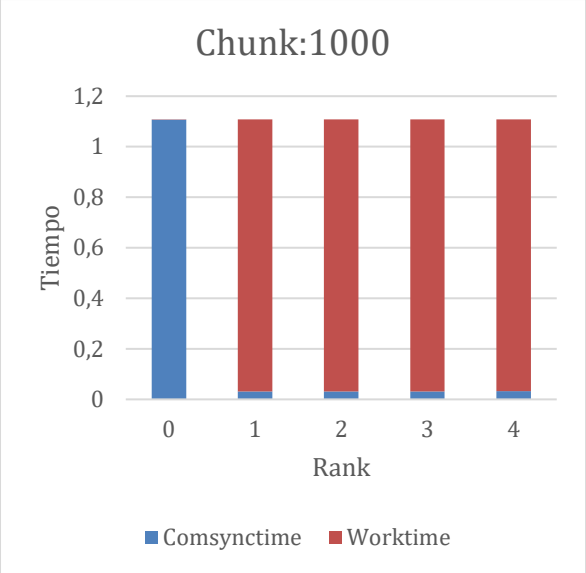
Rank	Comsynctime	Worktime
0	1,056036	0,00002
1	0,000109	1,031335
2	0,000105	1,055966
3	0,000104	1,044569
4	0,00011	1,017807

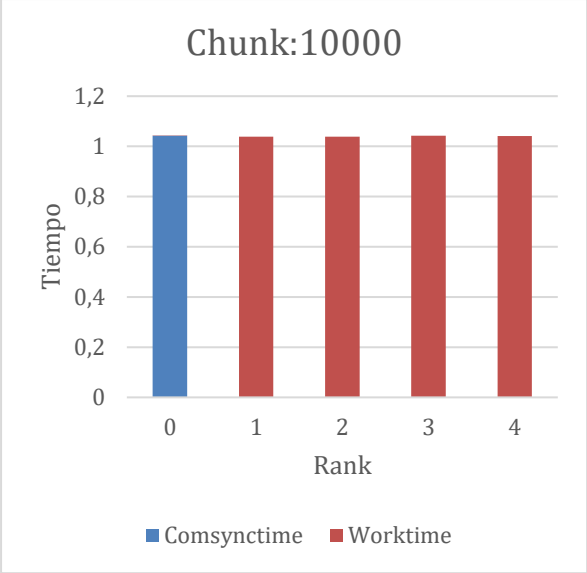
Chunk:1000000

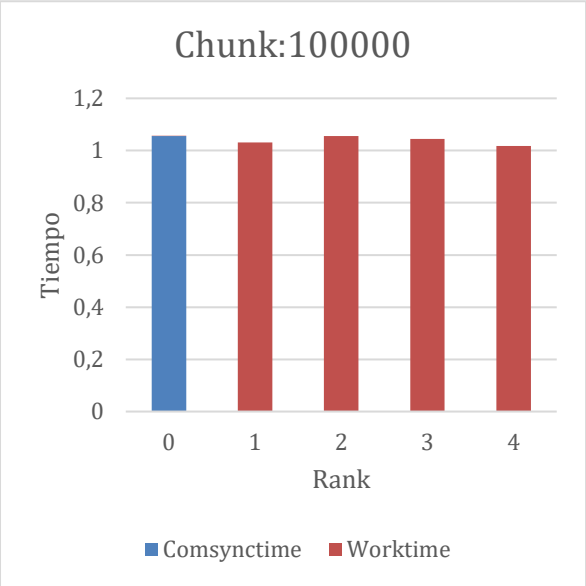
Rank	Comsynctime	Worktime
0	1,29713	0,000003
1	0,000025	0,897506
2	0,000022	0,820273
3	0,000028	1,297116
4	0,000027	1,133647

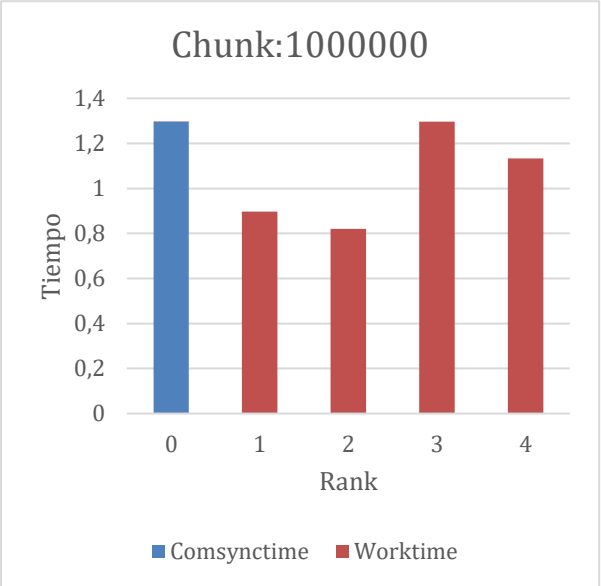
Chunk:2000000

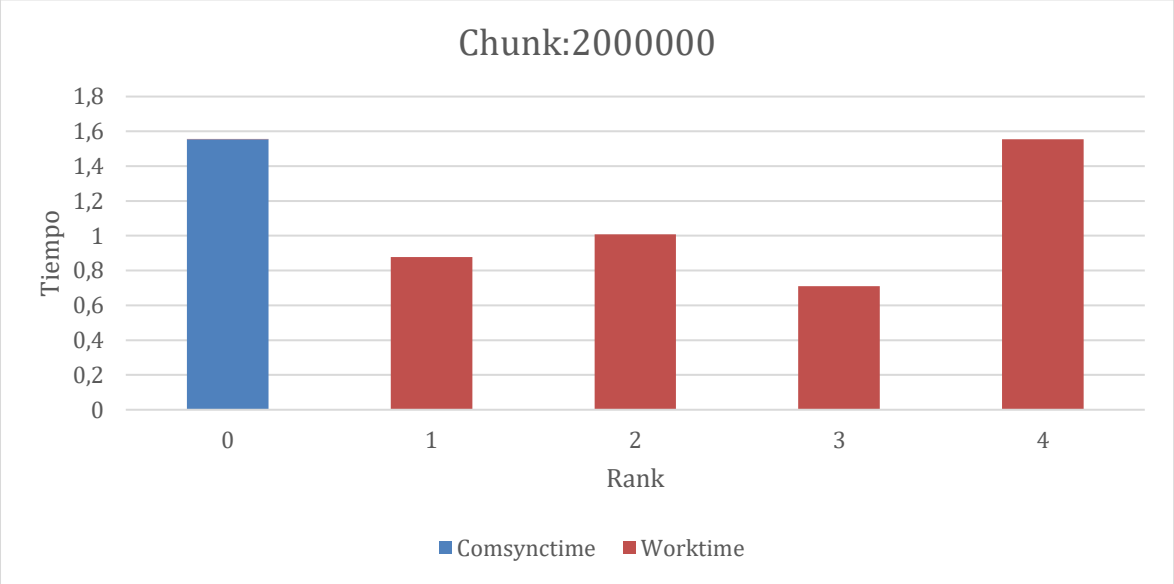
Rank	Comsynctime	Worktime
0	1,55396	0,000002
1	0,00002	0,87719
2	0,000021	1,00793
3	0,00002	0,710813
4	0,000024	1,55395


Chunk:1000
1,2
1
0,8
0,6
0,4
0,2
0
Tiempo
0
1
2
3
4
Rank
Comsynctime
Worktime


Chunk:10000
1,2
1
0,8
0,6
0,4
0,2
0
Tiempo
0
1
2
3
4
Rank
Comsynctime
Worktime


Chunk:100000
1,2
1
0,8
0,6
0,4
0,2
0
Tiempo
0
1
2
3
4
Rank
Comsynctime
Worktime


Chunk:1000000
1,4
1,2
1
0,8
0,6
0,4
0,2
0
Tiempo
0
1
2
3
4
Rank
Comsynctime
Worktime


Chunk:2000000
1,8
1,6
1,4
1,2
1
0,8
0,6
0,4
0,2
0
Tiempo
0
1
2
3
4
Rank
Comsynctime
Worktime

En base a los datos puede decirse que en el master a medida que se aumenta el tamaño del chunk se aumenta el tiempo necesario para comunicarse con los demás nodos, y lo contrario ocurre en los nodos slaves, también se reduce el tiempo de trabajo en el master. Además, en los nodos slaves al aumentar el tamaño del chunk el tiempo de sincronización y comunicación se reduce a valores despreciables, pero, los tiempos de trabajos entre los distintos nodos slaves se vuelven muy heterogéneos.

Códigos

Ejercicio1:

```
#include <stdio.h>
#include <mpi.h>

template<class T>
void mygatherv(T *sbuff, long nsend,MPI_Datatype tipo, T *rbuff, int
*nrecv, int *despl,MPI_Datatype tipo2, int recep, MPI_Comm com)
{
    MPI_Status status;
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    if(rank==recep)
    {
        for(int i=0;i<size;i++)
            if(i!=recep)

MPI_Recv(&rbuff[despl[i]],nrecv[i],tipo2,i,i,com,&status);
            else
                for(int j=0;j<nsend;j++) rbuff[despl[i]+j]=sbuff[j];
    }
    else
        MPI_Send(sbuff,nsend,tipo,recep,rank,com);
}

int main(int argc, char **argv) {

    int rank, size;
    double res, prom, starttime, endtime;
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&size);

    int tam=5003;
    int part=tam/size;
    int extra=tam-part*size;

    //Establece los tamaños de cada set de datos
    double middle=(size/2.0)-0.5;
    int n=part;
    if (rank<middle) n-=part/2;
    if (rank>middle) n+=part/2;
    if (rank==size-1) n+=extra;
    double *sbuff= new double[n];

    int *nrecv=NULL;
    int *despl=NULL;
    double *rbuff=NULL;
```

```
//Seteo de parametros de tamaños en el "master"
    if(rank==2)
    {
        rbuff= new double[tam];
        nrecv= new int[size];
        despl= new int[size];

        for (int j=0; j<size; j++)
        {
            if (j<middle) nrecv[j]=part-part/2;
            if (j==middle) nrecv[j]=part;
            if (j>middle) nrecv[j]=part+part/2;
        }

        nrecv[size-1]+=extra;
        despl[0]=0;
        for (int j=1; j<size; j++)despl[j] = despl[j-1] + nrecv[j-1];
    }

    //Setear datos en el "master"
    if (rank==2)
        for (int j=0; j<tam; j++)rbuff[j] = j;

    //Envío de los datos a procesar a cada procesador

MPI_Scatterv(rbuff,nrecv,despl,MPI_DOUBLE,sbuff,n,MPI_DOUBLE,2,MPI_COM
M_WORLD);

    //Operacion sobre los datos
    for (int j=0; j<n; j++)
        sbuff[j]*=rank;


//MPI_Gatherv(sbuff,n,MPI_DOUBLE,rbuff,nrecv,despl,MPI_DOUBLE,2,MPI_CO
MM_WORLD);

mygatherv(sbuff,n,MPI_DOUBLE,rbuff,nrecv,despl,MPI_DOUBLE,2,MPI_COMM_W
ORLD);
    if(rank==2)for(int i=0;i<tam;i++)
printf("rbuff[%d]=%f\n",i,rbuff[i]);

    if(rbuff) delete rbuff;
    if(nrecv) delete nrecv;
    if(sbuff) delete sbuff;
    if(despl) delete despl;
    MPI_Finalize();
}
```

Ejercicio2:

```
#include <stdio.h>
#include <mpi.h>
#include <math.h>

int isprime(int n)
{
    int m = int(sqrt(n));
    for (int j=2; j<=m; j++)
        if (!(n % j)) return 0;
    return 1;
}

int main(int argc, char **argv) {

    MPI_Init(&argc,&argv);
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&size);

    int N=10000000;//Contando el 0
    MPI_Status status;
    int stat[2]; // checked,primes
    double starttime,comsynctime,worktime;
    int chunk=1000;
```

```
for(int i=0;i<5;i++)
    {
        if(i==1 or i==2 or i==3) chunk*=10;
        if(i==4) chunk*=2;
        comsynctime=worktime=0;

        if (rank==0)
        {
            int first=2, checked=2, down=0, primes=0;
            while (down!=size-1)
            {
                starttime=MPI_Wtime();
                MPI_Recv(&stat,2,MPI_INT,MPI_ANY_SOURCE,MPI_ANY_TAG,
MPI_COMM_WORLD,&status);
                comsynctime+=MPI_Wtime()-starttime;
                starttime=MPI_Wtime();
                int source = status.MPI_SOURCE;
                if (stat[0])
                {
                    checked += stat[0];
                    primes += stat[1];
                }
                MPI_Send(&first,1,MPI_INT,source,0,MPI_COMM_WORLD);
                if (first<N) first += chunk;
                    else down++;
                worktime+=MPI_Wtime()-starttime;
            }
        }
        else
        {
            int start;
            stat[0]=0; stat[1]=0;
            starttime=MPI_Wtime();
            MPI_Send(stat,2,MPI_INT,0,0,MPI_COMM_WORLD);
            while(true)
            {

MPI_Recv(&start,1,MPI_INT,0,MPI_ANY_TAG,MPI_COMM_WORLD,&status);
                comsynctime+=MPI_Wtime()-starttime;
                starttime=MPI_Wtime();
                if (start>=N) break;
                int last = start + chunk;
                if (last>N) last=N;
                stat[0] = last-start ; stat[1] = 0;
                for (int n=start; n<last; n++)
                    if (isprime(n)) stat[1]++;
                worktime+=MPI_Wtime()-starttime;
                starttime=MPI_Wtime();
                MPI_Send(stat,2,MPI_INT,0,0,MPI_COMM_WORLD);
            }
        }
        MPI_Barrier(MPI_COMM_WORLD);
        if(rank==0)
printf("Chunk:%d\nRank;Comsynctime;Worktime\n",chunk);
        MPI_Barrier(MPI_COMM_WORLD);
        printf("%d;%f;%f\n",rank,comsynctime,worktime);
    }
    MPI_Finalize();
}
```