Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-4841 Introducción al Paralelismo
Trimestre Abril-Julio 2016

Práctica 5: Comunicación Colectiva con MPI Scatter, Gather, y Allgather

1. Introducción a MPI Scatter, Gather, y Allgather

El contenido de esta sección esta basado en el tutorial presentado en [1]. En esta sección se va explicar en que consisten algunas de las funciones de MPI para comunicación colectiva, llamados `MPI_Scatter` y `MPI_Gather`. También se explicará una variante de `MPI_Gather`, llamada `MPI_Allgather`.

1.1. Sobre `MPI_Scatter`

El función `MPI_Scatter` tiene como finalidad que un proceso raíz pueda enviar *partes de un arreglo* a un conjunto de procesos con los que tiene comunicación. La Figura 1 muestra como trabaja `MPI_Scatter`. Al ejecutar `MPI_Scatter` se tiene un proceso raíz que toma el arreglo y lo divide en un número de partes que son iguales al número de procesos que forman parte de la ejecución del programa MPI. Luego el proceso raíz le envía a cada uno de los procesos, incluyéndose, una parte del arreglo, siguiendo el orden de la identificación (*rank*) de cada proceso. Esto es, la primera parte del arreglo (en rojo en la Figura 1) será enviada al proceso 0, la segunda parte (en azul en la Figura 1) se le envía al proceso 1, y así sucesivamente con las todas las otras partes. El objetivo que se busca, es que cada proceso ejecute alguna tarea sobre la parte del arreglo que recibe. Aunque el proceso raíz contiene el arreglo completo, él copia en el *buffer* de todos los demás procesos la parte del arreglo que le corresponde a cada uno. A continuación se presenta el prototipo de la función `MPI_Scatter` se muestra a continuación:

```
int MPI_Scatter(void *sendbuf, int sendcnt, MPI_Datatype sendtype,
                void *recvbuf, int recvcnt, MPI_Datatype recvtype,
                int root, MPI_Comm comm)
```

donde se tiene que:

senbuf es la dirección del *buffer* de envío, es decir, el arreglo que reside en proceso raíz
sendcnt número de elementos ha ser enviados a cada proceso. Por ejemplo, si es 2 entonces el proceso 0 recibe el primer y segundo elemento del arreglo, el proceso 2 recibe el tercer y el cuarto elemento del arreglo y así sucesivamente con todos los procesos. Frecuentemente este valor corresponde al número de elementos del arreglo dividido entre el número de procesos creados
sendtype tipos de datos de los elementos ha ser enviados a cada proceso
recvbuf dirección del *buffer* de recepción, es el parámetro de salida
recvcnt número de elementos del *buffer* de recepción
recvtype tipos de datos del *buffer* de recepción
root identificador (*rank*) del proceso que envía los datos
comm comunicador

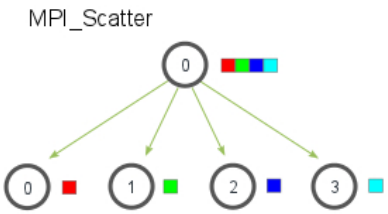

Figura 1: Ejemplo de como trabaja el función `MPI_Scatter`, el proceso 0 contiene un arreglo el cual es divido en varias partes, cada parte se envía a cada uno de los otros procesos. La figura fue tomada de [1].

1.2. Sobre `MPI_Gather`

La función `MPI_Gather` toma los resultados de varios procesos y los reúne para que los maneje un único proceso. La Figura 2 ilustra el funcionamiento de `MPI_Gather`.

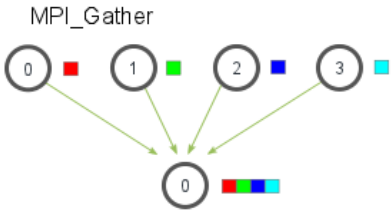

Figura 2: Ejemplo de como trabaja el función `MPI_Gather`, los resultados de todos los procesos son recolectados por el proceso 0 en un arreglo. La figura fue tomada de [1]

Se tiene que `MPI_Gather` toma los elementos que son retornados por todos los procesos, y los reúne para entregárselos a un proceso raíz que los almacena en arreglo. Los elementos en el arreglo son ordenados por el identificador (*rank*) de los procesos que enviaron los datos. El prototipo de función de `MPI_Gather`, es igual a la de la función `MPI_Gather` y se muestra a continuación:

```
int MPI_Gather(void *sendbuf, int sendcnt, MPI_Datatype sendtype,
               void *recvbuf, int recvcnt, MPI_Datatype recvtype,
               int root, MPI_Comm comm)
```

donde se tiene que:

senbuf es la dirección del *buffer* de envío de cada uno de los procesos
sendcnt número de elementos del *buffer* de envío
sendtype tipos de datos de los elementos ha ser enviados por cada proceso
recvbuf dirección del *buffer* de recepción, es el parámetro de salida y es manejado por el proceso raíz.
recvcnt número de elementos que se espera recibir por proceso
recvtype tipos de datos del *buffer* de recepción
root identificador (*rank*) del proceso que recibe los datos
comm comunicador

1.3. Ejemplo computando el promedio de los números en un arreglo

En la dirección `http://chimo.ldc.usb.ve/practica5.tar.xz` puede descargar el código de la práctica que está compuesto de los siguientes archivos:

- `avg.c` programa que usa `MPI_Scatter` y `MPI_Gather`,
- `all_avg.c` programa que usa `MPI_Scatter` y `MPI_Allgather`,
- `makefile` para compilar los archivos en C,
- `maquinas.txt` conjunto de nodos del *cluster*.

Para ilustrar el uso de `MPI_Scatter` y `MPI_Gather`, se muestra como ejemplo un programa que computa el promedio de todos los elementos que están contenidos en un arreglo. El programa que contiene el ejemplo se llama `avg.c` y en él se distinguen los siguientes pasos:

1. El proceso raíz genera un arreglo de números reales lleno con números aleatorios.
2. Dado un número de procesos, se le envía a cada proceso una misma cantidad de números (*Scatter*).
3. Cada proceso computa el promedio de números que le fueron enviados.
4. El proceso raíz recoge (*Gather*) todos los promedios que les envían todos los procesos. El proceso raíz computa el promedio de estos números para obtener el resultado final.

Estas cuatro partes se observan en el siguiente el Listado 1.

```
if (world_rank == 0) {
  rand_nums = create_rand_nums(num_elements_per_proc * world_size);
}

// For each process, create a buffer that will hold a subset of the entire
// array
float *sub_rand_nums = (float *)malloc(sizeof(float) *
  num_elements_per_proc);
assert(sub_rand_nums != NULL);

// Scatter the random numbers from the root process to all processes in
// the MPI world
MPI_Scatter(rand_nums, num_elements_per_proc, MPI_FLOAT, sub_rand_nums,
            num_elements_per_proc, MPI_FLOAT, 0, MPI_COMM_WORLD);

// Compute the average of your subset
float sub_avg = compute_avg(sub_rand_nums, num_elements_per_proc);
printf("I am the processs %d and my average is:%f\n", world_rank, sub_avg);

// Gather all partial averages down to the root process
float *sub_avgs = NULL;
if (world_rank == 0) {
  sub_avgs = (float *)malloc(sizeof(float) * world_size);
  assert(sub_avgs != NULL);
}
MPI_Gather(&sub_avg, 1, MPI_FLOAT, sub_avgs, 1, MPI_FLOAT, 0,
  MPI_COMM_WORLD);

// Now that we have all of the partial averages on the root, compute the
// total average of all numbers. Since we are assuming each process
  computed
// an average across an equal amount of elements, this computation will
// produce the correct answer.
if (world_rank == 0) {
  float avg = compute_avg(sub_avgs, world_size);
```

Listado 1: Fragmento del código de avg.c.

Al ejecutar el archivo `makefile`, se obtiene el ejecutable `avg` del archivo `avg.c`. El programa `avg.c` recibe como entrada el número de elementos por proceso. Si se ejecuta el programa `avg`, se obtiene un resultado semejante al que se muestra a continuación:

```
$ mpiexec --disable-hostname-propagation --machinefile maquinas.txt -n 4 ./avg 700
I am the processs 0 and my average is: 0.496193
I am the processs 2 and my average is: 0.489221
I am the processs 3 and my average is: 0.513167
I am the processs 1 and my average is: 0.496983
Avg of all elements is 0.498891
Avg computed across original data is 0.498891
```

1.4. Sobre `MPI_Allgather`

`MPI_Allgather` es una función que hace que todos los procesos reciban los elementos enviados por todos los procesos. La Figura 3 muestra el funcionamiento de `MPI_Allgather`.

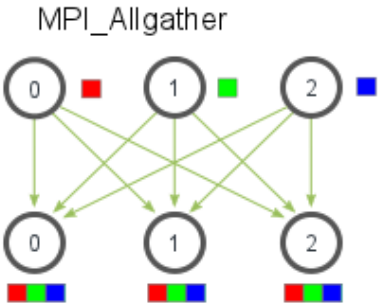

Figura 3: Ejemplo de como trabaja el función `MPI_Allgather`, los resultados de todos los procesos son recolectados por todos los procesos en un arreglo. La figura fue tomada de [1]

En `MPI_Allgather` los elementos son recolectados siguiendo el orden de identificación (*rank*) de los procesos. La función prototipo de `MPI_Allgather` es casi idéntica a la de `MPI_Gather`, con la única diferencia de que no hay un proceso raíz.

```
int MPI_Allgather(void *sendbuf, int sendcnt, MPI_Datatype sendtype,
                  void *recvbuf, int recvcnt, MPI_Datatype recvtype,
                  MPI_Comm comm)
```

El programa `all_avg.c` modifica el programa `avg.c` para que en lugar de `MPI_Gather` use `MPI_Allgather`. El principal diferencia entre los dos códigos se muestra en el Listado 2.

```
// Gather all partial averages down to all the processes
float *sub_avgs = (float *)malloc(sizeof(float) * world_size);
MPI_Allgather(&sub_avg, 1, MPI_FLOAT, sub_avgs,1, MPI_FLOAT, MPI_COMM_WORLD);

// Compute the total average of all numbers.
float avg = compute_avg(sub_avgs, world_size);
```

Listado 2: Fragmento del código de all_avg.c.

Al compilar el programa `all_avg.c` se obtiene el programa `all_avg`, que al ejecutarlo con 4 procesos y con 700 elementos a ser estudiados, se obtiene un resultado como el siguiente:

```
$ mpiexec --disable-hostname-propagation --machinefile maquinas.txt -n 4 ./all_avg 700
Avg of all elements from proc 0 is 0.504305
Avg of all elements from proc 2 is 0.504305
Avg of all elements from proc 1 is 0.504305
Avg of all elements from proc 3 is 0.504305
```

2. Actividad a realizar

Se quiere que implemente programa similar al presentado en `avg.c` y `all_avg.c`, pero en este caso el objetivo será encontrar el valor mínimo, máximo y promedio de los elementos que están contenidos en una matriz de números reales. El programa debe crear una matriz de dimensión $N \times N$, donde N es el número de procesos a ejecutar. La matriz debe ser llenada con números reales generadas aleatoriamente. Luego el proceso raíz usando `MPI_Scatter` le envía cada una de las N filas, a cada uno de los N procesos. Cada proceso debe encontrar el valor mínimo, máximo y promedio de cada fila. Una vez procesada cada fila, cada proceso haciendo uso de `MPI_Gather`, envía al proceso raíz un arreglo de tres elementos. El arreglo contiene en la posición primera, el valor mínimo de la fila, en la posición segunda el valor máximo de la fila, y finalmente en la posición tercera el valor promedio. Para terminar, el proceso raíz toma el contenido de todos los arreglos y en encuentra los valores mínimo, el máximo y promedio, entre los valores mínimos, máximos y promedios de cada fila.

Su programa se debe llamar `practica5.c` y debe mostrar por la salida estándar los siguientes elementos:

1. La matriz de elementos generada por el proceso raíz.
2. Cada proceso debe mostrar su identificador, junto con el valor mínimo, máximo y promedio de su fila.
3. El proceso raíz debe mostrar el mínimo, máximo y promedio computados con los valores que le enviaron.
4. El proceso raíz debe mostrar el mínimo, máximo y promedio computados sobre la matriz original de elementos, como manera de comprobar los resultados de la computación en paralelo.

Por ejemplo, si al compilar `practica5.c` se obtiene un ejecutable llamado `practica5`, al ejecutarlo debe mostrar una salida como la siguiente:

```
$ mpiexec --disable-hostname-propagation --machinefile maquinas.txt -n 4 ./practica5

Matrix:
0.566 0.611 0.506 0.180
0.817 0.183 0.585 0.422
0.025 0.316 0.061 0.084
0.977 0.978 0.874 0.053

I am the processs 0 with row 0 - min 0.180 - max 0.611 - avg 0.466
I am the processs 1 with row 1 - min 0.183 - max 0.817 - avg 0.502
I am the processs 2 with row 2 - min 0.025 - max 0.316 - avg 0.122
I am the processs 3 with row 3 - min 0.053 - max 0.978 - avg 0.720

Final results: min 0.025 - max 0.978 - avg 0.452

Results using the matrix: min 0.025 - max 0.978 - avg 0.452
```

Se recomienda implementar en el lenguaje C la matriz por medio de un arreglo de $N \times N$ elementos.

Debe entregar los códigos fuentes de solución y un informe en donde explique el diseño de solución y muestre ejemplos de los resultados obtenidos con diferentes corridas de su programa.

Referencias

[1] Wes Kendall. Mpi scatter, gather, and allgather. `http://mpitutorial.com/tutorials/mpi-scatter-gather-and-allgather/`, 2016.

Guillermo Palma / gvpalma@usb.ve / Junio 2016