

## Puzzle, Parte 1

Mikel Dalmau

8 de Marzo de 2016

### 1 Comunicaciones colectivas

Las comunicaciones colectivas consisten en distribuir la carga de trabajo que conlleva la comunicación de forma equitativa o eficiente entre los procesos.

#### 1.1 Comunicación en forma de árbol

Esta forma de comunicación, aunque inicialmente pueda parecer que no mejora demasiado, ya que la mitad de los nodos realizan las mismas comunicaciones que realizarían punto a punto (esto es, cuando todos los nodos envían su valor a un único nodo), mejora considerablemente reduciendo la cantidad de recepciones y sumas que tiene que realizar el nodo líder.

En el ejemplo de la imagen, el nodo 0 pasa de realizar  $n-1$  recepciones y sumas a realizar 3 recepciones y sumas ( $\log_d n$  en el caso general, siendo  $d$  el grado de árbol). De esta forma la carga de trabajo de los nodos crece logarítmicamente con el número de procesadores, la mejora es notable frente al crecimiento lineal de la comunicación punto a punto.

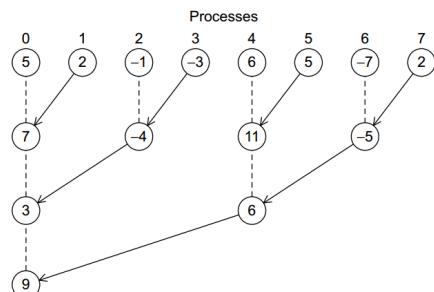


FIGURE 3.6

A tree-structured global sum

El problema de este modelo de comunicación es la dificultad de programarlo, existen incontables maneras distintas de hacerlo pero no sabríamos cual es la mejor, para que tipos y tamaños de problemas cual es la óptima.

#### 1.2 MPI\_Reduce

Esta función implementa la comunicación colectiva vista en el apartado anterior, las decisiones de implementación tomadas en base a los criterios de los desarrolladores de MPI.

##### NAME

`MPI_Reduce` – Reduces values on all processes to a single value

##### SYNOPSIS

```
int MPI_Reduce(MPICH2_CONST void *sendbuf,
void *recvbuf, int count, MPI_Datatype
datatype, MPI_Op op, int root, MPI_Comm comm)
```

##### INPUT PARAMETERS

- `sendbuf` – address of send buffer (choice)
- `count` – number of elements in send buffer (integer)
- `datatype` – data type of elements of send buffer (handle)
- `op` – reduce operation (handle)
- `root` – rank of root process (integer)
- `comm` – communicator (handle)

##### OUTPUT PARAMETER

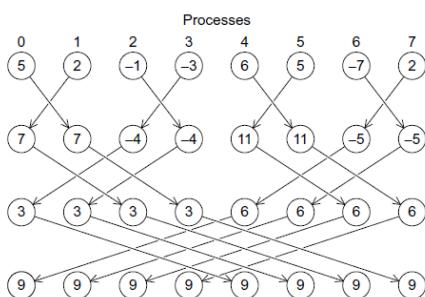
- `recvbufm` – address of receive buffer (choice, significant only at root )

Es de especial interés el parámetro *op* que define el tipo de operación a realizar. Existen los siguientes tipos de operaciones:

Operation Value	Meaning
MPI_MAX	Máximo
MPI_MIN	Mínimo
MPI_SUM	Suma
MPI_PROD	Producto
MPI_AND	And lógico
MPI_BAND	And binario
MPI_OR	Or lógico
MPI_BOR	Or binario
MPI_LXOR	Or exclusivo lógico
MPI_BXOR	Or exclusivo binario
MPI_MAXLOC	Máximo y su dirección
MPI_MINLOC	Mínimo y su dirección

### 1.2.1 MPI\_ALLreduce

Esta función es una variación de **MPI\_Reduce** en la que todos los nodos involucrados en la comunicación reciben el resultado. Los parámetros son idénticos con la excepción de que ahora sobra el parámetro *root* ya que todos recibirán el mensaje.

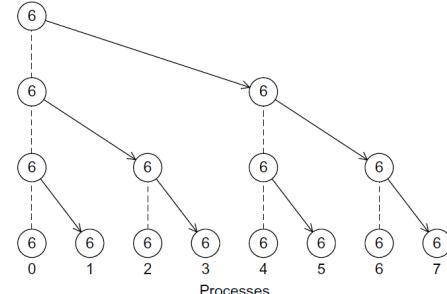


**FIGURE 3.9**  
 A butterfly-structured global sum

La siguiente imagen muestra la estructura de comunicación utilizada por **MPI\_ALLreduce** que tiene forma de mariposa.

### 1.3 MPI\_Bcast

En el caso de desear transmitir un mensaje a todos los nodos, se utiliza la misma estructura de árbol que hemos visto en el apartado anterior pero a la inversa.



**FIGURE 3.10**  
 A tree-structured broadcast

#### NAME

**MPI\_Bcast** – Broadcasts a message from the process with rank "root" to all other processes of the communicator

#### SYNOPSIS

```
int MPI_Bcast( void *buffer, int count,
               MPI_Datatype datatype,
               int root,
               MPI_Comm comm )
```

#### INPUT/OUTPUT PARAMETER

buffer – starting address of buffer  
(choice)

#### INPUT PARAMETERS

count – number of entries in buffer (integer)  
 datatype – data type of buffer (handle)  
 root – rank of broadcast root (integer)  
 comm – communicator (handle)

### 1.4 MPI\_Scatter

### 1.5 MPI\_Gather

### 1.6 MPI\_Allgather

## 2 Ejercicios, primera parte del puzzle.

### 2.1 P1.1

Hay que repartir un vector de N elementos entre npr procesos. Completa el programa serie *P11-distribute0.c*, para que genere el tamaño de cada trozo del vector y el desplazamiento desde el origen del vector al comienzo de cada trozo, en estos dos casos:

- a. los posibles restos se añaden al último trozo
- b. los posibles restos se añaden uno a uno a diferentes trozos

### 2.1.1 a.

Inicialmente calculo Nloc y remainder.

```
//Compute Nloc and remainder
Nloc = floor((double)N/(double)npr);
remainder = N - Nloc*npr;
```

Este caso es sencillo y se resuelve con el siguiente bucle y las asignaciones finales.

```
//We distribute the work among the
//processes
for(i=0; i<npr-1; i++){
  size[i] = Nloc;
  shift[i] = i*Nloc;
}
//Finally we charge the last process
//with the remainder
size[npr-1] = Nloc + remainder;
shift[npr-1] = (npr-1)*Nloc;
```

### 2.1.2 b.

En este segundo caso he comenzado distribuyendo la carga del resto entre los primeros procesadores, luego, las cargas distintas entre procesos no permiten el cálculo de shift usado anteriormente, por lo que tomo las referencias de tamaño y shift calculadas en la anterior iteración para calcular el shift, sabemos donde empezamos porque sabemos dónde termina el anterior.

```
//Value assignment to first process
size2[0] = Nloc;
```

```

shift2[0] = 0;

//Distribution of the remainder among
//the first processes
i = 0;
while(remainder){
  size2[i] += 1;
  remainder -= 1;
  i++;
}

//Distribute the rest of the vector
//among the processes
for(i=1; i<npr; i++){
  size2[i] += Nloc;
  shift2[i] = shift2[(i-1)] + size2[(i
-1)];
}
```

## 2.2 P1.2

El programa *P12-inteser.c* calcula el valor de una integral mediante el conocido método de sumar las áreas de n trapecios bajo la curva que representa una función. A mayor valor de n, más preciso el resultado.

Completa el programa MPI *P12-inteser.c* para realizar esa misma función entre P procesos, utilizando funciones de comunicación colectiva. Compara el resultado con el de la versión serie.

## Bibliografía

- [1] Pacheco P.:*An Introduction to Parallel Programming*. Morgan Kaufmann, 2011. Capítulo 3, apartado 4.