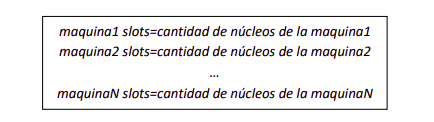
**Programación en pasaje de mensajes / Programación híbrida**

**Enunciado**

**1. Resuelva los ejercicios 2 y 3 de la práctica 4.**

mpirun –np P -machinefile archivoMaquinas nombreBinario arg1 arg2 … argN donde: - P representa el número de procesos a generar. - archivoMaquinas es un archivo de texto que lista las máquinas a utilizar y su cantidad de núcleos. Debe tener el siguiente formato:

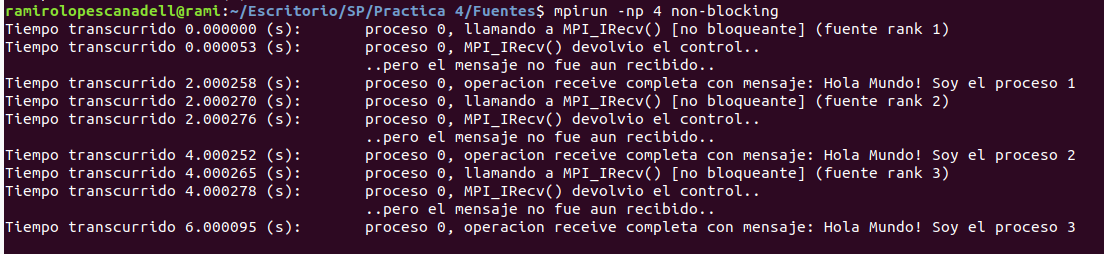
****

Nota: para ejecutar en el cluster de la cátedra, siga las instrucciones detalladas en el instructivo.

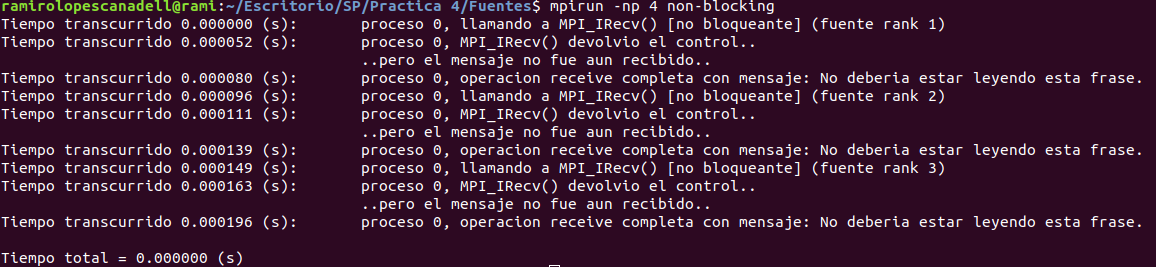
**2. Los códigos blocking.c y non-blocking.c siguen el patrón master-worker, donde los procesos worker le envían un mensaje de texto al master empleando operaciones de comunicación bloqueantes y no bloqueantes, respectivamente.**

* **Compile y ejecute ambos códigos usando P={4,8,16} (no importa que el número de núcleos sea menor que la cantidad de procesos). ¿Cuál de los dos retorna antes el control?**El algoritmo no bloqueante retorna antes el control pero luego debe frenarse en la instruccion **MPI\_Wait()** para esperar a que se complete el request. El tiempo que transcurre hasta que se completa el request y finaliza el **MPI\_WAIT()** es ligeramente mayor al tiempo que transcurre en el algoritmo bloqueante entre que ejecuta el **receive()** hasta que se le retorna el control.
* **En el caso de la versión no bloqueante, ¿qué sucede si se elimina la operación MPI\_Wait() (línea 52)? ¿Se imprimen correctamente los mensajes enviados? ¿Por qué?**

Antes:



Despues:



Se puede ver que no se imprimen correctamente los mensajes.

El buffer por defecto tiene el mensaje “no deberia estar leyendo esta frase”.

La sentencia **MPI\_WAIT(&request, &status)** hace que el proceso espere hasta que se cumpla el request (es decir hasta que se ejecuta la siguiente sentencia: *MPI\_Irecv(message, BUFSIZ, MPI\_CHAR, source, tag, MPI\_COMM\_WORLD,* ***&request****);* Por eso al quitar esa sentencia el proceso master itera sobre el for sin esperar a que se ejecute la sentencia **Irecv** y por lo tanto cuando los procesos realicen el **Isend()** el proceso master ya va a estar finalizado.

Por lo tanto, no se estaria cumpliendo el objetivo, donde los workers resuelven una tarea, y una vez resuelta le comunican al master

**3. Los códigos blocking-ring.c y non-blocking-ring.c comunican a los procesos en forma de anillo empleando operaciones bloqueantes y no bloqueantes, respectivamente. Compile y ejecute ambos códigos empleando P={4,8,16} (no importa que el número de núcleos sea menor que la cantidad de procesos) y N={10000000, 20000000, 40000000, …}. ¿Cuál de los dos algoritmos requiere menos tiempo de comunicación? ¿Por qué?**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **10.000.000** | **20.000.000** | **40.000.000** |
| **4** |  | **0.493307** |  |
| **8** |  |  |  |
| **16** |  |  |  |

El algoritmo **no-bloqueante** tarda menos tiempo.

Ambos algoritmos calculan por cada proceso de quien recibirá el mensaje y a quien le enviará el mensaje:

* El **proceso 0** recibe del **último proceso** y le envía al **proceso 1**
* El **último proceso** recibe del proceso **ante-ultimo** y envía al **proceso 0**
* El resto de los proceso reciben de **ID-1** y envían a **ID+1**

El algoritmo **bloqueante** tiene el orden de **receive()** y del **send()** distinto dependiendo de que proceso es: todos los procesos menos el último ejecutan primero el **receive()** y después el **send()**. Por lo tanto todos los procesos van a estar esperando en el **receive() ,**excepto uno, y se va a disminuir el paralelismo, debido a que va a haber a lo sumo un proceso en **send()** y un proceso en **receive().**Esto se hace de esta manera para evitar el deadlock, que en algoritmos con mensaje bloqueantes es más fácil que ocurra.

Por otro lado el algoritmo **no-bloqueante** va a ejecutar, para todos los procesos, primero el **Isend ()**  y luego el **Ireceive** debido que, al no bloquearse en los mensajes, no puede generarse deadlock en ese aspecto. Luego los procesos esperan a que se envíen y reciban los mensajes con las sentencias ***MPI\_WAIT (&send\_request,&status);*** y ***MPI\_Wait(&recv\_request,&status);***  respectivamente.

El algoritmo **no-bloqueante** tarda menos tiempo porque varios procesos al mismo tiempo pueden ejecutar las sentencias **Isend()** y/o **Ireceive()**, por lo tanto aumenta el paralelismo. En el caso del bloqueante en una instancia de tiempo va a haber a lo sumo un proceso ejecutando un **send()** y un proceso ejecutando un **receive().**

**2. Dada la siguiente expresión:**

**𝐷 = ((𝑚𝑎𝑥𝐴 ⋅ 𝑚𝑎𝑥𝐵 ⋅ 𝑚𝑎𝑥𝐶 )− (𝑚𝑖𝑛𝐴 ⋅ 𝑚𝑖𝑛𝐵 ⋅ 𝑚𝑖𝑛𝐶) / (𝑎𝑣𝑔𝐴 ⋅ 𝑎𝑣𝑔𝐵 ⋅ 𝑎𝑣𝑔𝐶 )) \* ABC**