COMPUTACIÓN DE ALTAS PRESTACIONES – PRÁCTICA

Semestre Septiembre 2019

- Presentación y objetivos
- Enunciado de la práctica
- Criterios de evaluación
- Formato de entrega
- Fecha de entrega

Presentación y objetivos

Objetivos

Esta práctica tiene como objetivos introducir los conceptos básicos del modelo de programación MPI, y de su entorno de ejecución. Para la realización se pondrán en práctica varios de los conceptos presentados en esta asignatura.

Presentación de la práctica

Hay que entregar los ficheros con el código fuente realizado y un documento con las respuestas a las preguntas formuladas y los comentarios que consideréis. Toda la codificación se realiza exclusivamente en lenguaje C con las extensiones asociadas a MPI.

Restricciones del entorno

Aunque el desarrollo de la práctica es mucho más interesante usando docenas de computadores, se asume que inicialmente tendréis acceso a un número bastante reducido de computadores con varios núcleos por nodo.

Material para la realización de la práctica

En los servidores de la UOC tenéis el software necesario para ejecutar MPI.

De todos modos, si queréis hacer vuestro desarrollo y pruebas en vuestro sistema local tendréis que tener instalado algún software que implemente MPI. Podéis utilizar el que mejor os parezca pero os aconsejamos OpenMPI o MPICH2, que es una distribución libre, portable y muy popular. Podéis descargarla y encontrar más información sobre la instalación/utilización en la siguiente dirección:

http://www.mcs.anl.gov/research/projects/mpich2/

En cualquier caso se espera que realicéis vuestra investigación y si es necesario que se debata en el foro de la asignatura.

Se espera que se produzca debate en el foro de la asignatura para profundizar en el uso de los modelos de programación y los sistemas paralelos proporcionados.

Enunciado de la práctica

1. Entorno de ejecución MPI

La primera parte de esta práctica consiste en familiarizarse con el entorno y ser capaz de compilar y ejecutar programas MPI.

Utilizaremos el siguiente programa MPI que implementa el típico programa que devuelve "hello world" (hello.c):

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

int main(int argc, char **argv)
{
  MPI_Init(&argc, &argv);

  printf("Hello World!\n");

  MPI_Finalize();
}
```

podéis apreciar que hay que incluir "mpi.h" y que es imprescindible empezar los programas MPI con MPI_Init y al final terminarlos con MPI_Finalize. Estas llamadas se encargan de trabajar con el runtime de MPI para la creación y destrucción de procesos MPI.

A continuación se presentan los pasos necesarios para compilar y ejecutar un simple "hello world" utilizando MPI:

- Compilar el programa hello.c con mpicc (más detalles en la documentación).
- Ejecutar el programa MPI hello world mediante SGE. Un script de ejemplo para la ejecución del programa hello utilizando 8 procesos MPI se puede encontrar a continuación:

```
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N hello
#$ -o hello.out.$JOB_ID
#$ -e hello.err.$JOB_ID
#$ -pe orte 8

mpirun -np 8 ./hello
```

La opción -pe orte 8 le indica a SGE que se necesitan 8 núcleos (los cuales se asignarán en más de un nodo en el clúster de la UOC). La llamada mpirun se encarga de llamar al runtime de MPI para realizar la ejecución del programa MPI. La opción -np 8 le indica al runtime de MPI que utilice 8 procesos para la ejecución del programa MPI y al final indicamos el nombre del binario del programa. Podéis estudiar otras opciones de mpirun.

Preguntas:

1. **¿Cuál es el resultado de la ejecución del programa MPI? ¿Por qué?**
2. **Explicad qué ocurre si utilizáis la opción -np 4 en cambio de -np 8 en el mpirun**

2. Procesos MPI

A continuación, introducimos un programa MPI que incluye llamadas básicas que se pueden encontrar en cualquier programa MPI. Nótese que la variable "rank" es básica en la ejecución de programas MPI ya que permite identificar el número de proceso dentro de un comunicador (en este caso MPI_COMM_WORLD). Por favor, consultad los apuntes para más detalles.

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

int main(int argc, char **argv)
{
  int rank, numprocs;
  char hostname[256];

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

  gethostname(hostname,255);

  printf("Hello world! I am process number %d of %d MPI processes on host %s\n", rank,
numprocs, hostname);

  MPI_Finalize();

  return 0;
}
```

Preguntas:

3. ¿Cuál es el resultado de la ejecución del programa MPI? ¿Por qué?

3. Paso de mensajes punto a punto

En este apartado os pedimos el primer ejercicio de programación. Os facilitamos otro ejemplo de aplicación MPI llamado "hellompi.c" junto con el anunciado de la PEC. Este programa está diseñado para ser ejecutado sólo con 2 procesos MPI y básicamente se encarga de hacer de paso de mensajes entre los dos procesos MPI.

Se pide que realicéis una variante del programa "hellompi.c" en el cual cada uno de los procesos MPI haga el envío de un mensaje MPI al resto de procesos.

Un ejemplo de salida con 3 procesos MPI se muestra a continuación:

```
Proc #1 sending message to Proc #0
Proc #1 sending message to Proc #2
Proc #1 received message from Proc #0
Proc #1 received message from Proc #2
Proc #0 sending message to Proc #1
Proc #0 sending message to Proc #2
Proc #0 received message from Proc #1
Proc #0 received message from Proc #2
Proc #2 sending message to Proc #0
Proc #2 sending message to Proc #1
Proc #2 received message from Proc #0
Proc #2 received message from Proc #1
```

Preguntas:

4. Proporcionad el código de vuestro programa.

4. Orden de escritura de la salida estándar

En este apartado se pide que implementéis un programa MPI que realice paso de mensajes entre procesos MPI en forma de anillo, es decir, que cada proceso MPI haga el envío de un mensaje al siguiente rank y reciba uno del rank anterior (excepto el primero y el último que han de cerrar el anillo).

Así pues, dados N procesos MPI, se espera que el proceso MPI con rank 0 envíe de un mensaje al proceso con rank 1, el proceso con rank 1 reciba el mensaje el proceso con rank 0 y envíe un mensaje al proceso con rank 2, y así hasta llegar al proceso con rank N-1 que recibirá un mensaje del proceso N-2 y enviará un mensaje al proceso con rank 0.

Podéis usar como referencia la siguiente línea como ejemplo para la salida de vuestro programa.

```
Proc 2: received message from proc 1 sending to 3
```

Preguntas:

5. Proporcionad el código de vuestro programa.

6. ¿En qué orden aparecen los mensajes de salida? ¿Por qué?

5. Programación mediante MPI

En este apartado se pide que implementéis un pequeño problema mediante MPI y también que proporcionéis una versión paralela de un código secuencial proporcionado.

Preguntas:

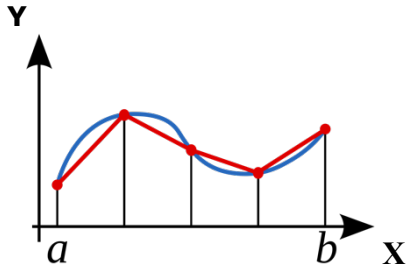
7. Proporcionad tres (3) programes paralelos simples para sumar una cantidad de números aleatorios. Cada proceso primero debería generar y sumar localmente la cantidad asignada de números aleatorios. Para encontrar la suma suma total entre todos los procesos, hay tres opciones:

Opción 7.a: que un proceso sea el "master" y que cada proceso use MPI_Send per enviar su suma local al master.

Opción 7.b: permitir que un proceso sea el "master" de manera que recoja de la resta de procesos mediante MPI_Gather.

Opción 7.c: que un proceso sea el "master" y uséis MPI_Reduce.

El fichero "p8.c" proporciona el código secuencial que se puede usar para aproximar el área del gráfico correspondiente a la función y=f(x), teniendo en cuenta dos líneas verticales (a y b) y el eje de las x. La siguiente figura proporciona una idea básica del problema que es una forma de aproximar la integral definida.



En el caso de la figura se usan n=4 segmentos para aproximar la integral.

Si los puntos de un segmento son x_i y x_{i+1}, entonces el tamaño de este fragmento es $h=x_{i+1}-x_i$. Si el tamaño (vertical) de los segmentos son $f(x_i)$ y $f(x_{i+1})$, entonces el área del segmento es: $h/2\ [f(x_i)+f(x_{i+1})]$.

Como que usaremos n fragmentos que tendrán todos el mismo tamaño, podemos saber que $h=(b-a)/n$ y, por lo tanto, la suma de las áreas de los segmentos es $h[f(x_0)/2 + f(x_1)+f(x_2)+...+f(x_{n-1})+f(x_n)/2]$.

El código proporcionado (p8.c) utiliza la función "fragment" que puede ser útil para vuestra implementación.

Preguntas:

8. Proporcionad una versión paralela del código que resuelve la ecuación integral mediante MPI. Explicad vuestras decisiones/observaciones.

9. Evaluad la ejecución de vuestro programa MPI en relación a la versión secuencial. En particular, se pide que estudiéis (y proporcionéis gráficos) el speedup respecto el número de núcleos utilizados y el número n de fragmentos.

6. Herramientas de evaluación de rendimiento (TAU)

Este apartado de la PEC consiste en realizar un estudio del solver implementado.

Para la realización de este estudio utilizaremos el paquete de análisis de rendimiento TAU (https://www.cs.uoregon.edu/research/tau/home.php). Este paquete se puede utilizar a nivel de usuario y se encuentra en la siguiente ubicación en el clúster de la UOC

```
/export/apps/tau/
```

También tendréis que añadir los siguiente a $PATH:

```
export PATH=$PATH:/export/apps/tau/x86_64/bin
```

El paquete de TAU incluye tanto las herramientas de instrumentación y de análisis como el visualizador de trazas jumpshot. Los pasos básicos para la utilización de TAU y las herramientas relacionadas se proporcionan a continuación:

- **Uso recomendado de TAU para la PEC (con instrumentación dinámica)**

En vuestro script SGE tendréis que usar lo siguiente:

```
export TAU_TRACE=1
(activates tracing)

mpirun -np NUM_PROCS tau_exec ./YOUR_MPI_PROGRAM
(dynamic instrumentation - recommended)
```

- **Análisis con pprof (podéis explorar por vuestra cuenta paraprof si queréis profundizar)**

Sólo hay que ejecutar el comando pprof en el directorio donde se haya ejecutado el programa MPI y donde habrán generado los ficheros "profile. *. *. *". Un ejemplo básico se muestra a continuación:

```
[ivan@eimtarqso p4]$ pprof
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        3,000        3,015           1          12    3015921 .TAU application
  0.3           10           10           1           0      10235 MPI_Init()
  0.1            3            3           1           0       3450 MPI_Finalize()
  0.0        0.863        0.863           2           0        432 MPI_Barrier()
  0.0        0.545        0.545           3           0        182 MPI_Send()
  0.0        0.018        0.018           3           0          6 MPI_Recv()
  0.0        0.001        0.001           1           0          1 MPI_Comm_rank()
  0.0        0.001        0.001           1           0          1 MPI_Comm_size()

NODE 1;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        3,000        3,015           1          12    3015923 .TAU application
  0.3           10           10           1           0      10190 MPI_Init()
  0.1            3            3           1           0       3470 MPI_Finalize()
```

Podéis encontrar las opciones de este comando a continuación:

```
usage: pprof [-c|-b|-m|-t|-e|-i|-v] [-r] [-s] [-n num] [-f filename] [-p] [-l] [-d] [node
numbers]
 -a : Show all location information available
 -c : Sort according to number of Calls
 -b : Sort according to number of suBroutines called by a function
 -m : Sort according to Milliseconds (exclusive time total)
 -t : Sort according to Total milliseconds (inclusive time total)  (default)
 -e : Sort according to Exclusive time per call (msec/call)
 -i : Sort according to Inclusive time per call (total msec/call)
 -v : Sort according to Standard Deviation (excl usec)
 -r : Reverse sorting order
 -s : print only Summary profile information
 -n <num> : print only first <num> number of functions
 -f filename : specify full path and Filename without node ids
 -p : suPpress conversion to hh:mm:ss:mmm format
 -l : List all functions and exit
 -d : Dump output format (for tau_reduce) [node numbers] : prints only info about all
contexts/threads of given node numbers
```

- **Visualización de trazas con jumpshot**

Para poder entender mejor los posibles problemas relacionados con el paso de mensajes en MPI suele ser muy útil visualizar las trazas de la ejecución de los programas y hacer un estudio con herramientas que trabajen con estas trazas. Hay que decir que el estudio de rendimiento mediante estas técnicas es un arte y se requiere experiencia para hacer análisis en profundidad. En esta PEC se pide que observéis las características más básicas y que os familiaricéis un poco con estas herramientas.

El paquete de TAU incluye la herramienta jumpshot que se puede utilizar en esta PEC mediante los siguientes pasos:

```
tau_treemerge.pl
tau2slog2 tau.trc tau.edf –o tau.slog2
jumpshot tau.slog2
```

Si utilizáis jumpshot directamente en el clúster de la UOC el tiempo de respuesta será muy grande y es posible que sea impracticable. Por ello se recomienda que descarguéis la herramienta en vuestro entorno local. La herramienta se encuentra en el siguiente directorio del paquete de TAU:

```
/export/apps/tau/x86_64/bin/tau2slog2
```

Podéis encontrar el fichero `jumpshot.jar`

```
/export/apps/tau/x86_64/lib/jumpshot.jar
```

Sólo hay que tener instalado el entorno de ejecución de Java en vuestro sistema (ya sea Linux, mac o Windows) para utilizar la herramienta.

Preguntas:

10. Haced un breve estudio del comportamiento de vuestra implementación del ejercicio 8 (resolución de ecuación integral) mediante las herramientas introducidas. Podéis incluir resúmenes estadísticos, capturas de pantalla y los comentarios que consideréis apropiados.

7. Herramientas de evaluación de rendimiento (Extrae/Paraver)

Este apartado consiste en realizar un estudio de rendimiento de algunas de las aplicaciones de los NAS Parallel Benchmarks (NPB) - https://www.nas.nasa.gov/publications/npb.html. En concreto se pide que estudiéis los benchmarks EP, BT y CG (clase S). La motivación de este ejercicio es entender cómo realizar estudios de rendimiento de aplicaciones no conocidas a priori y de las cuales no se disponga del código fuente.

Los binarios están en los siguientes directorios en el nodo de login:

```
/export/apps/aca/benchmarks/NPB3.2/NPB3.2-MPI/bin/ep.S.16
/export/apps/aca/benchmarks/NPB3.2/NPB3.2-MPI/bin/bt.S.16
/export/apps/aca/benchmarks/NPB3.2/NPB3.2-MPI/bin/cg.S.16
```

Estos son de la clase S que es la más pequeña de los NPB y están compilados para ser ejecutados con 16 procesos MPI.

En este caso usaremos la herramienta Extrae (https://tools.bsc.es/extrae) desarrollada por el Barcelona Supercomputing Center.

Para usar Extrae tendréis que definir las siguientes variables y ejecutar lo siguiente:

```
export EXTRAE_HOME=/export/apps/extrae
cp /export/apps/extrae/share/example/MPI/extrae.xml YOUR_DIRECTORY
source /export/apps/extrae/etc/extrae.sh
export EXTRAE_CONFIG_FILE=YOUR_DIRECTORY/extrae.xml
export LD_PRELOAD=/export/apps/extrae/lib/libmpitrace.so
```

y hacer las ejecuciones como habitualmente, por ejemplo:

```
mpirun -np 16 ./filename
```

Como resultado de la ejecución de la aplicación instrumentada se obtiene una traza compuesta de tres ficheros: `filename.prv`, `filename.row` i `filename.pcf`.

Para visualizar y analizar la traza obtenida usaremos la herramienta Paraver (https://tools.bsc.es/paraver), la cual está disponible para múltiples plataformas y tendréis que instalar en vuestro terminal.

Preguntas:

11. Realizad un breve estudio comparativo del comportamiento de los benchmarks NPB EP, BT y CG usando Extrae/Paraver. Podéis incluir resúmenes estadísticos, capturas de pantalla y los comentarios que consideréis apropiados.

Criterios de evaluación

Se valorará el correcto uso del modelo de programación MPI. También se tendrá en cuenta la utilización de las herramientas de evaluación de rendimiento y los comentarios proporcionados.

Formato de entrega

Se creará un fichero `tar` con todos los ficheros fuente de la práctica.

Con el siguiente comando:

```
$ tar cvf todo.tar fichero1 fichero2 ...
```

se creará un fichero "todo.tar" donde se tendrán almacenados todos los ficheros "fichero1", "fichero2", ...

Para listar la información de un fichero `tar` se puede utilizar el siguiente comando:

```
$ tar tvf todo.tar
```

Para extraer la información de un fichero `tar` se puede utilizar:

```
$ tar xvf todo.tar
```

El nombre del fichero tendrá el formado siguiente: "Apellido1Apellido2PEC3.tar". Los apellidos se escribirán sin acentos. Por ejemplo la estudiante Marta Garrido León utilizará el nombre de fichero siguiente: `GarridoLeonPEC3.tar`

Fecha de entrega

Domingo 1 de Diciembre de 2019.