Asignación de Prácticas Número 10
Programación Concurrente y de Tiempo Real

Kevin J. Valle Gómez

Departamento de Ingeniería Informática
Universidad de Cádiz

2020

Contenido

Objetivos de la práctica

- Aplicar los conocimientos sobre MPJ-Express.
  - Desarrollar los programas que se indican.
  - Someter los programas a un análisis de rendimiento.
- Desarrollar soluciones para sistemas donde no existe memoria compartida. Se estructurará la comunicación y sincronización entre hebras mediante el envío de mensajes.

MPJ-Express

- Repasa el concepto de *Paso de Mensajes* visto en clase de teoría.
- Implementación de MPI para Java.
- Permite ejecutar aplicaciones paralelas en *clusters* o redes de computadores.

MPJ-Express (II)

- Tiene dos modos de configuración:
  - *Multicore configuration:* para ejecutar aplicaciones en ordenadores domésticos haciendo uso de la memoria compartida y procesadores multinúcleo. Usada en el campo de la enseñanza y recomendada para realizar pruebas.
  - *Cluster configuration:* para ejecutar aplicaciones de manera distribuida.

Instalación

1. `windowsguide.pdf`
2. `linuxguide.pdf`
3. Sigue las instrucciones de descarga e instalación en cada caso.

Instalación (II)

Comprueba que la instalación es correcta descargando el fichero `puntoAPuntoSincrono.java` disponible en la carpeta de trabajo de esta asignación de prácticas.

- **Compilación:** javac -cp .;%MPJ_HOME%/lib/mpj.jar puntoAPuntoSincrono.java
- **Ejecución:** mpjrun.bat -np 2 puntoAPuntoSincrono

Un repaso a las funciones básicas

- Conviene tener muy claras las diapositivas 19 y 20 del tema 6 ;-)

- MPI.Init() para iniciar la ejecución paralela.
- MPI.COMM_WORLD.Size() para determinar el número de procesos en ejecución paralela.
- MPI.COMM_WORLD.Rank() para que cada proceso obtenga su identificador dentro de la colección de procesos que forman la aplicación.
- MPI.Finalize() para terminar la ejecución del programa.

...y a los tipos de datos

TIPO DE DATO DE MPJ-Express	TIPO DE DATO JAVA
MPI.BYTE	byte
MPI.CHAR	char
MPI.SHORT	short
MPI.BOOLEAN	boolean
MPI.INT	int
MPI.LONG	long
MPI.FLOAT	float
MPI.DOUBLE	double
MPI.OBJECT	Object

¡No olvidemos los emisores y receptores!

- void Send(bufer, offset, datatype, destination, tag)
- Status Recv(bufer, offset, datatype, source, tag)

Ejercicio 1: buscando números primos

- Volvemos al conocido problema de números primos: debemos encontrar los números primos en un rango dado.
- Sobradamente conocido ;-)
  - Asignación de prácticas número 5.
  - Códigos de ejemplo del tema 4.
- **Parte 1:** resuelve el problema con tareas `Runnable` y un ejecutor de tamaño fijo.
- **Parte 2:** resuelve el problema con paso de mensajes y MPJ-Express.

Ejercicio 1 - Parte 1: Solución *tradicional*

- Rescata `primosParalelos.java` de la asignación de prácticas número 5 (por ejemplo).
- Para la segunda parte de esta asignación de prácticas, deberás tener control sobre el número de tareas paralelas.
- `int nTareas = Runtime.getRuntime().availableProcessors();`

Ejercicio 1 - Parte 2: Solución con paso de mensajes y MPJ-Express

- Resuelve el mismo problema utilizando MPJ-Express.
- Recomendación: parte del ejemplo `distribInteg.java` proporcionado en la carpeta de esta asignación de prácticas.
  - Estructura muy similar a este problema.

Ejercicio 2: análisis de rendimiento

- Selecciona un rango de números naturales fijo: 10^8
- Toma los tiempos **para cada solución**
  - Tareas Runnable y ejecutor de tamaño fijo.
  - Paso de mensajes y MPJ-Express.
- Traza curvas de tiempo en función del número de tareas.
- Analiza los resultados: ¿qué opción es mejor? ¿por qué?