Cómputo paralelo

TEMA 13

Esquema

Ideas clave

13.1. ¿Cómo estudiar este tema?

Para estudiar este tema lee las **Ideas clave** que encontrarás a continuación. Además, deberás estdiar el apéndice 9 (páginas 711-719) del siguiente libro, disponible en el aula virtual en virtud del artículo 32.4 de la Ley de Propiedad Intelectual:

Dean, J. S. y Dean, R. H. (2009). *Introducción a la programación con Java*. México: Mc Graw Hill.

Para comprobar si has comprendido los conceptos puedes realizar el test de autoevaluación del tema.

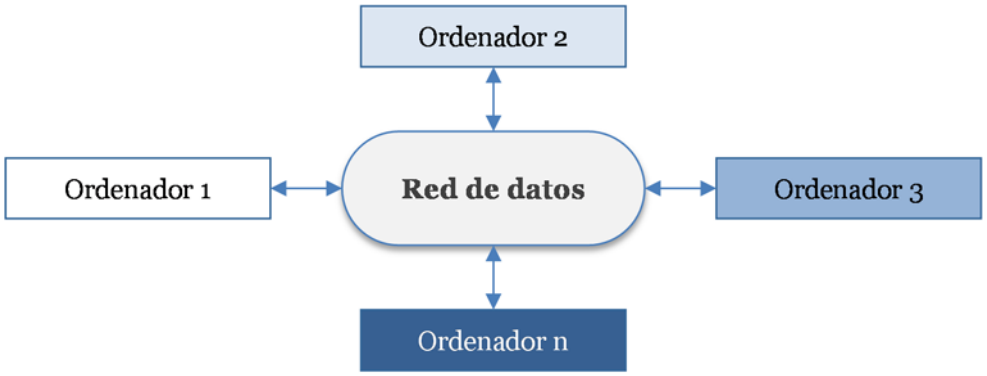
En este tema vamos a ver una introducción al cómputo paralelo usando Java.

13.2. Introducción al cómputo paralelo

En la actualidad existen diferentes ordenadores comerciales con capacidad **multi-core** como se estudió en el anterior tema de alto rendimiento. Este tipo de arquitecturas permite un **procesamiento en paralelo** para ejecutar un determinado programa. La computación paralela o procesamiento en paralelo puede ser realizada de dos maneras: (1) a través de un ordenador con capacidad multi-core o (2) utilizando varios ordenadores conectados en red (**cómputo distribuido**) para resolver un proceso informático. Para poder realizar el último cómputo en paralelo se necesita de un software de procesamiento distribuido.

Con una arquitectura paralela es posible resolver problemas muy complejos, con un procesamiento de cómputo masivo en donde un monoprocesador sería incapaz de resolver. Gracias a esta nueva técnica, varias aplicaciones que requieren procesar datos demasiado grandes han sido beneficiadas, tales como en la predicción del comportamiento atmosférico, análisis aerodinámicos en la industria área, simulaciones moleculares entre muchas otras.

Por otro lado, también existen diferentes tipos de hardware basados en GPU, FPGA entre otros que permiten procesar información de forma paralela para poder realizar diferentes tareas con hardware dedicado.



Arquitectura distribuida.

13.3. Uso de la memoria en una arquitectura paralela

El acceso de la memoria es un aspecto importante, ya que permitirá que un procesador pueda acceder a ella. Podemos mencionar **tres tipos de memoria**: memoria compartida, memoria distribuida y memoria compartida distribuida.

» **Memoria compartida**: En esta arquitectura varios o múltiples programas pueden acceder a la memoria. Una característica importante es que los programas pueden ser ejecutados en un solo procesador o en procesadores separados.

» **Memoria distribuida**: En esta arquitectura hay dos tipos:

1. Un multicore conectado a un solo bus donde varios procesadores acceden a la memoria.
2. Varios ordenadores conectados a través de una red de alta velocidad.

» **Memoria compartida distribuida**: En esta arquitectura, tanto el *software* como el *hardware* están implementados para que cada nodo de un cluster tenga acceso a una memoria compartida y grande. Cada nodo tiene una memoria privada limitada que será ampliada cuando el cluster tenga acceso.

13.4. Paralelismo en procesos internos

En varios procesos internos de código pueden existir procesos en los que algunas variables son independientes de otros procesos dependientes, es decir, que si hay procesos independientes que no afecten al proceso dependiente, podría calcularse de forma paralela. Por ejemplo, se puede analizar las siguientes ecuaciones, donde hay ecuaciones independientes que se pueden calcular de manera paralela:

$$X1=X2+X3+X4$$
$$X5=X6+X7*X8$$
$$Z=X1+X5$$

En este ejemplo las ecuaciones dos primeras ecuaciones se pueden calcular de forma simultánea, ya sea en un sistema multicore o distribuido.

Por otro lado, en Java podemos crear **programas multihilo** para varios procesos, pero no se considera como la mejor solución para lograr paralelismo debido a problemas de rendimiento.

```
class Ejemplohilo implements Runnable
{
	private Thread a;
	private String Nombrehilo;
	Ejemplohilo (String nombre){
	Nombrehilo = nombre;
	System.out.println("Hilo creado" +Nombrehilo);
	}

	public void run(){
		System.out.println("Hilo Funcionando" +Nombrehilo);


	}
	public void start () {
		System.out.println("Iniciando " +Nombrehilo );
		if (a == null) {
		   a = new Thread (this, Nombrehilo);
		   a.start ();
		}
	}
	}
```



```
public class Main {
        public static void main(String args[]) {
             Ejemplohilo P1 = new Ejemplohilo( "Proceso 1");
             P1.start();

             Ejemplohilo P2 = new Ejemplohilo( "Proceso 2");
             P2.start();

             Ejemplohilo P3 = new Ejemplohilo( "Proceso 3");
             P3.start();

          }
}
```

Ejemplo Multihilo.

En la versión de Java 5 se introdujo el nuevo **paquete de concurrencia** (java.util.concurrency) para mejorar algunos problemas al usar multihilo. En este nuevo paquete se puede hacer uso de **ejecutores** (Executors). Estos ayudan a separar el proceso o tarea creada y ayudan a gestionar el subproceso. Las principales funciones de esta utilidad son:

```
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
```

La interfaz de ExcutorService es una implementación de la interfaz de Executor y usa `Thread Pools`, donde se pueden gestionar varios hilos de trabajo. La forma de trabajo del Executor es la siguiente:

```
ExecutorService miEjecutor = Executors.newFixedThreadPool (n);
miEjecutor.execute(miInstancia1Funcionando);
```

Una tarea es subida a la piscina de hilos *n* (`Thread pool`) a través de una cola interna de trabajo, en la que se asigna a un hilo de trabajo una vez que está completa su tarea previa, invocando el método de `newFixedThreadPool()` de la interfaz Executor, se puede crear un grupo de subprocesos con un tamaño fijo, asegurándose de que siempre hay un número fijo de subprocesos disponibles para procesar las tareas.

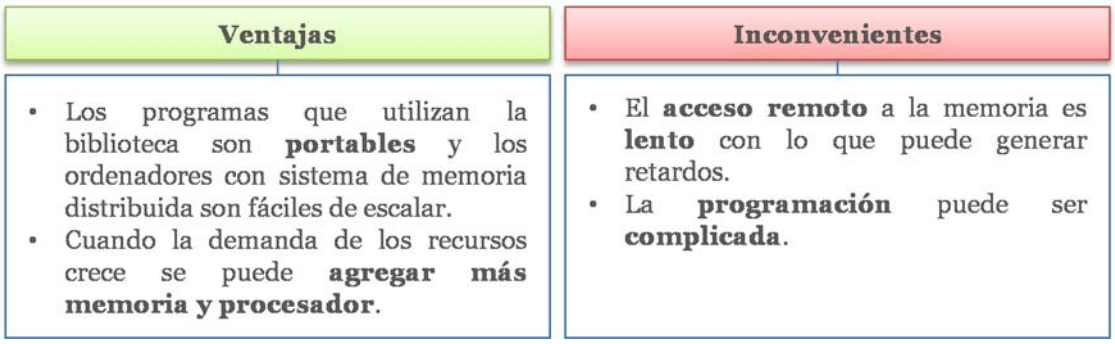
13.5. Paralelismo en procesos usando librerías externas

En el caso de Java, la librería MPI (del inglés Massage Passing Interface) se utiliza en una arquitectura de cómputo en paralelo. MPI es una interfaz de paso de mensajes y se utiliza como **protocolo de comunicación entre ordenadores**. Con esta librería se pueden usar nodos que ejecutan un programa de un sistema de memoria distribuida.

La librería MPI es de gran utilidad, ya que se puede usar en diversos lenguajes de programación como Java y Python, entre otros. Esto permite que varios ordenadores puedan comunicarse a través de la red de comunicación.

Uno de los objetivos con esta implementación es lograr portabilidad al obtener un lenguaje de programación que permita ejecutar de manera transparente aplicaciones sobre sistemas heterogéneos.

El uso de esta librería tiene algunas ventajas y desventajas.



A continuación, se presenta un ejemplo de cómo implementar la librería MPI express (0.44) para Java (http://mpj-express.org).

```
import mpi.MPI;
public class multicore {
	public static void main (String[] args) throws Exception {
		MPI.Init(args);
		int rank = MPI.COMM_WORLD.Rank();
		int size = MPI.COMM_WORLD.Size();
		System.out.println("Numero de procesadores: <"+rank+"> de
<"+size+">");
	}

}
```


La salida es:

```
MPJ Express (0.44) is started in the multicore configuration
Numero de procesadores: <2> de <10>
Numero de procesadores: <1> de <10>
Numero de procesadores: <5> de <10>
Numero de procesadores: <4> de <10>
Numero de procesadores: <3> de <10>
Numero de procesadores: <6> de <10>
Numero de procesadores: <0> de <10>
Numero de procesadores: <7> de <10>
Numero de procesadores: <8> de <10>
Numero de procesadores: <9> de <10>
```

Para este ejemplo se han asignado 10 CPUs. El número de procesadores se puede agregar con el siguiente argumento en las configuraciones del debuger:

```
-jar ${MPJ_HOME}\lib\starter.jar -np 10
```

Se puede concluir que en el entorno Java se pueden usar dos métodos para realizar un cómputo paralelo. Es decir, mediante una **combinación de multihilo** con las nuevas funciones de la utilidad concurrente (java.util.concurrent) y por medio de **librerías externas** como MPI.

Lo + recomendado

No dejes de leer...

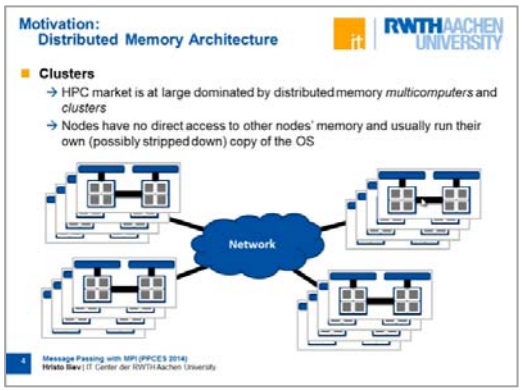
Sistemas de memoria compartida

En este recurso podrás aumentar la información sobre los sistemas de memoria compartida y la clasificación de redes de interconexión.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
http://sistemasdememoriacompartida.blogspot.com.es/2013/11/sistemas-de-memoria-compartida.html

No dejes de ver...

Una introducción a la programación MPI



Este vídeo presenta una introducción a la programación MPI.

Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:
https://www.youtube.com/watch?v=LBgx_S5ougk

+ Información

A fondo

Principios y Campos de Aplicación en CUDA

Rivera, I. O. y Vargas-Lombardo, M. (2012). *Principios y Campos de Aplicación en CUDA Programación paralela y sus potencialidades. Nexo Revista Científica*, Vol. 25, núm. 2, pp. 39-46.

La siguiente monografía es un compendio de información sobre la tecnología CUDA (Compute Unifed Device Architecture, que es un modelo de programación y una arquitectura de cálculo) de procesamiento paralelo en la GPU, para uso computacional, en general, descripción de la misma, como también algunas aplicaciones.

Accede al artículo a través de la Biblioteca Virtual de UNIR.

Test

1. En el computó paralelo, ¿qué arquitecturas son más usadas?

A. Monoprocesador.

B. Varios procesadores conectados en serie.

C. Procesadores multicore y ordenadores conectados en una red de comunicación.

D. CUDA.

2. Tipos de memoria usada en una arquitectura paralela:

A. Memoria RAM.

B. Memoria de acceso remoto.

C. Memoria ampliada.

D. Memoria compartida y distribuida.

3. En una serie de instrucciones, ¿qué operaciones pueden realizarse de manera paralela?

A. Variables independientes.

B. Variables dependientes.

C. Funciones de tiempo.

D. Vectores y arreglos.

4. Al usar un método multihilo, ¿qué paquete de Java puede ser utilizado para realizar procesos paralelos?

A. `Multithread`.

B. `java.util.concurrency`.

C. MPI

D. `ThreadPool`.

5. La interfaz de Executor usa `TheadPool` para:

A. Dar prioridades

B. Inicializar la interfaz

C. Declarar una variable paralela

D. Gestionar varios hilos de trabajo

6. ¿Qué método asegura que siempre hay un número fijo de subprocesos disponibles para procesar las tareas?

A. `ExecutorService`

B. `newFixedThreadPool`

C. `java.util.concurrent.Future`

D. `ThreadPool`

7. ¿Qué otro método se puede usar en Java para realizar tareas paralelas en un sistema distribuido?

A. Uso de librerías externas para computo paralelo compatibles con el lenguaje de programación.

B. Computo paralelo a través de una red de comunicación.

C. Métodos multihilo.

D. Uso de paquetes.

8. ¿Qué significa MPI?

A. Interfaz de programación multiuso.

B. Modo de Prueba y Error.

C. Método de Programación Iterativa.

D. Interfaz de paso de mensajes.

9. ¿Qué ventaja tiene el uso de MPI?

A. Mayor procesamiento.

B. La biblioteca es portable y los ordenadores con sistema de memoria distribuida son fáciles de escalar.

C. Interfaz fácil.

D. Comunicación entre ordenadores.

10. ¿Qué desventaja tiene el uso de MPI?

A. Fácil programación.

B. Comunicación lenta.

C. Difícil programación.

D. Escaso soporte.

