Uso de ForkJoinPool en java

`ForkJoinPool` es una implementación avanzada del modelo de hilos en Java, introducida en **Java 7** como parte del paquete `java.util.concurrent`. Está diseñado para dividir tareas grandes en subtareas más pequeñas, ejecutarlas de forma paralela y luego unir los resultados. Esta técnica sigue el patrón de **"divide y vencerás"** (divide-and-conquer). El uso de `ForkJoinPool` es ideal para tareas recursivas como algoritmos de búsqueda, procesamiento de grandes volúmenes de datos, o cálculos que puedan dividirse en varias partes independientes.

La clase `ForkJoinPool` utiliza un esquema de **trabajo robado** (*work stealing*), lo que significa que los hilos o "trabajadores" (workers) que terminan su tarea buscan subtareas pendientes en otras colas para evitar que queden inactivos. Esto maximiza el uso de los núcleos del procesador, permitiendo a la aplicación ejecutar varias subtareas simultáneamente y equilibrar la carga de trabajo automáticamente.

Para utilizar `ForkJoinPool`, es necesario definir una tarea que extienda las clases abstractas `RecursiveTask<V>` o `RecursiveAction`. **`RecursiveTask<V>`** se utiliza cuando la tarea devuelve un resultado, mientras que **`RecursiveAction`** se usa cuando no se espera ningún valor de retorno. El método `compute()` define la lógica de la tarea, incluyendo cómo dividirse en subtareas.

```
import java.util.concurrent.*;

public class SumaForkJoin extends RecursiveTask<Integer> {
	private final int[] array;
	private final int inicio, fin;

	public SumaForkJoin(int[] array, int inicio, int fin) {
	this.array = array;
	this.inicio = inicio;
	this.fin = fin;
	}

	@Override
	protected Integer compute() {
	if (fin - inicio <= 10) {  // Caso base: sumar directamente si el rango es pequeño
	int suma = 0;
	for (int i = inicio; i < fin; i++) {
		suma += array[i];
	}
	return suma;
	} else {  // Dividir la tarea en dos subtareas
	int mitad = (inicio + fin) / 2;
	SumaForkJoin izquierda = new SumaForkJoin(array, inicio, mitad);
	SumaForkJoin derecha = new SumaForkJoin(array, mitad, fin);
```

```
            izquierda.fork();  // Ejecutar la tarea izquierda en paralelo
            return derecha.compute() + izquierda.join();  // Combinar los resultados
            }
            }
}
```

Una vez que se define la tarea, se utiliza una instancia de `ForkJoinPool` para ejecutarla. La tarea principal se envía al pool, que se encarga de dividirla en subtareas y ejecutarlas en paralelo. El método `invoke()` espera el resultado de la tarea.

```
public class Main {
    public static void main(String[] args) {
        int[] array = new int[100];
        for (int i = 0; i < array.length; i++) {
            array[i] = i + 1;
        }

        ForkJoinPool pool = new ForkJoinPool();
        SumaForkJoin tarea = new SumaForkJoin(array, 0, array.length);
        int resultado = pool.invoke(tarea);
        System.out.println("Suma total: " + resultado);
    }
}
```

El uso de `ForkJoinPool` ofrece varias ventajas en aplicaciones que requieren un alto grado de paralelismo. Optimiza el uso de la CPU mediante el esquema de trabajo robado y permite manejar tareas complejas de manera más eficiente que los hilos tradicionales. Sin embargo, es importante tener en cuenta que **no todas las tareas son aptas para este modelo**. Si las subtareas no son independientes o si la sobrecarga de dividirlas es mayor que los beneficios del paralelismo, el rendimiento puede disminuir. Además, debe manejarse correctamente la recursión para evitar el desbordamiento de pila (*stack overflow*).

El patrón Fork/Join es particularmente útil en algoritmos como **merge sort**, **búsqueda paralela**, o tareas de análisis de grandes volúmenes de datos, haciendo que Java sea una opción potente para aplicaciones de alto rendimiento en sistemas modernos multicore.