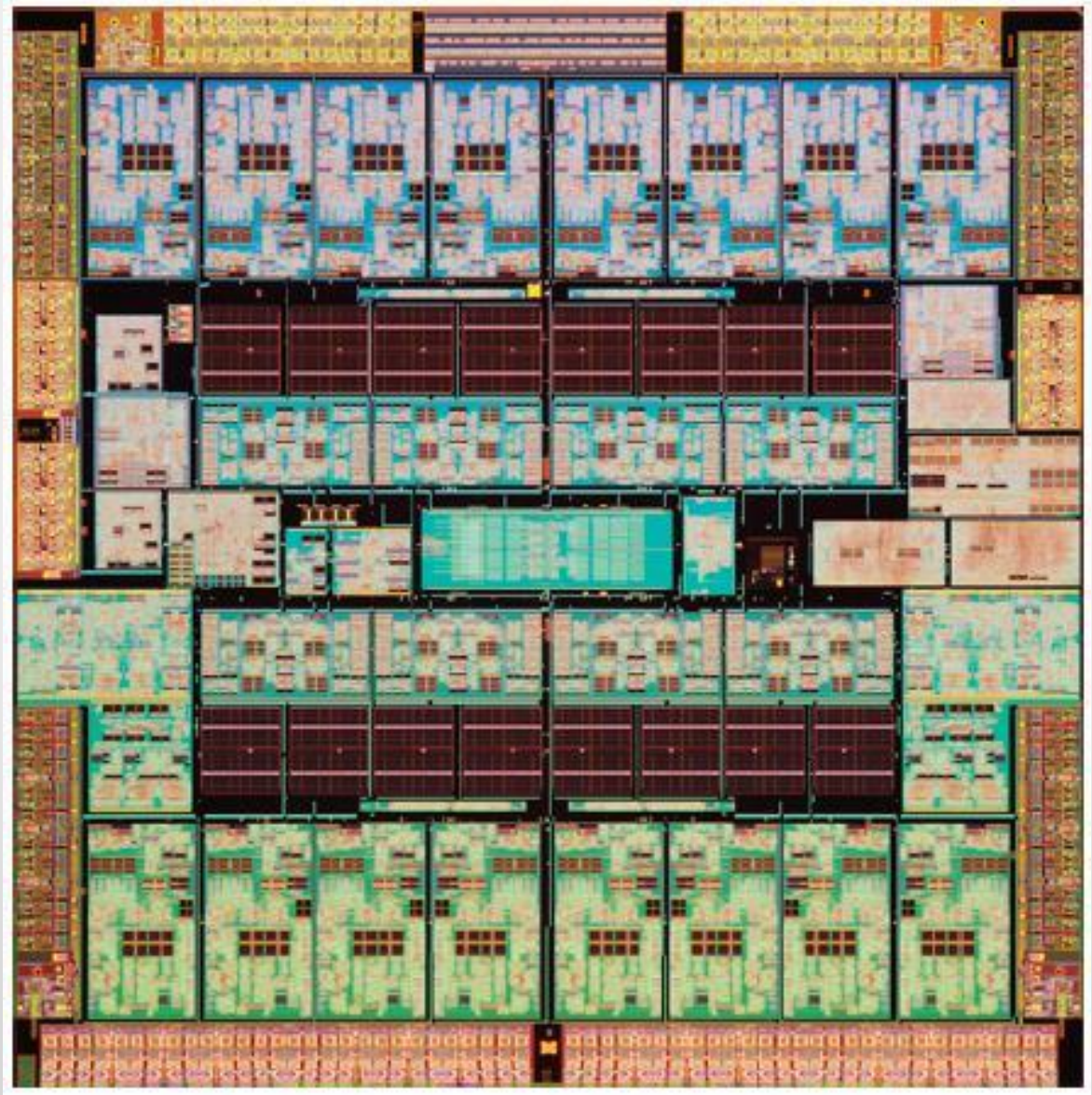
Algoritmia
Grado en Ingeniería Informática del Software
Escuela de Ingeniería Informática – Universidad de Oviedo

Divide y Vencerás Paralelo

Juan Ramón Pérez Pérez

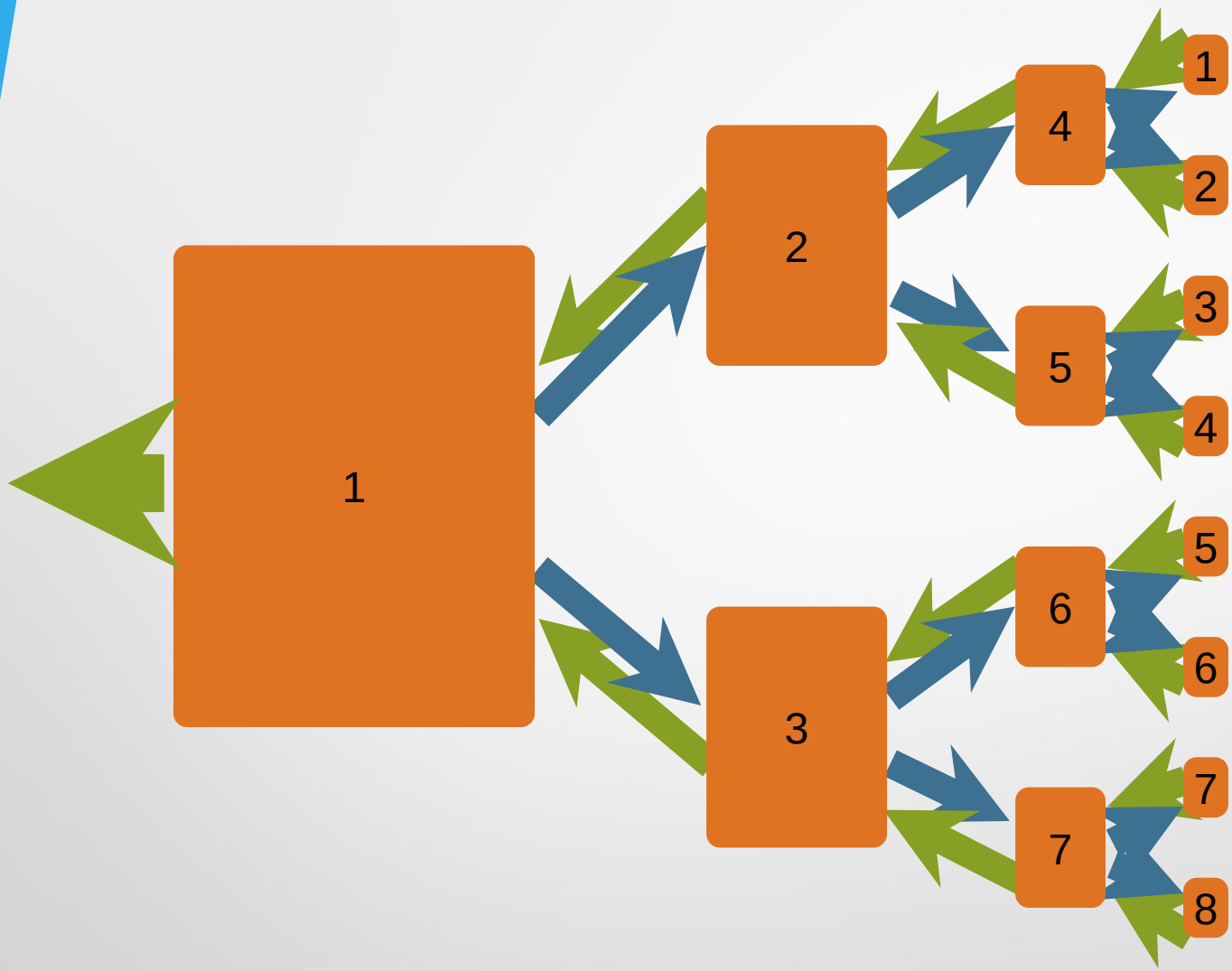
jrpp@uniovi.es

Introducción al diseño y análisis de algoritmos en Java
© Juan Ramón Pérez Pérez



Objetivos

1. Mantener todos los nucleos de los procesadores ocupados
2. Minimizar la sobrecarga de sincronización
3. Permitir un escalado lineal en relación al incremento del número de nucleos



| Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Plantamiento paso recursivo → paralelo (1)

```
Result compute(Task t) {
    if (t.size() < SEQUENTIAL_THRESHOLD) {
        return t.computeSequentially();
    } else {
        Result left, right;
        left = compute(p.leftHalf());
        right = compute(p.rightHalf());
        return combine(left, right);
    }
}
```

| Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Plantamiento paso recursivo → paralelo (2)

Ojo, esto no es código real, es pseudocódigo

```
Result compute(Task t) {
    if (t.size() < SEQUENTIAL_THRESHOLD) {
        return t.computeSequentially();
    } else {
        Result left, right;
        INVOKE-IN-PARALLEL {
                left = compute(p.leftHalf());
                right = compute(p.rightHalf());
        }
        return combine(left, right);
    }
}
```

| Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Fork/Join (I)

- A partir de Java 7
- Implementa el interfaz *ExecutorService*
- Trabajos que pueden **dividirse recursivamente** en diferentes piezas que **no trabajen con datos comunes**.
- Objetivo → Utilizar toda la potencia de procesamiento para mejorar el rendimiento de la aplicación.
- El **framework fork/join** se basa en la clase *ForkJoinPool* que ejecuta *ForkJoinTask*.

Esquema de utilización básica

```
if (porción de trabajo es suficientemente pequeña)
    Hacer el trabajo directamente
else
    dividir el trabajo en trozos
    invocar los trozos en diferentes hilos y
      esperar resultados
```

- Este código debe encapsularse en una subclase de ForkJoinTask:
  - **RecursiveTask**, que puede devolver resultados
  - **RecursiveAction**
- Para ejecutarlo: crear una instancia de la tarea y pasarlo al método *invoke()*.

Implementación búsqueda de máximo paralelo (RecursiveAction)

```
public class MaximoParalelo extends RecursiveAction {
private int inicio, fin, resultado, array[];
…
@Override
protected void compute() {
     if (fin-inicio==1) {
          resultado= array[inicio];
     }
     else {
          int mitad= (inicio+fin)/2;
          MaximoParalelo max1= new MaximoParalelo(array,inicio,mitad);
          MaximoParalelo max2= new MaximoParalelo(array,mitad,fin);
          invokeAll(max1,max2);
          resultado= Math.max(max1.resultado, max2.resultado);
     }
}
```

Implementación búsqueda de máximo paralelo (RecursiveTask)

```
public class MaximoParalelo extends RecursiveTask<Integer> {
private int inicio, fin, array[];
…
@Override
protected Integer compute() {
    if (fin-inicio==1) {
        return array[inicio];
    }
    else {
        int mitad= (inicio+fin)/2;
        MaximoParalelo max1= new MaximoParalelo(array,inicio,mitad);
        MaximoParalelo max2= new MaximoParalelo(array,mitad,fin);
        max1.fork();
        return Math.max(max2.compute(), max1.join());
    }
}
```

Invocación desde el método main

```
public static void main(String args[]) {

    MaximoParalelo maximo= new MaximoParalelo(array, 0,
array.length);
    ForkJoinPool pool= new ForkJoinPool();
    pool.invoke(maximo);
    System.out.println(maximo.resultado);
    ...
}
```

Concurrent Java Animated

- Aplicación que permite visualizar animaciones de las principales estructuras de concurrencia de alto nivel
  - Animación que muestra el funcionamiento
  - Código que ilustra la estructura
  - Breve explicación
- http://sourceforge.net/projects/javaconcurrenta/

Ejercicios propuestos

- Recupera el código recursivo del método de ordenación Quicksort
- Convierte el algoritmo en paralelo utilizando el framework Fork/Join
- Mide tiempos de ejecución para distintos tamaños del problema (*n*) tanto para el algoritmo paralelo como para el secuencial.
- Estudia los tiempos en función del número de núcleos de los que disponga tu ordenador.