Asignación de Prácticas Número 5

Programación Concurrente y de Tiempo Real

Departamento de Ingeniería Informática
Universidad de Cádiz

PCTR, 2022

Contenido I

Objetivos de la Práctica

- Hacer paralelismo de datos con división automática de la nube de datos.
- Aprender a determinar cuántas tareas paralelas deben utilizarse para resolver un problema de forma paralela, en función del coeficiente de bloqueo del problema y el número de *cores* disponibles, mediante la ecuación de Subramanian.
- Desarrollar computación asícrona a futuro mediante el uso combinado de las interfaces `Callabe` y `Future`
- Aplicar todo lo anterior a resolver algunos problemas de forma paralela: producto matricial, operador de resaltado de una imagen y búsqueda de números perfectos.

Algunas Definiciones I

- Speedup: número adimensional que cuantifica la aceleración de una solución paralela frente a la solución secuencial para un problema dado. Se define por

$$S = \frac{T(1)}{T(n)}$$

donde $T(1)$ es el tiempo de ejecución del mejor algoritmo secuencial disponible para resolver el problema, y $T(n)$ es el tiempo de ejecución de la solución paralela con n tareas.

Algunas Definiciones II

- Coeficiente de bloqueo: número C_b entre 0 y 1 que mide la fracción (en tanto por uno) de tiempo donde un código esté detenido por latencias de E/S, de red, etc. En arquitecturas multicore de memoria común, la latencia de memoria se suele considerar despreciable, y en arquitecturas heterogéneas tipo cluster, la latencia de memoria podría requerir ser cuantificada, e integrada en C_b.
- Ecuación de Subramanian: es una ecuación de balanceado de carga elemental, que determina el número de tareas paralelas N_t necesarias para resolver una problema, en función de dos variables: número de *cores* lógicos disponibles N_c, y coeficiente de bloqueo C_b:

$$N_t = \frac{N_c}{1 - C_b}$$

Tipos de Problemas Según C_b I

- Utilizamos C_b para clasificar los problemas en dos grandes grupos:
- Problemas de computación Numérica, con $C_b = 0$:
  - Producto escalar de vectores
  - Producto de matrices
  - Ecuación de ondas en un medio bidimensional, integración numérica, etc.
- Problemas de cualquier otra naturaleza, con $0 < C_b < 1$
  - Implementación de servidores multihebrados
  - Software con alta interacción de red...
  - ... e incluso problemas del primer grupo, cuando se resuelven en arquiteturas heterogéneas

Cómo Preguntarle al S.O. Cuántos *Cores* Hay I

- Es necesario cuando programamos una aplicación que utiliza paralelismo y puede ser ejecutada en diferentes plataformas.
- Cuando la aplicación se ejecuta, antes de decidir cuántas tareas paralelas va a tener, y dividir la nube de datos entre ellas, necesita saber cuántos *cores* hay.
- Con Java, es posible interrogar al sistema operativo sobre esto utilizando el método `Runtime.getRuntime().availableProcessors()`.
- El sistema contesta con el número de núcleos lógicos visibles... y la aplicación puede aplicar la ecuación de Subramanian, determinar cuántas tareas paralelas se requieren, y dividir -en su caso- la nube de datos entre ellas.

Ejemplillo de Uso de availableProcessors() I

```
1  public class nNuc{
2    public static void main(String[] args){
3      int nProc = Runtime.getRuntime().availableProcessors();
4      System.out.println("Nucleos disponibles: "+nProc);
5    }
6  }
```

Trabajamos En el Ejercicio 1: Búsqueda Secuencial de Números Primos I

```
1   public class PrimosSecuencial{
2
3   public static boolean esPrimo(long n){
4     if(n<=1) return(false);
5     for(long i=2; i<=Math.sqrt(n); i++)
6       if(n%i == 0) return(false);
7     return(true);
8   }
9
10  public static void main(String[] args) throws Exception{
11    long intervalo = Long.parseLong(args[0]);
12    int total = 0;
13
14    long inicTiempo = System.nanoTime();
15    for(long i=0; i<=intervalo;i++)
16      if(esPrimo(i)) total++;
17    long tiempoTotal =
          (System.nanoTime()-inicTiempo)/(long)1.0e9;
18    System.out.println("Encontrados "+total+" primos"+" en "+
          tiempoTotal+" segundos");
```

Trabajamos En el Ejercicio 1: Búsqueda Secuencial de Números Primos II

```
19        }
20    }
```

Trabajamos En el Ejercicio 1: Búsqueda Paralela de Números Primos con Callable-Future I

Escribimos la clase para las tareas paralelas...

```
1  import java.util.concurrent.Callable;
2  public class tareaPrimos implements Callable{
3
4    private final long linf;
5    private final long lsup;
6    private Long total = new Long(0);
7    //construtor para Long esta "deprecated"
8
9    public tareaPrimos(long linf, long lsup){
10     this.linf = linf;
11     this.lsup = lsup;
12   }
13
14   public boolean esPrimo(long n){
15     if(n<=1) return(false);
16     for(long i=2; i<=Math.sqrt(n); i++)
17       if(n%i == 0) return(false);
18     return(true);
```

Trabajamos En el Ejercicio 1: Búsqueda Paralela de Números Primos con Callable-Future II

```
19      }
20
21      public Long call(){
22        for(long i=linf; i<=lsup;i++)
23          if(esPrimo(i)) total++;
24      return(total);
25      }
26    }
```

Y ahora el programa principal...

```
1
2   import java.util.concurrent.*;
3   import java.util.*;
4
5   public class primosParalelos {
6
7     public static void main(String[] args) throws Exception {
8       long nPuntos     = Integer.parseInt(args[0]);
```

Trabajamos En el Ejercicio 1: Búsqueda Paralela de Números Primos con Callable-Future III

```
9       int  nTareas     =
            Runtime.getRuntime().availableProcessors();
10      long tVentana    = nPuntos/nTareas;
11      long primosTotal = 0;
12      long linf        = 0;
13      long lsup        = tVentana;
14
15      List<Future<Long>> contParciales =
            Collections.synchronizedList(
16        new ArrayList<Future<Long>>());
17      long inicTiempo = System.nanoTime();
18      ThreadPoolExecutor ept = new ThreadPoolExecutor(
19        nTareas,
20        nTareas,
21        0L,
22        TimeUnit.MILLISECONDS,
23        new LinkedBlockingQueue<Runnable>());
24      for(int i=0; i<nTareas; i++){
25        contParciales.add(ept.submit(
26           new tareaPrimos(linf, lsup)));
```

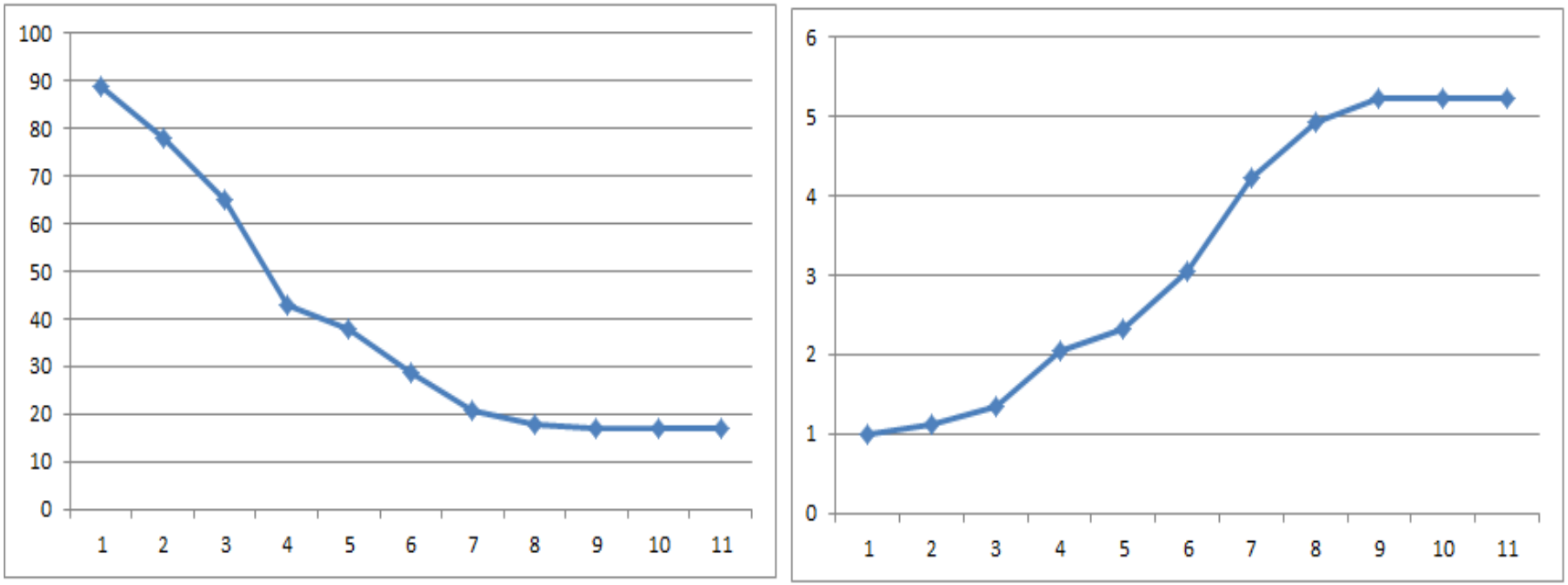
Trabajamos En el Ejercicio 1: Búsqueda Paralela de Números Primos con Callable-Future IV

```
27          linf=lsup+1;
28          lsup+=tVentana;
29        }
30        for(Future<Long> iterador:contParciales)
31          try{
32              primosTotal +=  iterador.get();
33          }catch (CancellationException e){}
34           catch (ExecutionException e){}
35           catch (InterruptedException e){}
36        long tiempoTotal =
              (System.nanoTime()-inicTiempo)/(long)1.0e9;
37        ept.shutdown();
38        System.out.println("Primos hallados: "+primosTotal);
39        System.out.println("Calculo finalizado en "+tiempoTotal+"
              segundos");
40     }
41   }
```

Trabajamos En el Ejercicio 1: Calculando El SpeedUp I

- Ejecutamos el programa secuencial y tomamos tiempos.
- Ejecutamos el programa paralelo y tomamos tiempos.
- Ahora calculamos el speedup y sabemos la aceleración lograda.

Curvas Típicas en Problemas con C_b=0 I



Curvas de Tiempo y Speedup

Trabajamos En el Ejercicio 1: Descarga Secuencial de Página Web I

```
1  import java.util.*;
2  import java.io.*;
3  import java.net.*;
4
5  public class volcadoRedSecuencial{
6
7    public static void descargar(String dir, int i){
8        try{
9           URL url = new URL(dir);
10          BufferedReader reader = new BufferedReader(new
                InputStreamReader(url.openStream()));
11           FileWriter           f = new FileWriter(i+".html");
12          BufferedWriter writer = new BufferedWriter(f);
13          String line;
14          while ((line = reader.readLine()) !=
                null){writer.write(line);}
15          f.close();
16          System.out.println(dir+" descargada...");
17       }catch(IOException e){}
18    }
```

Trabajamos En el Ejercicio 1: Descarga Secuencial de Página Web II

```
19
20      public static void main(String[] args){
21
22        int cont=0;
23        long iniTiempo = System.nanoTime();
24        try {
25           String linea=" ";
26           RandomAccessFile direcciones = new
                 RandomAccessFile("direccionesRed.txt","r");
27            while(linea!=null){
28              linea =(String)direcciones.readLine();
29              if(linea!=null)descargar(linea, cont);
30              cont++;
31            }
32            direcciones.close();
33           }catch (FileNotFoundException e) {}
34            catch (IOException e) {}
35        long finTiempo = System.nanoTime();
36        System.out.println("Tiempo Total (segundos):
              "+(finTiempo-iniTiempo)/1.0e9);
```

Trabajamos En el Ejercicio 1: Descarga Secuencial de Página Web III

```
37        }
38    }
```

Trabajamos En el Ejercicio 1: Descarga Paralela de Página Web I

Escribimos la clase para las tareas paralelas...

```
1  import java.util.*;
2  import java.io.*;
3  import java.net.*;
4
5  public class tareaRed implements Runnable{
6
7    private String dir;
8    private URL url;
9    private int j;
10
11   public tareaRed(String d, int i){dir=d; j=i;}
12
13   public void run(){
14     try{
15          URL url = new URL(dir);
16          BufferedReader reader = new BufferedReader(new
                InputStreamReader(url.openStream()));
17          FileWriter           f = new FileWriter(j+".html");
```

Trabajamos En el Ejercicio 1: Descarga Paralela de Página Web II

```
18              BufferedWriter writer = new BufferedWriter(f);
19              String line;
20              while ((line = reader.readLine()) != null) {
21                 writer.write(line);
22              }
23              f.close();
24              System.out.println(dir+" descargada...");
25           }catch(IOException e){}
26       }
27   }
```

Y ahora el programa principal...

Trabajamos En el Ejercicio 1: Descarga Paralela de Página Web III

```
1  import java.util.*;
2  import java.io.*;
3  import java.util.concurrent.*;
4
5  public class volcadoRed
6  {
7    public static void main(String[] args) throws Exception
8    {
9      long iniTiempo=0;
10     LinkedList<tareaRed> tareas = new LinkedList<tareaRed>();
11     int nNuc = Runtime.getRuntime().availableProcessors();
12     float Cb = Float.parseFloat(args[0]);
13     int tamPool = (int)(nNuc/(1-Cb));
14     ThreadPoolExecutor ept = new ThreadPoolExecutor(
15           tamPool,
16           tamPool,
17           0L,
18             TimeUnit.MILLISECONDS,
19             new LinkedBlockingQueue<Runnable>());
20     ept.prestartAllCoreThreads();
```

Trabajamos En el Ejercicio 1: Descarga Paralela de Página Web IV

```
21        try {
22           int cont = 0;
23           String linea=" ";
24           RandomAccessFile direcciones = new
                 RandomAccessFile("direccionesRed.txt","r");
25             iniTiempo = System.nanoTime();
26             while(linea!=null){
27               linea =(String)direcciones.readLine();
28               if(linea!=null)tareas.add(new tareaRed(linea, cont));
29               cont++;
30             }
31             direcciones.close();
32            } catch (EOFException e) {}
33        for (Iterator iter = tareas.iterator(); iter.hasNext();)
34          ept.execute((Runnable)iter.next());
35        ept.shutdown();
36        while(!ept.isTerminated()){}
37        long finTiempo = System.nanoTime();
38        System.out.println("Numero de Nucleos: "+nNuc);
39        System.out.println("Coficiente de Bloqueo: "+Cb);
```

Trabajamos En el Ejercicio 1: Descarga Paralela de Página Web V

```
40          System.out.println("Tamano del Pool: "+tamPool);
41          System.out.println("Tiempo Total (segundos):
                "+(finTiempo-iniTiempo)/1.0e9);
42      }
43  }
```

Trabajamos En el Ejercicio 1: Qué hago ahora? I

- Para el problema de los números primos, experimente con un número de tareas creciente $n = 2, 4, 8...$ y deduzca cuál es el número de tareas óptimo.
- Para el problema de la descarga de páginas web, experimente con diferentes coeficientes de bloqueo (entre cero y uno) y estima su valor óptimo aproximado.

Trabajamos En los ejercicios 2, 3, 4 y 5: Qué hago ahora? I

- Son ejemplos de problemas de computación numérica para los cuales es posible suponer que $C_b = 0$.
- Para cada ejercicio, desarrolle una solución paralela con división automática del dominio de datos entre tareas, y contraste la mejora de rendimiento que se alcanza para diferentes números de tareas, de acuerdo a las especificaciones recogidas en el documento de asignación.
- Prepare los documentos de análisis solicitados incluyendo la información que se pide para ellos, y la intepretación que realiza de la misma.
- En este punto, usted debería dominar el procesamiento paralelo con memoria común aplicado a nubes de datos en una y dos dimensiones de estructura regular.