Asignación de Prácticas Número 5

Programación Concurrente y de Tiempo Real

Antonio J. Tomeu[1]

[1]Departamento de Ingeniería Informática
Universidad de Cádiz

PCTR, 2020

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
8    public tareaPrimos(long linf, long lsup){
9      this.linf = linf;
10     this.lsup = lsup;
11   }
12
13   public boolean esPrimo(long n){
14     if(n<=1) return(false);
15     for(long i=2; i<=Math.sqrt(n); i++)
16       if(n%i == 0) return(false);
17     return(true);
18   }
```

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

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

Y ahora el programa principal...

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

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

```
10      long primosTotal = 0;
11      long linf        = 0;
12      long lsup        = tVentana;
13
14      ArrayList<Future<Long>> contParciales =
15        new ArrayList<Future<Long>>();
16      long inicTiempo = System.nanoTime();
17      ThreadPoolExecutor ept = new ThreadPoolExecutor(
18        nTareas,
19        nTareas,
20        0L,
21        TimeUnit.MILLISECONDS,
22        new LinkedBlockingQueue<Runnable>());
23      for(int i=0; i<nTareas; i++){
24        contParciales.add(ept.submit(
25           new tareaPrimos(linf, lsup)));
26        linf=lsup+1;
27        lsup+=tVentana;
28      }
29      for(Future<Long> iterador:contParciales)
```

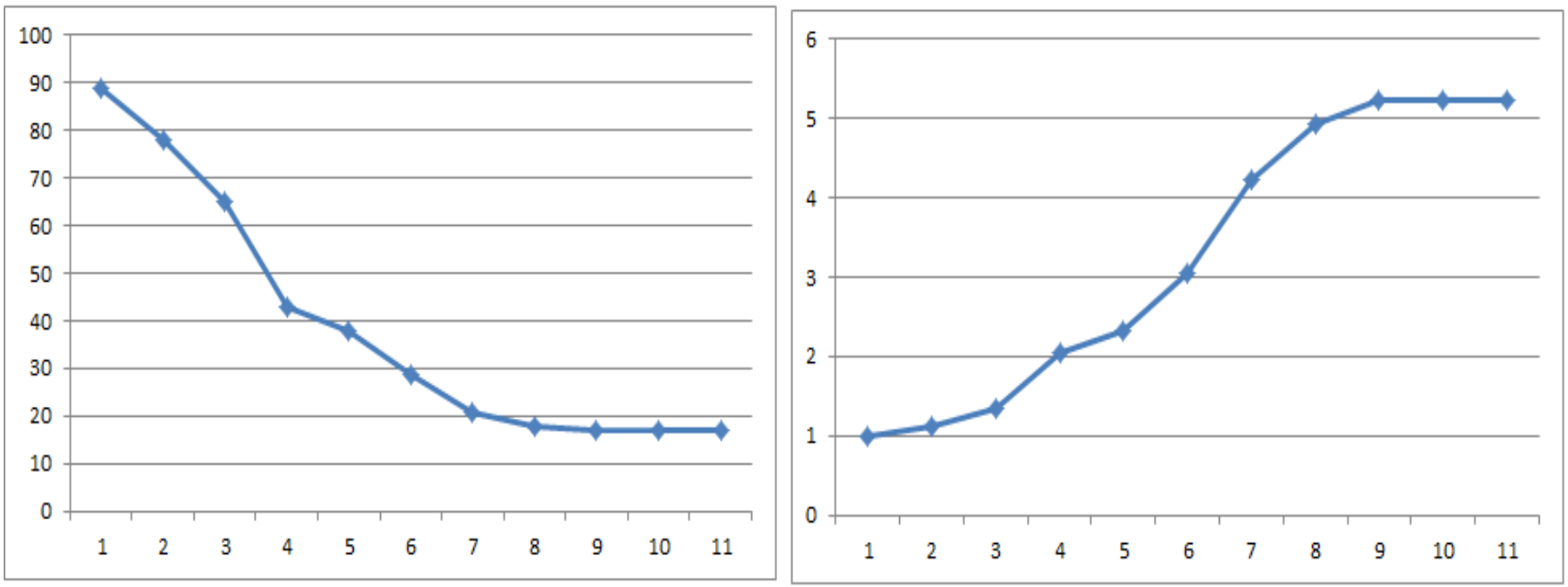
Trabajamos En el Ejercicio 1: Búsqueda Paralela de Números Primos con Callable-Future IV

```
30          try{
31              primosTotal +=  iterador.get();
32          }catch (CancellationException e){}
33           catch (ExecutionException e){}
34           catch (InterruptedException e){}
35        long tiempoTotal =
              (System.nanoTime()-inicTiempo)/(long)1.0e9;
36        ept.shutdown();
37        System.out.println("Primos hallados: "+primosTotal);
38        System.out.println("Calculo finalizado en "+tiempoTotal+"
              segundos");
39     }
40   }
```

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 navegarURL(String d, int i){
8
9      String dir=d;
10     String datos;
11     RandomAccessFile volcado;
12     URL url;
13     Integer j=new Integer(i);
14
15     try{
16         url =  new URL(dir);
17         System.out.println("Contactando con "+dir);
18
19         String name = j.toString()+".html";
20         volcado = new RandomAccessFile(name, "rw");
```

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

```
21          BufferedReader lector = new BufferedReader(
22         new InputStreamReader(url.openStream()));
23          do{
24             datos = lector.readLine();
25             if(datos!=null)volcado.writeChars(datos);
26          }while(datos!=null);
27          volcado.close();
28        }catch(IOException e){}
29
30    }
31
32
33
34    public static void main(String[] args){
35      int cont=0;
36      long iniTiempo = System.nanoTime();
37      try {
38         String linea=" ";
39         RandomAccessFile direcciones = new
               RandomAccessFile("direccionesRed.txt","r");
```

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

```
40              while(linea!=null){
41                linea =(String)direcciones.readLine();
42                if(linea!=null)navegarURL(linea, cont);
43                cont++;
44              }
45              direcciones.close();
46             }catch (FileNotFoundException e) {}
47              catch (IOException e) {}
48        long finTiempo = System.nanoTime();
49        System.out.println("Tiempo Total (segundos):
              "+(finTiempo-iniTiempo)/1.0e9);
50      }
51    }
```

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

Escribimos la clase para las tareas paralelas...

```
1  /*tareaRed.java
2   @author A.Tomeu
3  */
4  import java.util.*;
5  import java.io.*;
6  import java.net.*;
7
8  public class tareaRed implements Runnable{
9
10   private String dir;
11   private String datos;
12   private RandomAccessFile volcado;
13   private URL url;
14   private Integer j;
15
16   public tareaRed(String d, int i){dir=d; j=new Integer(i);}
17
18   public void run()
```

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

```
19      {
20        try{url =  new URL(dir);} catch(MalformedURLException
              e){}
21        System.out.println("Contactando con "+dir);
22        try{String name = j.toString()+".html";
23            volcado = new RandomAccessFile(name, "rw");
24            BufferedReader lector = new BufferedReader(
25          new InputStreamReader(url.openStream()));
26        do{
27           datos = lector.readLine();
28           if(datos!=null)volcado.writeChars(datos);
29        //System.out.println("escribiendo...");
30        }while(datos!=null);
31        volcado.close();
32       }catch(IOException e){}
33    }
34   }
```

Y ahora el programa principal...

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

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

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

```
21              0L,
22                TimeUnit.MILLISECONDS,
23                new LinkedBlockingQueue<Runnable>());
24        ept.prestartAllCoreThreads();
25        try {
26           int cont = 0;
27           String linea=" ";
28           RandomAccessFile direcciones = new
                 RandomAccessFile("direccionesRed.txt","r");
29             iniTiempo = System.nanoTime();
30             while(linea!=null){
31               linea =(String)direcciones.readLine();
32               if(linea!=null)tareas.add(new tareaRed(linea, cont));
33               cont++;
34             }
35             direcciones.close();
36            } catch (EOFException e) {}
37        for (Iterator iter = tareas.iterator(); iter.hasNext();)
38          ept.execute((Runnable)iter.next());
39        ept.shutdown();
```

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

```
40          while(!ept.isTerminated()){}
41          long finTiempo = System.nanoTime();
42          System.out.println("Numero de Nucleos: "+nNuc);
43          System.out.println("Coficiente de Bloqueo: "+Cb);
44          System.out.println("Tamano del Pool: "+tamPool);
45          System.out.println("Tiempo Total (segundos):
                "+(finTiempo-iniTiempo)/1.0e9);
46      }
47  }
```

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.