Java Parallel Processing Framework

JPPF

Reporte

Autor:

Montané Jiménez Luis Gerardo

Cómputo de Alto Desempeño.

Xalapa de Eqz., Ver. Febrero 2015

Contenido

Implementación de una aplicación

A continuación se presenta una aplicación que forma parte de un algoritmo genético (AG). Esta aplicación tiene como fin, generar la población mediante un Grid de computadoras utilizando JPPF, para después utilizarla en las otras fases del AG.

Un algoritmo genético (AG) es una meta-heurística que sirve para optimizar diferentes problemas. Estos fueron propuestos por *Holland* en los años 60´s en la Universidad de Michigan.

Los AG se componen de cinco elementos:

1. Representación de individuos (población)
2. Selección de padres
3. Recombinación o cruza
4. Mutación
5. Reemplazo

En esta aplicación se mostrará el código mínimo para la representación de individuos (población) del AG. Para la generación de la población mediante JPPF se utilizaron tres clases:

- Población
- Generar Población
- Main

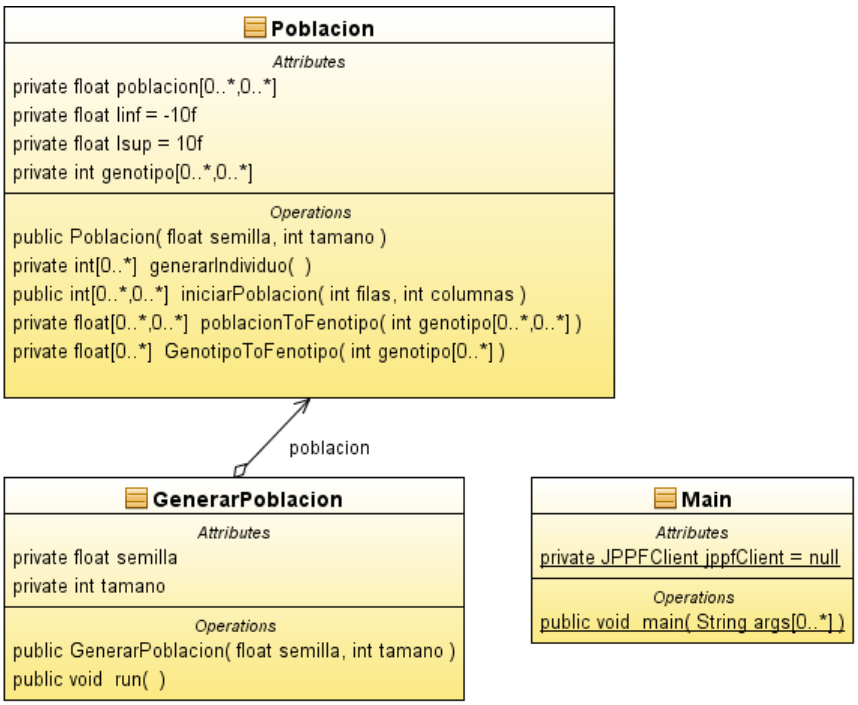

Fig 5. Diagrama de Clases

Código fuente

```
//Librerías para el funcionamiento de JPPF en la clase Main.java
importorg.jppf.JPPFException;
importorg.jppf.client.JPPFClient;
importorg.jppf.client.JPPFJob;
importorg.jppf.server.protocol.JPPFTask;


public class Main {

//Declararun objeto cliente JPPF, que se utilizará para enviar las
peticiones de ejecución:

      private static JPPFClientjppfClient = null;

public static void main(String[] args) throws IOException,
JPPFException {
int tarea1 = 50;
int tarea2 = 30;
int tarea3 = 20;
int genotipo[][] = null;

//Se crea un objeto JOB para poder realizar las tareas en paralelo

jppfClient = new JPPFClient();
JPPFJob job = new JPPFJob();
job.setId("initPopulation");
job.addTask(new GenerarPoblacion(0.3f,tarea1));
job.addTask(new GenerarPoblacion(.23f,tarea2));
job.addTask(new GenerarPoblacion(.53f,tarea3));
job.setBlocking(true);

Poblacionpoblacion = null;

try {
System.out.println("Lanzando inicio de poblacion....");

//Se genera una lista resultados provenientes de las tareas ejecutadas
por JPPF

List<JPPFTask> results = jppfClient.submit(job);
List<Poblacion> poblaciones = new ArrayList<Poblacion>();


for (int i=0; i<results.size(); i++)
{
            System.out.println("Obteniendo resultados");

//Obtener la matriz de tareas realizadas por las computadoras nodos del
GRID

      JPPFTaskmatrixTask = results.get(i);

if(matrixTask.getException() != null) throw
```

```
matrixTask.getException();
//La instancia población es llenada con la matriz de tareas

poblacion = (Poblacion) matrixTask.getResult();
poblaciones.add(poblacion);
if (poblacion != null){
System.out.println("Poblacion generada exitosamente");
            binario.util.BitsOperation.imprimirPoblacion
(poblacion.getGenotipo());
}
else
System.out.println("Poblacion no generadaaa!!");

} //for

//Se juntan todos los individuos generados
genotipo=
newint[tarea1+tarea2][poblacion.getGenotipo()[0].length];
intindexGlobal = 0;
for (int index = 0;index <poblaciones.size(); index++){
Poblaciontmp = poblaciones.get(index);
for (int index0=0;index0<tmp.getGenotipo().length;index0++)
genotipo[indexGlobal++] = tmp.getGenotipo()[index0];

}

} catch (Exception ex) {
Logger.getLogger(Main.class.getName()).log(
Level.SEVERE, null, ex);
}finally
            {
                  if (jppfClient != null) jppfClient.close();
            }
System.out.println("Termino de generar poblacion");
System.out.println("Genotipo completo ---- >");

//Imprimir en pantalla la población generada
binario.util.BitsOperation.imprimirPoblacion(genotipo);
EvolutionBitseb = new EvolutionBits(.3f,genotipo);
eb.start();
}
}
```

La clase generar poblacion.java, es la clase que se va a distribuir en el Grid de computadoras, en otras palabra es la parte del procesamiento en paralelo.

Para empezar se necesitan las siguientes librerías:

```
importorg.jppf.server.protocol.JPPFTask;

//Se tiene que hacer una herencia de la super clase JPPFTask la cual
contiene el método run() que //sirve para ejecutarse en paralelo.
```

```
public class GenerarPoblacion extends JPPFTask{
privatePoblacionpoblacion;
privatefloat semilla;
privateinttamano;
publicGenerarPoblacion(float semilla, inttamano){
this.semilla = semilla;
this.tamano = tamano;
}

//Todo lo escrito en este método será distribuido por el JPPF en el Grid
de computadoras.

public void run() {
try
{
System.out.println("Generandopoblacion.....");
System.out.println("Semilla: "+semilla);
System.out.println("Tamaño de poblacion a generar: "
+ tamano);
poblacion = new Poblacion(semilla, tamano);
setResult(poblacion);
}
catch(Exception e)
{
setException(e);
}

}

}
```

Aplicación

Definición del problema

En la sección 2 de este documento se presentó la de implementación de una aplicación donde se generaba la población de un algoritmo genético de forma distribuida. En esta sección se continuará con el desarrollo de la misma problemática, sin embargo, los puntos a paralelizar ahora son la traducción de un individuo genotipo a fenotipo. Este proceso consiste en tener una cadena de bits que representan 10 variables en número reales (esto se le llama genotipo), la representación en números reales es el fenotipo.

El proceso de traducción o mapeo de genotipo a fenotipo se realiza al finalizar la generación de población, y posteriormente en el proceso de reemplazo de cada generación, donde por cada hijo generado se realiza la traducción. La última parte que ha sido distribuida para la ejecución del algoritmo es la fase de Cruza. A continuación se muestra el diagrama de procesos que muestra la representación del algoritmo genético.

Diagrama de procesos

En la figura 13 que a continuación se muestra se representan las fases del algoritmo genético, existe un estado de inicio que posteriormente va al proceso generar población y mapeo de individuos de genotipo a fenotipo. El proceso que continua es la selección, este se comunica con la cruza que a su vez va al proceso de mutación y reemplazo. En el proceso de reemplazo es donde también se realiza el mapeo de Genotipo a Fenotipo. Los óvalos resaltados son los procesos que se distribuyeron con JPPF.

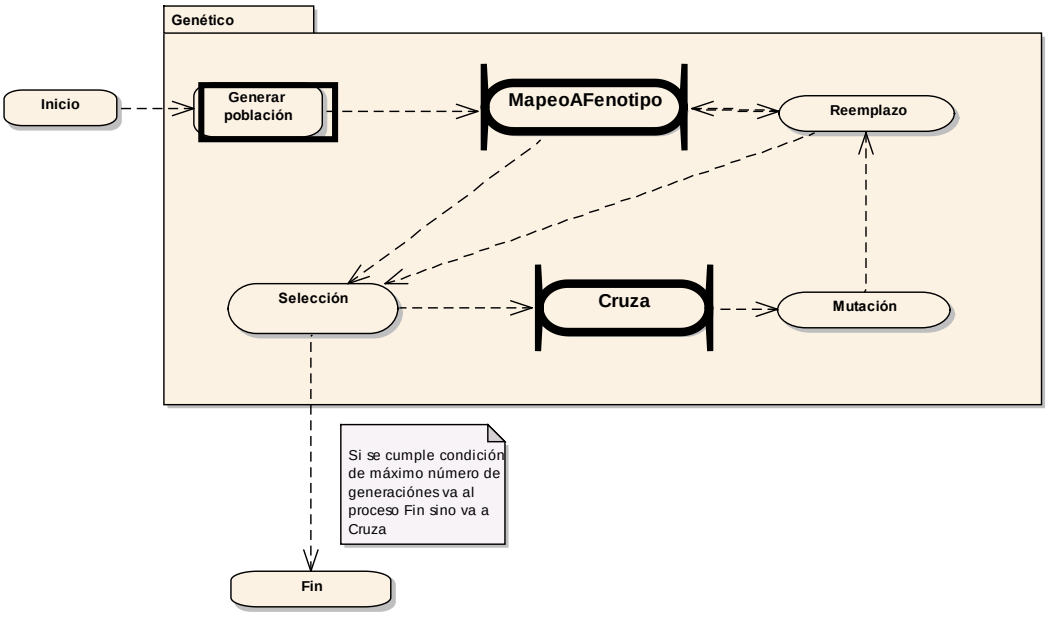

Figura 13. Diagrama de proceso.

Solución

Las fases del algoritmo genético que se estarían definiendo para ser ejecutadas en el grid serian:

1. Representación de individuos (Explicada en la sección 2)
2. Recombinación o cruza (Clase `EjecutarCruza`)
3. Reemplazo (Parte de la codificación en clase `GenotipoToFenotipo`)

Para realizar las tareas se debe importar la librería necesaria para heredar de JPPFTask:

```
import org.jppf.server.protocol.JPPFTask;
```

A continuación se muestra la clase que representa la tarea EjecutarCruza, esta clase es llamada en el algoritmo principal del genético, lo primero que puede apreciarse es la herencia que hay con la clase JPPFTask e implementación del método run (que resulta necesario para el framework).

```
public class EjecutarCruza  extends JPPFTask{
    private int[][] hijos;
    private int tamano;
    private int longitudCadena;
    private int individuoTamano;
    private int[][] padres1;
    private int[][] padres2;
    private float probCruza;

    public  EjecutarCruza(int  tamano,  int  longitudCadena,  int  individuoTamano,  float
probCruza, int [][] padres1, int [][] padres2){

        this.longitudCadena = longitudCadena;
        this.tamano = tamano;
        this.individuoTamano = individuoTamano;
        this.probCruza = probCruza;
        this.padres1 = padres1;
        this.padres2 = padres2;
    }

    public void cruza(){
        hijos = new int[tamano][longitudCadena * individuoTamano];
        int indexHijo1 = 0;
        int indexHijo2 = indexHijo1 + 1;
        for (int index = 0; index < padres1.length; index++){
            hijos[indexHijo1] = padres1[index];
            hijos[indexHijo2] = padres2[index];

            //Si probabilidad es 1 entonces aplica cruza de dos puntos
            if (Random.flip(this.probCruza) == 1){
                int ktmp0 =  Random.rnd(0, (this.longitudCadena * 10) - 1);
                int    ktmp1    =        BitsOperation.generarAleatorioDiferente(ktmp0,0,
(this.longitudCadena * 10) - 1);
                int k1,k2;

                if (ktmp0 > ktmp1){
                    k1 = ktmp1;
                    k2 = ktmp0;
                }else{
                    k1 = ktmp0;
                    k2 = ktmp1;
                }
```

```
                for (int ik = k1; ik <= k2;ik++){

                    //intercambio
                    int aux1 = hijos[indexHijo1][ik];
                    int aux2 = hijos[indexHijo2][ik];
                    hijos[indexHijo1][ik] = aux2;
                    hijos[indexHijo2][ik] = aux1;
                }
            }
            indexHijo1 += 2;
            indexHijo2 = indexHijo1 + 1;
        }

    }

    public void run() {
        try
        {
            System.out.println("Realizando cruza!!");
             cruza();
             setResult(hijos);
         }
         catch(Exception e)
         {
        setException(e);
         }
    }
}
```

La traducción de genotipo a fenotipo es realizada con la siguiente clase, y al igual que la anterior se hereda de JPPFTask y se implementa el método run:

```
Public class GenotipoToFenotipo  extends JPPFTask{
   private int genotipo[];
   private int individuoTamano;
   private int longitudCadena;
   private float linf;
   private float lsup;

   public GenotipoToFenotipo(int[] genotipo, int individuoTamano, int longitudCadena, float
linf, float lsup){
        this.genotipo = genotipo;
        this.individuoTamano = individuoTamano;
        this.longitudCadena = longitudCadena;
        this.linf = linf;
        this.lsup = lsup;
   }

   public float[] GenotipoToFenotipo(){
        float[] fenotipo = new float[this.individuoTamano];
        int indexGenotipo = 0;
        int[] individuoGenotipo = new int[this.longitudCadena];
        for(int index = 0; index < this.individuoTamano; index++){
                System.arraycopy(genotipo,    indexGenotipo,    individuoGenotipo,    0,
longitudCadena);
                indexGenotipo = index * this.longitudCadena;
                fenotipo[index]                                                          =
TruncarDecimal.truncate(BitsOperation.codificacion(individuoGenotipo, linf, lsup));
        }
        return fenotipo;
    }

    public void run() {
        //Traducir genotipo
        System.out.println("Traduciendo genotipo a fenotipo...");
        setResult(this.GenotipoToFenotipo());
    }
}
```

Las llamadas a las ejecuciones de traducción y cruza se realiza mediante la definición de un Job (trabajos), a este Job se le agregan las tareas que ejecutaran los nodos como se muestra en el siguiente segmento de código.

```
        float[][] poblacion = new

float[this.getPoblacionTamano()][this.individuoTamano];
        try {
                                    //JPPFJob en la se agregan las
tareas de traducción de GenotipoToFenotipo
            JPPFJob job = new JPPFJob();
            job.setId("toGenotipo" + 1);

            for (int index0 = 0; index0 < genotipo.length;
index0++)
                 job.addTask(new
GenotipoToFenotipo(genotipo[index0], individuoTamano,
longitudCadena, linf, lsup));

            job.setBlocking(true);
          //  System.out.println("ToGenotipo....");

            List<JPPFTask> results = jppfClient.submit(job);
            for (int i = 0; i < results.size(); i++)
            {
            JPPFTask matrixTask = results.get(i);
                if (matrixTask.getException() != null) throw
matrixTask.getException();
                poblacion[i] = (float[]) matrixTask.getResult();
            }

       }catch(JPPFException ex){
           System.out.println("Error en conversión a genotipo" +
ex.toString());
       }catch(Exception ex1){
           System.out.println("Error Exception en conversión a
genotipo");
       }
       return poblacion;
  }
```

Resultados

El resultado devuelto por la aplicación cliente es el siguiente:

```
Gen 198
0.08,0.08,-0.02,0.11,0.17,-0.36,-0.27,0.17,0.01,0.04, f(x) = 0.28
Gen 199
0.08,0.08,-0.02,0.11,0.17,-0.36,-0.27,0.17,0.01,0.04, f(x) = 0.28
Número de evaluaciones = 20000
Número de generaciones = 200
BUILD SUCCESSFUL (total time: 29 seconds)
```

Figura 14. Resultados de la aplicación cliente.

En los nodos la información plasmada es:

```
[java] Traduciendo genotipo a fenotipo...
[java] Traduciendo genotipo a fenotipo...
[java] Traduciendo genotipo a fenotipo...
[java] Realizando cruza!!
[java] Traduciendo genotipo a fenotipo...
```

Figura 15. Resultado de los nodos.

Referencias

http://www.jppf.org/

http://ant.apache.org/

http://www.sandia.gov/about/faq/