Taller Paralelización de Datos

Estudiante: Juan Sebastian Getial Getial

Código: 202124644

Asignatura: Fundamentos de Programación Funcional y Concurrente

Docente: Juan Francisco Diaz Frias

Universidad del Valle

Santiago de Cali

2022

Informe de Uso de Colecciones Paralelas

Funciones	Colecciones Paralelas
clasificarSeq	**NO**
clasificarPar	**SI**
actualizarSeq	**NO**
actualizarPar	**SI**
hayConvergenciaSeq	**NO**
hayConvergenciaPar	**SI**
kMedianasSeq	**NO**
kMedianasPar	**SI**

En cuanto a la dificultad a la hora de implementar las colecciones paralelas, se puede decir que no hubo ninguna, ya que básicamente solo era cambiar el tipo de dato en todos los casos.
Por otro lado, de acuerdo a lo realizado, se puede apreciar como las colecciones paralelas son una herramienta muy útil a la hora de paralelizar, debido a que hacen todo el trabajo de paralelización automáticamente, por lo que el programador sólo debe tener en cuenta en qué casos no se deben utilizar.

Informe de Corrección

Argumentación Sobre la Corrección

Preliminares

A continuación se tendrán en cuenta las siguientes definiciones:

$\forall\, p \in Punto: p = (x: Double,\ y: Double,\ z: Double)$

Función clasificarSeq

Preliminares.

A continuación se tendrá en cuenta la siguiente función, para la cual se da por hecho su buen funcionamiento, ya que fue proporcionada por el docente:

Función hallarPuntoMasCercano.

$\forall\, p \in Punto,\ medianas \in Seq[Punto]: hallarPuntoMasCercano(p,\ medianas) = hallarPuntoMasCercano(p,\ Seq[m_0,\ ...,\ m_n]) = m_i$, tale que m_i es el Punto más cercano a p.

A continuación se demostrará que:

$\forall\, puntos,\ medianas \in Seq[Punto]: clasificarSeq(puntos,\ medianas) = Map[m_i,\ ptosm_i]$

donde $m_i \in medianas$ y $ptosm_i$ contiene los puntos $\in puntos$ cuya mediana más cercana es m_i.

Demostración.

$clasificarSeq(puntos,\ medianas)$

$\equiv \quad puntos.groupBy(p \Rightarrow hallarPuntoMasCercano(p,\ medianas))$

Note que $hallarPuntoMasCercano(p,\ medianas)$ devuelve la mediana m_i más cercana al punto p, por lo que se agruparan los puntos que tengan la misma mediana m_i como punto más cercano.

$\equiv \quad Map[m_i,\ ptosm_i]$ ♦

Función clasificarPar

Esta función presenta la misma lógica de su versión secuencial, con la única diferencia de que esta utiliza colecciones paralelas, por lo que su argumentación es la misma, por consiguiente, se da por hecho su buen funcionamiento.

Función actualizarSeq

Preliminares.

A continuación se tendrá en cuenta la siguiente función, para la cual se da por hecho su buen funcionamiento, ya que fue proporcionada por el docente:

Función calculePromedioSeq.

$\forall\, m \in Punto,\ puntos \in Seq[Punto]$: $calculePromedioSeq(m,\ puntos)$

Si $puntos \neq Seq[]$

$= p$, donde p es el promedio de todos los puntos $\in puntos$

Si $puntos = Seq[]$

$= m$ ♦

Cabe recalcar que **calculePromedioPar** presenta el mismo funcionamiento pero con colecciones paralelas.

Función nuevaMediana.

$\forall\, mediana \in Punto$: $nuevaMediana(mediana)$

Si $mediana$ no tiene puntos asociados

$= mediana$

Si tiene puntos asociados

$= calculePromedioSeq(mediana,\ puntosAsociados)$

A continuación se demostrará que:

$\forall\, clasif \in Map[Punto,\ Seq[Punto]],\ medianasViejas \in Seq[Punto]$:

$actualizarSeq(clasif,\ medianasViejas)$

$actualizarSeq(Map[m_i,\ Seq[Punto]],\ (m_0,\ \ldots,\ m_n))$

$= (mA_0,\ \ldots,\ mA_n)$, donde mA_i es la mediana m_i actualizada con respecto al promedio de sus puntos correspondientes.

Demostración.

$actualizarSeq(clasif,\ medianasViejas)$

$\equiv$ $medianasViejas.map(nuevaMediana)$

Note que a cada mediana vieja se le aplica la la función nuevaMediana, por consiguiente

$\equiv$ $(mA_0,\ ...,\ mA_n)$ ♦

Función actualizarPar

Esta función presenta la misma lógica de su versión secuencial, con la única diferencia de que esta utiliza colecciones paralelas, por lo que su argumentación es la misma, por consiguiente, se da por hecho su buen funcionamiento.

Función hayConvergenciaSeq

Preliminares.

A continuación se tendrá en cuenta la siguiente función, para la cual se da por hecho su buen funcionamiento, ya que fue proporcionada por el docente:

Funcion distanciaAlCuadrado.

$\forall\ p, that \in\ Punto: p.distanciaAlCuadrado(that) = d^2_{p-that}$, donde d_{p-that} es la distancia entre p y $that$.

A continuación se demostrará que:

$\forall\ eta \in\ Double,\ medianasViejas, medianasNuevas \in Seq[Punto]$:

$hayConvergenciaSeq(eta,\ medianasViejas,\ medianasNuevas) =$

$hayConvergenciaSeq(eta,\ (mV_0,\ ...,\ mV_n),\ (mN_0,\ ...,\ mN_n)) = Bool$, donde $Bool$ sera $True$, si hay convergencia, o de lo contrario será $False$.

Demostración.

$hayConvergenciaSeq(eta,\ (mV_0,\ ...,\ mV_n),\ (mN_0,\ ...,\ mN_n))$

$\equiv$ $val\ l\ =\ medianasViejas.length$

$!\,(for\ (i\ <-\ 0\ until\ l)\ yield$

$medianasViejas(i).distanciaAlCuadrado(medianasNuevas(i))\ <\ (eta\ *\ eta)$

$).contains(false)$

$\equiv$ $val\ l\ =\ medianasViejas.length$

$!\,(for\ (i\ <-\ 0\ until\ l)\ yield$

$d^2_{mV_i - mN_i}\ <\ eta^2)$

$).contains(false)$

Note que se verifica que cada distancia entre la mediana vieja y la nueva sea menor a eta, por lo que si todas son menores, hay convergencia, de lo contrario no habrá.

$\equiv$ bool ♦

Función hayConvergenciaPar

Esta función presenta la misma lógica de su versión secuencial, con la única diferencia de que esta utiliza colecciones paralelas, por lo que su argumentación es la misma, por consiguiente, se da por hecho su buen funcionamiento.

Función kMedianasSeq

A continuación se demostrará que:

$\forall\ puntos,\ medianas \in Seq[Punto]\ ,\ eta \in Double$: $kMedianasSeq(puntos,\ medianas,\ eta)$

$=\ clusters$, donde $clusters \in Seq[Punto]$ son las medianas correspondientes a aplicar el algoritmo kmeans correctamente, es decir hasta que se cumpla el criterio de convergencia.
Para ello se mostrará como la función $kMedianasSeq$ sigue todos los pasos del algoritmo kmeans(Inicialización, Clasificación, Actualización y Convergencia).

Demostración.

$kMedianasSeq(puntos,\ medianas,\ eta)$

La fase de $Inicialización$ se realiza aleatoriamente antes de ejecutar la función con ayuda de las funciones generarPuntosSeq y inicializarMedianasSeq, las cuales fueron proporcionadas por el docente, por lo que se da por hecho su buen funcionamiento.

$\equiv$ $//Clasificación$

$val\ clasif\ =\ clasificarSeq(puntos,\ medianas)$

Note que se utiliza la función $clasificarSeq$, la cual ya fue demostrada.

$\equiv$ $//Actualización$

$val\ medianasNuevas\ =\ actualizarSeq(clasif,\ medianas)$

Note que se utiliza la función $actualizarSeq$, la cual ya fue demostrada.

$\equiv$ $//Convergencia$

$if(hayConvergenciaSeq(eta,\ medianas,\ medianasNuevas))$

$medianasNuevas$

$else\ kMedianasSeq(puntos,\ medianasNuevas,\ eta)$

Note que se utiliza la función$hayConvergenciaSeq$, la cual ya fue demostrada.

Por lo que, si hay convergencia se retornan las $medianasNuevas$, las cuales serán los clusters finales, de lo contrario, se realiza un llamado recursivo con las $medianasNuevas$ hasta que haya convergencia. ♦

Función kMedianasPar

Esta función presenta la misma lógica de su versión secuencial, con la única diferencia de que esta utiliza colecciones paralelas, por lo que su argumentación es la misma, por consiguiente, se da por hecho su buen funcionamiento.

Casos de Prueba

Por favor revise los casos de prueba en el código fuente, muchas gracias.

En cuanto a los casos de prueba, se eligieron un rango de puntos para los cuales el algoritmo no tardará demasiado y su revisión fuera sencilla en algunos casos. Cabe resaltar que, puede existir una pequeña diferencia entre los resultados obtenidos por parte de las funciones kMedianasSeq y kMedianasPar en casos donde el número de puntos es muy alto, sin embargo, dicha diferencia es demasiado mínima como para considerar que alguna de las funciones no funciona correctamente y además, a la hora de graficar, dichos resultados son totalmente iguales. Aun así, no se puede concluir que los casos de prueba sean suficientes para reforzar la corrección de cada programa, ya que habría que considerar infinidad de casos con muchos puntos y diversa cantidad de clusters.

Evaluación Comparativa

A continuación se presentan las tablas con los resultados de tiempo de ejecución del algoritmo kMedianas según su versión(secuencial o paralela), cantidad de puntos y el número de clusters a calcular (se tomó eta = 0.01 para todos los casos).

1000 Puntos			
	Tiempo [ms]		
Clusters	Secuencial	Paralelo	Aceleración
2	0,201	0,802	0,251
4	0,228	0,852	0,268
8	0,155	1,198	0,129
16	0,212	0,841	0,252
32	0,684	1,039	0,658
64	0,718	1,619	0,443

Como se puede apreciar, para una cantidad de 1000 puntos la versión paralela no consigue aceleración con respecto a su versión secuencial para ninguna cantidad de clusters.

10000 Puntos			
	Tiempo [ms]		
Clusters	Secuencial	Paralelo	Aceleración
2	1,34	3,29	0,407
4	1,25	2,07	0,604
8	1,67	1,63	1,025
16	1,98	1,45	1,366
32	3,58	1,99	1,799
64	5,68	9,78	0,581

Si se aumenta la cantidad de puntos, se puede apreciar una pequeña aceleración por parte de la versión paralela con respecto a la versión secuencial cuando los clusters son igual o mayores a 8, sin embargo, dicha aceleración es muy baja en la mayoría de los casos como para considerarla relevante.

100000 Puntos			
	Tiempo [ms]		
Clusters	Secuencial	Paralelo	Aceleración
2	19,35	15,67	1,235
4	16,71	10,68	1,565
8	26,18	12,15	2,155
16	19,96	10,08	1,980
32	35,66	13,75	2,593
64	43,67	20,32	2,149

Cuando la cantidad de puntos es 100000, ya se puede notar una aceleración considerable por parte de la versión paralela, la cual tiende a ser el doble con respecto a la versión secuencial en la mayoría de los casos.

1000000 Puntos			
	Tiempo [ms]		
Clusters	Secuencial	Paralelo	Aceleración
2	156,82	154,11	1,018
4	253,23	134,74	1,879
8	343,57	195,45	1,758
16	443,53	212,12	2,091
32	417,67	206,05	2,027
64	1113,59	403,24	2,762

Finalmente, cuando se aumenta la cantidad de puntos a un millón, al igual que el caso anterior, la aceleración obtenida por parte de la versión paralela es bastante notoria en la mayoría de los casos, en especial cuando el número de clusters es alto.

En conclusión, se puede decir que es aconsejable utilizar paralelismo sólo cuando la cantidad de puntos es alta, mínimo 10000, recomendado 100000, ya que solo en estos casos la versión paralela obtiene una ventaja considerable de tiempo respecto a la versión secuencial, de lo contrario, no es aconsejable.