Proyecto con Python / MPI4PY

Objetivos

Que el estudiante profundice su aprendizaje sobre:

1. El diseño de programas paralelos, eficientes y "escalables" basados en procesos y memoria distribuida usando Python y MPI4PY.
2. La implementación de un programa que mezcla paralelización por datos y tareas.
3. La depuración sistemática de un programa paralelo.
4. La evaluación del desempeño de un programa paralelo utilizando las métricas: "tiempo pared", "aceleración" y "eficiencia" para mejorarlo.

Descripción del problema

El "agrupamiento de datos" o "análisis de grupos de datos" es una técnica básica de la minería de datos (https://es.wikipedia.org/wiki/An%C3%A1lisis_de_grupo) utilizada en general en el "aprendizaje automático". Existen muchas técnicas, o familias de algoritmos, para el agrupamiento de datos. Una familia de algoritmos se basa en la identificación de "centroides" para construir una clasificación de los datos que cumpla con las características de una "clase de equivalencia" en donde todos los datos pertenecen a una y sólo una clasificación. Por otro lado, la inteligencia de enjambres es una familia de algoritmos de la inteligencia artificial (https://es.wikipedia.org/wiki/Inteligencia_de_enjambre) que busca emular el comportamiento de los insectos sociales para la resolución de problemas que típicamente son de optimización. El "agrupamiento de datos" puede verse como un problema de optimización en el que se busca MINIMIZAR las diferencias entre los datos de una clase y MAXIMIZAR la diferencia entre los datos de dos diferentes clases. Por esta razón, la inteligencia de enjambres se ha usado para resolver el problema de "agrupamiento de datos".

A la fecha se han estudiado muchos algoritmos de inteligencia de enjambres y se han aplicado con diferentes grados de éxito al problema del "agrupamiento de datos". Los algoritmos más conocidos de la inteligencia de enjambres son "optimización basada en colonias de hormigas" ("ant colony optimization" o ACO) y la "optimización basada en enjambres de partículas" ("particle swarm optimization" o PSO). Más recientemente en 2005 se publicó el algoritmo "optimización basada en enjambres de gusanos luminiscentes" ("glowworm swarm optimization" o GSO). Este algoritmo se inspira en el comportamiento de los gusanos que, al igual que las luciérnagas, tienen la capacidad de iluminarse sin generar calor (ver: https://www.youtube.com/watch?v=nsd9HmzIwRQ). Este algoritmo ha capturado la atención de los investigadores porque resuelve el problema de optimización identificando varios valores óptimos en el espacio de búsqueda, lo cual representa una ventaja en relación con otros algoritmos que se centran en la identificación de un único valor óptimo, el global o uno cercano. Aplicado específicamente al "agrupamiento de datos" el GSO se caracteriza por:

1. Dada una colección D de datos x_i, i = 1...n. Los datos se representan como puntos de un espacio con d dimensiones.
2. La búsqueda de las clases se basa en m gusanos g_j, j = 1...m.

3. Por medio de los gusanos el algoritmo trata de identificar k centroides, { c_1, c_2, … c_k} que dividan D en k grupos {C_1, C_2, … C_k}, tales que:
3.1 $U_{i=1...k}$ C_i = D.
3.2 $\cap_{i=1...k}$ Ci = {}.

4. Cada gusano g_j tiene cinco propiedades:
4.1 L_j o nivel de "luciferina" (https://es.wikipedia.org/wiki/Luciferina_(mol%C3%A9cula)),
4.2 posición p_j, que representa un punto en el espacio d-dimensional donde se representa D,
4.3 valor de adaptación F_j obtenido mediante la función objetivo EQ#9,

4.4 conjunto cubierto cr_j, que representa un subconjunto de datos (D) representados por el gusano,
4.5 intra-distancia $intraD_j$, o sumatoria de las distancias de cada cr_{ji} en cr_j con respecto a g_j.

5. En 2014 Ibrahim Aljarah y Simone A. Ludwig proponen un algoritmo serial que aplica el GSO al "agrupamiento de datos" y que **deberá usted transformar en paralelo e implementarlo usando mpi4py**:
5.1 Se crean m gusanos los cuales se inicializan de la siguiente forma:
5.1.2 nivel L_j idéntico para todos L_o,
5.1.3 posición p_j al azar distribuida uniformemente,
5.1.4 valor de adaptación F_j igual a cero,
5.1.5 el valor constante r_s se inicia con r_o.

5.2 Para cada gusano g_j, se extraen (mediante r_s) los datos cr_{ji} que le serán asociados en la primera iteración y se calcula $intraD_j$ (EQ#8).
5.3 Se seleccionan los primeros centroides candidatos (CC) como el conjunto de los k gusanos cuyo cr_j (o datos asociados) sea más grande, es decir, que aquéllos que tengan la mayor cardinalidad $| cr_j |$. Estos centroides candidatos también deben estar muy separados entre sí, la distancia siempre deberá ser mayor a la constante r_s.
5.4 Con base en CC, se calcula la "Sumatoria de Errores al Cuadrado" (Sum Squared Errors o SSE), EQ#6.
5.5 Con base en CC, se calcula la inter-distancia entre centroides candidatos o InterDist, EQ#7.

5.6 Se itera mientras $| CC |$ sea mayor a un valor dado o mientras no se haya cumplido cierta cantidad mínima de iteraciones, para buscar el conjunto de gusanos óptimos que representen a los centroides:
5.6.1 para cada g_j se actualiza F_j, mediante EQ#9, tomando los valores actuales de SSE, InterDist y cr_j,
5.6.2 para cada g_j se actualiza L_j, su nivel de "luciferina" mediante EQ#1,
5.6.3 para cada g_j se determina su nuevo vecindario de gusanos (OJO no se trata de cr_j) mediante la EQ#2,
5.6.4 para cada g_j se actualiza la probabilidad de vecindario mediante EQ#3 para encontrar el mejor vecino,
5.6.5 cada g_j se mueve hacia su mejor vecino actualizando p_j mediante EQ#4,
5.6.6 para cada g_j se actualiza cr_j y por ende $| cr_j |$. Si el cr_j para algún g_j es igual a vacío, se descarta del enjambre. Se debe verificar que las condiciones 3.1 y 3.2 se cumplen para la partición basada en los cr_j,
5.6.7 para cada g_j se actualiza $intraD_j$ mediante EQ#8,
5.6.8 se actualiza CC escogiendo los gusanos que tiene los mayores valores F_j, y manteniendo una distancia mayor a r_s,
5.6.9 se actualizan los valores de SSE (EQ#6), InterDist (EQ#7),
5.6.10 para cada g_j se actualiza rd_j mediante EQ#5, aunque los autores asumen $rd_j == r_s$, para todos los gusanos,
5.6.11 se vuelve a 5.6.1.

Las ecuaciones: en todos los casos Distance(x,y) representa la distancia euclideana entre dos puntos con d dimensiones. Ver: https://es.wikipedia.org/wiki/Distancia

$$L_j(t) = (1-\rho)L_j(t-1) + \gamma F(p_j(t)) \quad (1)$$

$$z \in N_j(t) \;\; iff \;\; Distance_{jz} < rd_j(t) \;\; and \;\; L_z(t) > L_j(t) \quad (2)$$

$$Prob_{jz} = \frac{L_z(t) - L_j(t)}{\sum_{k \in N_j(t)} L_k(t) - L_j(t)} \quad (3)$$

$$p_j(t) = p_j(t-1) + s\,\frac{p_z(t) - p_j(t)}{Distance_{jz}} \quad (4)$$

$$rd_j(t) = min\{rs, max[0, rd_j(t-1) + \beta(nt - |N_j(t-1)|)]\} \quad (5)$$

$$SSE = \sum_{j=1}^{k}\sum_{i=1}^{|C_j|}(Distance(x_i, c_j))^2 \quad (6)$$

$$InterDist = \sum_{i=1}^{k}\sum_{j=i}^{k}(Distance(c_i, c_j))^2 \quad (7)$$

$$intraD_j = \sum_{i=1}^{|cr_j|} Distance(cr_{ji}, g_j) \quad (8)$$

$$F1(g_j) = \frac{\frac{1}{n}|cr_j|}{SSE \times \frac{intraD_j}{\max_j (intraD_j)}} \quad (9)$$

Los parámetros de las ecuaciones

símbolo	aparece en	nombre	etiqueta por consola	valor del parámetro	significado
ρ	EQ#1	rho	-r	0.4	Tasa constante de decremento de la "luciferina".
γ	EQ#2	gamma	-g	0.6	Fracción constante de incremento de la "luciferina".
s	EQ#4	ese	-s	0.03	Distancia constante en que se mueven los gusanos.
r_s	EQ#5	ere-ese	-i	??	Rango de cobertura de un gusano para incluir datos asociados.
L_o	inicialización	ele-cero	-l	5.0	Valor inicial de "luciferina" en los gusanos.
r_o	inicialización	ere-cero	N/A	N/A	Valor incial de la constante r_s. Se trabajará sólo con r_s.

La colección de datos por agrupar

Para que un programa pueda jugar inteligentemente al póker, primero requiere: 1) clasificar cuál es la mano que le han entregado y luego 2) valorarla contra las posibilidades de sus contendores. El problema de clasificar una mano de póker resulta muy complejo porque existen diez clases de manos de póker y cada una con muchas posibles permutaciones. Por eso sólo habiendo sido entrenado previamente, un programa sería capaz de clasificar una mano de póker correctamente. En este proyecto nos concentraremos exclusivamente en el problema de clasificación.

Robert Cattral y Franz Oppacher han analizado este problema y han generado dos muestras de manos, una con 25010 instancias o manos que se puede usar para entrenar a un programa clasificador, y la otra con un millón de instancias que se puede usar para poner a prueba al programa y determinar si ha logrado representar correctamente el dominio total que tiene 311875200 instancias. A continuación una tabla con el detalle de las manos, permutaciones y probabilidades correspondientes con respecto al dominio completo (todas las posibles permutaciones):

Mano de póker	Cantidad de manos	Cantidad de permutaciones	Probabilidad
Royal Flush	4	480	0.00000154
Straight Flush	36	4320	0.00001385
Four of a kind	624	74880	0.0002401
Full house	3744	449280	0.00144058
Flush	5108	612960	0.0019654
Straight	10200	1224000	0.00392464
Three of a kind	54912	6589440	0.02112845
Two pairs	123552	14826240	0.04753902
One pair	1098240	131788800	0.42256903
Nula	1302540	156304800	0.50117739
TOTALES	2598960	311875200	1.0

En el archivo "poker-hand.names" los autores describen en detalle el dominio, así como la representatividad de cada mano en cada muestra. La muestra para entrenamiento aparece en el archivo "poker-hand-training-true.data" y la muestra para pruebas en "poker-hand-testing.data". Ambos se obtuvieron del portal: http://archive.ics.uci.edu/ml/index.html .

Deberá usar la muestra de entrenamiento para que el programa encuentre los centroides. Luego podrá comprobar si el resultado es correcto mediante la muestra de pruebas.

Las salidas esperadas

El programa debe generar un archivo con las posiciones de los diez centroides finales, uno para cada clase de mano. Obsérvese que en el punto #5.6.8 del algoritmo de Aljarah y Ludwig los centroides se escogen entre los gusanos que obtienen el mayor valor como función objetivo. Por tanto la posición de un centroide final es siempre la posición final de algún gusano cuya función objetivo estuvo entre los 10 más altos valores (porque se requieren 10 clases en este problema). El programa que aprende no debería terminar antes de haber reducido la cantidad de gusanos a 10, siempre que las condiciones #3.1 y #3.2 se cumplan.

Si el programa funciona correctamente, mediante las posiciones de cada uno de los centroides finales, debería ser posible la clasificación correcta de la muestra de prueba:

0: Mano nula, 501209 instancias.
1: One pair, 422498 instancias.
2: Two pairs, 47622 instancias.
3: Three of a kind, 21121 instancias.
4: Straight, 3885 instancias.
5: Flush, 1996 instancias.
6: Full house, 1424 instancias.
7: Four of a kind, 230 instancias.
8: Straight flush, 12 instancias.
9: Royal flush, 3 instancias.

Para determinar a cuál clase pertenece una mano, basta con identificar la distancia euclideana mínima entre el arreglo que la representa y el conjunto de centroides final.

Criterios de evaluación

Su programa será evaluado con base en los siguientes cinco criterios básicos, simplicidad, moduralidad, eficacia, eficiencia y escalabilidad:

- Se entiende por "simplicidad" que el programa sea lo más fácil de entender, de depurar y de modificar que sea posible.
- Se entiende por "modularidad" la correcta separación de funciones entre el código de los distintos "módulos". Un módulo es una clase, una función o grupo de funciones, y la función principal "main()". Por ejemplo, con una adecuada modularidad el "main()" se encarga de la entrada y salida de datos, así como la generación de mensajes por consola dirigidos al usuario. Las clases NO deben encargarse de las funciones del "main".
- Se entiende por "eficacia" que el programa cumpla con el objetivo de simular el proceso infeccioso correctamente con base en las reglas anteriores.
- Se entiende por "eficiencia" que el programa cumpla con el objetivo en el menor "tiempo pared" posible.
- Se entiende por "escalabilidad" que:
  - el código se adapte automáticamente mediante la función "comm.size()" a la cantidad de procesos indicada por el usuario. Suponga que la cantidad de procesos por núcleo oscila en el rango de 2 a 5, parte de su trabajo será encontrar el óptimo para su programa,
  - la "eficiencia" mejora cuando se agregan más procesos o procesadores, lo cual se mide mediante la "aceleración" y "aceleración por proceso o procesador", aunque en algún momento inevitablemente se cumpla la ley de Amdahl.

La evaluación de la "aceleración" y la "aceleración por proceso o procesador" (escalabilidad) será **comparativa**, lo que significa que su puntaje se establecerá en comparación con los demás trabajos presentados en el grupo:

1. La eficiencia se medirá en términos de qué tan rápido es su programa con base en la comparación del "tiempo pared" en Kabrè con los demás trabajos de su grupo.
2. La escalabilidad se medirá en términos de si su programa mejora significativamente (aceleración o "speedup") cuando se incrementa la cantidad de procesos o procesadores que variará en {16, 32}, así como de cuánto mejora la aceleración por proceso o procesador.

Dado que hay muchos factores que intervienen en cada ejecución de un programa, la única forma de comparar las mediciones o métricas especificadas de un programa con las de otros es mediante promedios simples: "promedio de tiempo pared", "promedio de aceleración" y "promedio de aceleración por proceso". Por tanto usted deberá adjuntar a la entrega de su trabajo un archivo pdf con la siguiente tabla a efecto de que su programa pueda ser debidamente comparado con los demás.

16 procesadores (1 proceso x núcleo)	**32 procesadores (1 proceso x núcleo)**		
cm == 25010	cm == 25010		
Promedio de tp_1	p-tp_2	p-ac	p-ac-x-prc
Desviación estándar de tp_1	de-tp_2	de-ac	de-ac-x-prc

Donde todos son valores en segundos:

- p-ac, es el promedio simple de las aceleraciones,
- de-ac, es la desviación estándar de las aceleraciones,
- p-ac-x-prc, es el promedio de las aceleraciones por procesador.
- de-ac-x-prc, es la desviación estándar de las aceleraciones por procesador.

En la tabla anterior no se varía la cantidad de procesos por núcleo porque se supone que es un proceso por procesador. El asistente a cargo validará la tabla aportada y asignará puntaje[1] a cada trabajo, en los rubros de eficiencia y escalabilidad, ordenando la lista de valores correspondiente de menor a mayor y luego dividiéndola en mínimo dos (si las desviaciones estándar son pequeñas) y máximo cuatro grupos (si las desviaciones son grandes) con resultados similares, luego asignará:

9 o 10: para los mejores promedios,
7 u 8: para los promedios buenos,
6: para los promedios regulares, y
menos de 6: para los promedios malos.

Tabla de Evaluación

Criterio	%
Simplicidad	5
Modularidad	5
Eficacia	35
Eficiencia	35
Escalabilidad	20
Hasta 10 puntos por un reporte de errores detallado **en caso que no haya funcionado** el programa	

Notas importantes:
1. Si el programa NO funciona, no obtendrá más de 5/100 puntos, o 15/100 si incluye un reporte de errores detallado. En este caso obtendrá cero puntos en todos los rubros.
2. Este proyecto deberá realizarse idealmente y a lo más en parejas. NO SE ACEPTARÁ NINGÚN TRABAJO ELABORADO POR MÁS DE DOS PERSONAS.
3. Cada hora de atraso en la entrega se penalizará con -1/100, lo que se aplicará a la nota obtenida.
4. A TODOS LOS ESTUDIANTES INVOLUCRADOS EN UN FRAUDE SE LES APLICARÁ EL ARTÍCULO #5 INCISO C DEL "Reglamento de Orden y Disciplina de los Estudiantes de la Universidad de Costa Rica".
5. NO SUBA ningún otro archivo que no sea de código fuente (*.py) o de datos para evitar la transmisión de virus.

1 Para validar los datos el asistente realizará al menos una prueba.