Reporte práctica seis: Sistemas multiagente

José Anastacio Hernández Saldaña
Posgrado de Ingeniería de Sistemas
1186622
jose.hernandezsld@uanl.edu.mx

17 de septiembre de 2017

Resumen

Este es un reporte sobre la práctica seis con respecto al tema de sistemas multiagente que se realizó en la clase de Simulación de sistemas, cómputo paralelo en R.

1. Tarea: Implementación paralela de un sistea multiagente

Un sistema multiagente es un sistema computacional donde interactúan varios agentes dentro de un ambiente o entorno, los agentes realizan acciones en el ambiente y a partir del estado del ambiente el agente realiza alguna acción. Este tipo de sistemas son descentralizados, es decir, los agentes no siguen órdenes de una unidad central, sino que cada agente es autónomo.

Para esta tarea se realizó una simulación de una epidemia de una cantidad n de agentes, donde cada agente podría estar en alguno de los siguientes estados: sano, infectado e inmune. El agente estaba sano si no se había contagiado al estar cerca de algún otro agente infectado, si estaba a una distancia d del infectado tal que era menor a un umbral r, hay una probabilidad $p_i = \frac{r-d}{r}$ de infectarse en otro caso $p_r = 0$, si el agente estaba infectado, tenia una probabilidad p_r de recuperarse y volverse inmune. Cada agente tenia una velocidad horizontal y vertical dx y dy respectivamente, asi que en cada paso de la simulacion los agentes avanzaban en funcion de su posicion inicial y su velocidad y se volvian a hacer los calculos para definir su estado, todo esto dentro de un espacio de dimensiones $l \times l$, el cual esta en forma de un torus o dona, ya que al exceder alguno de los bordes, el agente aparece en borde contrario.

1.1. Diseño del Experimento

Para el experimento se tomó como base el código de la página del curso, donde se encuentra la simulación ya programada con los siguientes parámetros, umbral de infección $r = 0.1$, probabilidad de recuperación $p_r = 0.2$, con una dimensión de $l = 1.5$, se tienen una cantidad de agentes de $n = 50$, estos se creaban posiciones x, y con velocidades dx, dy tomadas de una distribucion uniforme tal que $0 \leq x \leq l$ y $\frac{-l}{30} \leq dx \leq \frac{l}{30}$, de manera similar para y y dy. El estado inicial de los agentes era asignado con una probabilidad $\alpha_i = 0.05$ de comenzar infectado y una probabilidad $1 - \alpha_i$ de comenzar sano.

Este código no se encuentra paralelizado y cuenta con varias opciones para paralelizarse y optimizarse identificadas, que son las siguientes:

1. Asignación de estado inicial. La asignación se hace de manera iterativa para cada uno de los agentes puede hacerse de manera paralela.
2. Cálculo de la distancia de un agente y los agentes cercanos. El cálculo de distancia en cada agente hacia todos los demas puede hacerse de manera paralela y puede mejorarse si se reduce la cantidad de agentes a comparar.
3. Asignación de posiciones y velocidades iniciales. Esta asignación también se hace de manera iterativa para cada agente, dando la opción de paralelizarse.

Como el cálculo 2 se hace dentro del código de la asignación 1, estas pueden tener problemas si ambas se paralelizan ya que paralelizar procedimientos paralelos los hace competir por los recursos del procesador, asi que se decidió paralelizar la asignación de estados, y optimizar el cálculo de la distancia. Otro enfoque tambien considerado fue utilizar un método de linea de barrido para las asignaciones 1, 3 y que optimiza el cálculo 2 ya que en cada iteracion de la linea de barrido considera solo aquellos agentes que se encuentran a una distancia r de la linea, evitando calcular la distancia de

agentes mas allá del umbral. Aunque el algoritmo de linea de barrido puede paralelizarse en el cálculo de la distancia 2 no daria mucha mejora ya que la cantidad de agentes es pequeña y la paralilzacion de las asignaciones denpende de la iteracion anterior por lo que no podria paralelizarse. Pero tomando en cuenta el enfoque de linea de barrido puede hacerse una version paralela de este codigo sí en lugar de llevar un regitro de quienes estan dentro de la linea de barrido e ir actualizandolo cada iteracion, puede para cada punto de la linea de barrido calcularse los vecinos cercanos.

Por lo que se tuvieron cuatro algoritmos para comparar, que se clasificaron de la siguiente manera

1. Algoritmo orginal
2. Algoritmo original paralelizado
3. Algoritmo de linea de barrido
4. Algoritmo basado en linea de barrido paralelizado

Se decidio comprar los cuatro algoritmos para poder comparar si utlizar un buen algoritmo, como lo es el de linea de barrido, contra una implementacion paralela del algorimo original, y una implementacion paralela basada en la linea de barrido, el algoritmo original se tomo como cora superior de lo que podriamos esperar del desempeño de los algorimos, el código de los cuatro algoritmo esta dentro del repositorio git del curso.

Para comenzar la experimentación, se utilizó una computadora con las siguientes especificaciones, Procesador Intel Core i7-4790 CPU @ 3.6GHz $\times$ 8 y Memoria RAM de 24 GB. Utilizando solamente los cuatro núcleos físicos disponibles. Para cada experimento se hicieron 30 replicas.

esta gráfica en la figura 1.

Figura 1: Gráfica de la función $f(x) = \frac{1}{e^x + e^{-x}}$

1.2. Resultados

2. Extra Uno: Aproximación del valor de π

2.1. Diseño del Experimento

2.2. Resultados

3. Extra Dos: Pruebas estadísticas a las series sucesivas

4. Conclusiones