Joel Chacón Castillo

Resolución de sistemas de ecuaciones diagonalmente dominante con enfoque iterativo de Monte Carlo en paralelo

1 Introducción

En este trabajo se implementa el algoritmo de Monte Carlo para aproximar la evaluación de un sistema de ecuaciones diagonalmente dominante. Estos métodos son utilizados cuando se necesita una solución no muy exacta. El problema de aproximar una evaluación se encuentra cuando se desea encontrar la inversa de una matriz, esta matriz inversa es muy útil cuando se necesita encontrar matrices especiales precondicionadas utilizadas para acelerar la convergencia de los métodos básicos iterativos.
Existen varias ventajas de estos algoritmos dado que los algoritmos de Monte Carlo son algoritmos paralelizables. Los algoritmos de Monte Carlo son muy eficientes cuando el problema a considerar es muy largo.
Una de las más imporantes ventajas de estos algoritmos es que pueden ser utilizados para evaluar sólo un componente de la solución de algúna forma lineal.
Se aclara que el documento presente no explica toda la teoría sobre el proceso discreto de Markov y el método iterativo de Monte Carlo, dicha información se puede consultar en las referencias. También se explica que el pseudocódigo que se muestra es el que se programó, el cual está generado en base a las referencias (el pseudocódigo es un híbrido dadas las referencias).

2 Complejidad

Los métodos directos como es el caso de Gauss y eliminación de Gauss Jordan demoran un tiempo de

$$T(m) = O(m^3) \tag{1}$$

Mientras que los métodos iterativos, como es el caso de Jacobi, Gauss-Seidel y varias técnicas de relajación demoran un tiempo de

$$T(m,k) = O(m^3 k) \tag{2}$$

Si existen k iteraciones. Los métodos directos también son paralelizables, pero estos necesitan un número largo de procesadores p.
Las técnicas de Monte Carlo demora un tiempo de

$$T(m,k,n) = O(m^2 kn) \tag{3}$$

o menos, si existe un promedio de n muestras en caminos aleatorios que en promedio tienen una longitud k. En comparación con los métodos iteraticos se

tiene n reemplazando m. Entonces cuanto menor sea n de m $n < m$ se está más lejos de ser más eficiente que los métodos clásicos. La estimación presentada muestra que los algoritmos de Monte Carlo son prefereibles cuando se necesita tener una estimación de la matriz inversa. El problema es muy imporante cuando se desean encontrar aproximación de precondicionadores inversos para factorizar sistemas dispersons "sparse". Si un sistema con p procesadores está disponible entonces las técnicas de Monte Carlo demoran un tiempo de

$$T(m,k,n,p) = O(\frac{m^2}{p}kn) \tag{4}$$

Claramente la razón de convergencia respecto a al promedio de longitud k de la cadena de markov) depende del radio espectral de la matriz. Si el radio espectral es pequeño los algoritmos son considerados lejos de ser eficientes.

3 Ulam-von Neumann Scheme

- Ulam-von Neumman Scheme, construye cadenas de Markov para probar la serie de Neumann.
- Considerando un sistema lineal $x = Hx + b$
- Neumann Series $I + H + H^2 + H^3 + ...$
- Matriz de transición P $Pij >= 0$ $\sum_j P_{ij} <= 1$ $H_{ij} \neq 0$
- Probabilidad de terminación $T_i = i - \sum P_{ij}$
- Camino aleatorio $\gamma : r_0 -> t_1 -> r_2 -> ... -> r_k$
- Entonces: $X(\gamma) = \frac{H_{r_0 r_1} H_{r_1 r_2} ... H_{r_{k-1} r_k}}{P_{r_0 r_1} P_{r_1 r_2} ... P_{r_{k-1} r_k}} b_{rk}/T_{rk}$ el cual es un estimador insesgado de x_{r0}

La forma presentada se mejora con el método de "Monte Carlo Almost Optimal (MAO)" donde

- Una matriz de transición P alternativa $P_{ij}) \frac{|H_{ij}|}{\sum_j |H_{ik}}$
- Terminación adaptativa, $W_0 = 1, W_k = W_{k-1} \frac{H_{r_{k-1} r_k}}{P_{r_{k-1} r_k}}$
- Ofrece un estimador alternativo el cual es más exacto $X(\gamma) = \sum_k W_k b_{k r_k}$

Algorithm 1 Monte Carlo Almost Optimal (MAO)

Require: Matrix A, vector b, constant ϵ, the relaxation parameter $\gamma \in (0, 1]$

1: Compute the matrix L

$$\{l_{i,j}\}_{i,j=1}^{m} = \begin{cases} 1-\gamma & \text{when } i = j; \\ -\gamma \frac{a_{ij}}{a_{ii}} & \text{when } i \neq j. \end{cases}$$

2: Compute the vector f $f_i = \gamma \frac{b_i}{a_{ii}}$, $i = 1, ...m$.
3: Compute the vector $lsum$: $lsum(i) = \sum_{j=1}^{m} |l_{ij}|$ for $i = 1, 2, ..., m$
4: **for** Perform one trajectory **do**
5: $X = 0$, $W = 1$, $index = i$
6: Calculate $X = X + W * f_{index}$
7: **while** $|W| > \epsilon$ **do**
8: Generate an uniformly distributed random number $\xi \in [0, 1)$
9: set $j = 1$
10: **while** $\xi > \sum_{s=1}^{j} P_{index,s}$ **do**
11: $j = j + 1$
12: $W = W * sign(h_{index,j}) * hsum(index)$
13: $index = j$
14: Calculate the mean value based on n independent trajectories, step 4 and get solution: $\overline{X}$

4 Modelo en paralelo

Se implementaron tres técnicas de paralelización:

- Paralelización de memoria compartida OpenMP.
- Paralelización de memoria distribuida MPI con tamaños predefinidos.
- Paralelización de memoria distribuida MPI, modelo maestro escalvo, forma dinámica.

En todas las implementaciones la sección de código la cual se ejecuta en paralelo es el cálculo de cada variale (línea 4 del algoritmo), es decir, dado que este método está basado en estimadores de máxima verosimilitud el cálculo de cada variable de calcula como el promedio de un conjunto de trayectorias, entonces cada variable se puede calcular como un proceso independiente el cual como ya se mencionó es un conjunto de trayectorias.
En la paralelización utilizada con OpenMP, antes de realizar el cálculo de las trayectorias, al inicio se distribuyen las variables entre los hilos, si existe un mayor número de variables que hilos cada hilo tiene un conjunto igual de variables.
El modelo implementado para la paralelización de memoria distribuida es del tipo maestro esclavo, donde hay un proceso maestro "ROOT" el cual administra y distribuye todos los procesos hijo, en este caso se contemplan dos formas

para paralelizar las treyectorias del algoritmo, de asignación de procesos hijos de forma estática y dinámica. La forma estática realiza un precalculo de las variables que llevará cada proceso, de esta forma se envía la información inicial a cada proceso, cada proceso realiza el cálculo de n variables, hasta que el proceso termina y envía el resultado al proceso maestro.
En la segunda forma no se realiza un precalculo y cada variable se asigna a cada proceso de forma dinámica, cada que existe un proceso libre se asigna la variable. En este algoritmo la forma dinámica puede ser más adecuada ya que el el tiempo para calcular cada trayectoria no se encuentra definido ya que es un proceso estocástico.
Si el número de variables es muy elevado, entonces el método estático puede ser más adecuado ya que disminuye notablemente el interbloqueo de las comunicaciones entre el proceso maestro y los hijos.

5 Resultados

Se desarrolló un programa el cual genera matrices diagonalmente dominantes, el número de variables es de 350 para todos los casos, se ejecutó el programa de 1 a 19 procesos, y se midió el tiempo, se puede observar el SpeedUp en la figura de la sección (5), se puede observar que el método dinámico ofrece una mejor aceleración, la forma estática al inicio es muy similar a la estática, sin embargo tiende a oscilar más y con 19 procesos se considera inferior en cuanto aceleración. En la figura 2 y figura 3 se muestran los tiempos y el número de procesos que se implementaron.

6 Conclusiones

Los métodos de Monte Carlo como ya se mencionó son muy eficientes al momento de paralelizarse pero hay que analizar en que escenario se va a implementar el algoritmo, si el número de variables son muy elevadas(> 10000) y el tráfico de red es elevado, (comunicación WAN) es mejor implementar algún método en donde no se necesiten efectuar muchas comunicaciones, por otro lado si el equipo que procesa la información se encuentra cercano y existe menos tráfico de red, entonces el método dinámico es el adecuado (como es el caso del insurgente).

7 Referencias

1. A new iterative Monte Carlo approach for inverse matrix problem, I.T. Dimov*, T.T. Dimov, T.V. Gurov.

2. Revisit of Monte Carlo Methods on Solving Large-Scale Linear Systems, YAOHANG LI.

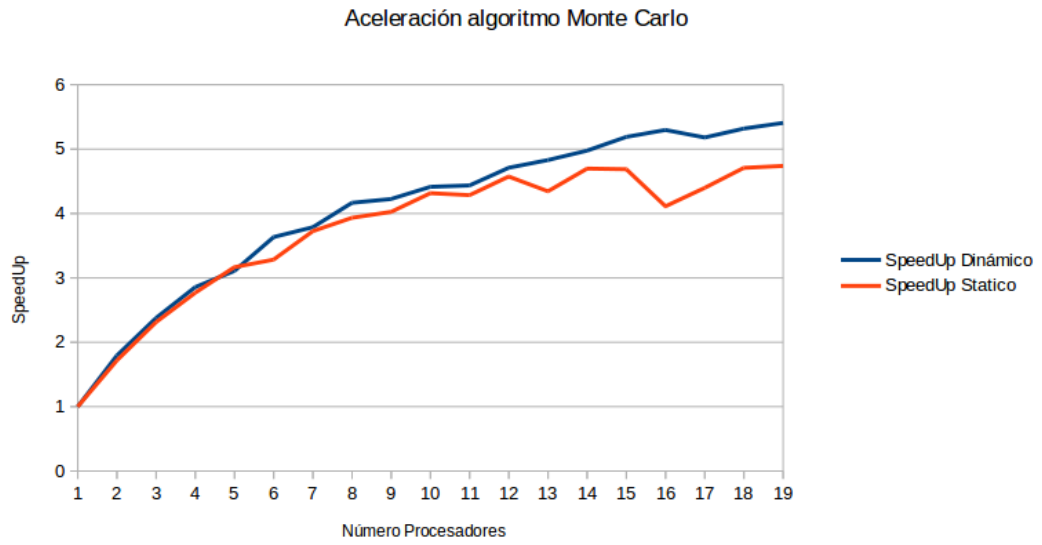

Figure 1: Análisis del SpeedUp de las dos implementaciones.

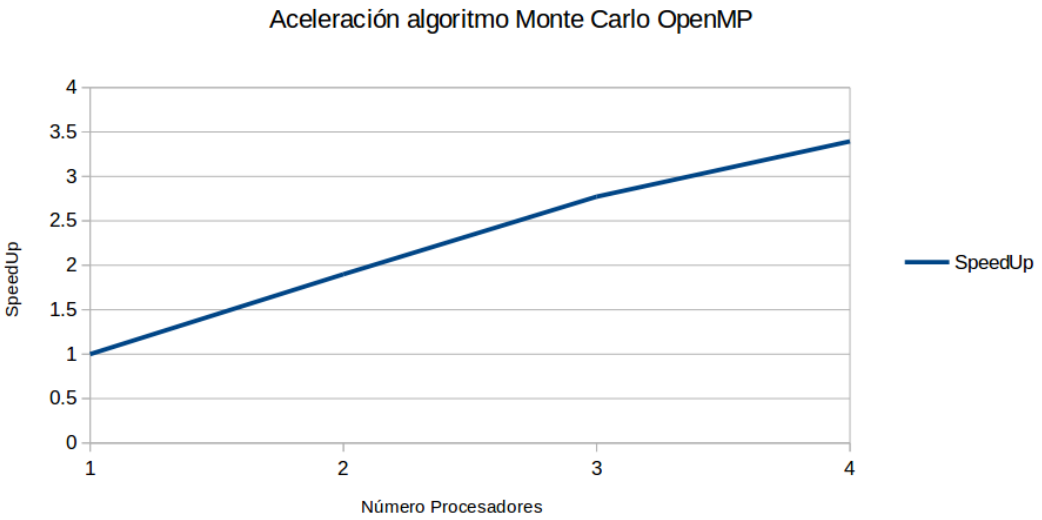

Figure 2: SpeedUp OpenMP.

Numero Procesadores	Tiempo seg.	SpeedUp Dinámico
1	9.19	1
2	5.123	1.7938707788
3	3.867	2.3765192656
4	3.217	2.8566987877
5	2.957	3.1078796077
6	2.529	3.6338473705
7	2.427	3.7865677792
8	2.206	4.1659111514
9	2.175	4.2252873563
10	2.082	4.414024976
11	2.072	4.4353281853
12	1.951	4.7104049206
13	1.903	4.8292170257
14	1.847	4.9756361668
15	1.771	5.1891586674
16	1.735	5.2968299712
17	1.774	5.1803833145
18	1.728	5.318287037
19	1.7	5.4058823529

Figure 3: Tiempo con el método dinámico.

Numero Procesadores	Tiempo seg.	SpeedUp Statico
1	9.093	1
2	5.293	1.7179293406
3	3.931	2.3131518698
4	3.284	2.7688794153
5	2.872	3.166086351
6	2.769	3.2838569881
7	2.441	3.7251126587
8	2.312	3.9329584775
9	2.259	4.0252324037
10	2.107	4.3156146179
11	2.122	4.2851083883
12	1.988	4.573943662
13	2.093	4.3444816054
14	1.936	4.6967975207
15	1.94	4.6871134021
16	2.212	4.1107594937
17	2.068	4.3970019342
18	1.931	4.7089590886
19	1.919	4.7384054195

Figure 4: Tiempo con el método estático.

Numero Proceso	Tiempo Real	SpeedUp
1	17.368	1
2	9.146	1.8989722283
3	6.264	2.7726692209
4	5.117	3.3941762752

Figure 5: Tiempo OpenMP.