Optimización del Algoritmo de Eliminación Gaussiana usando C++ y OpenMP

Andres Felipe Bayona, Aron Rene Forero Africano, Edwar Villarreal
Programación en Paralelo
Bucaramanga Colombia

Resumen – *En las matemáticas la eliminación de Gauss-Jordan o Eliminación Gaussiana es un algoritmo utilizado para determinar las soluciones de un sistema de ecuaciones lineales, encontrar matrices e inversas. El algoritmo para resolver computacionalmente la eliminación gaussiana será el algoritmo de pivoteo y la sustitución regresiva. Un problema que se presenta en la computación es el tiempo que se demoran los códigos en ejecutar, para disminuir este tiempo de ejecución son utilizadas varias técnicas, y una de ellas es OpenMP, de lo cual se hablará más adelante.*

I. Introducción

La eliminación de Gauss-Jordan, llamada así debido a Carl Friedrich Gauss y Wilhelm Jordan, es un algoritmo del álgebra lineal para determinar las soluciones de un sistema de ecuaciones lineales, encontrar matrices e inversas.

Un sistema de ecuaciones se resuelve por el método de Gauss cuando se obtienen sus soluciones mediante la reducción del sistema dado a otro equivalente en el que cada ecuación

tiene una incógnita menos que la anterior. El método de Gauss transforma la matriz de coeficientes en una matriz triangular superior. El método de Gauss-Jordan continúa el proceso de transformación hasta obtener una matriz diagonal.

Para producir un sistema equivalente usaremos el teorema de transformaciones elementales aplicado a sistemas de ecuaciones lineales, el cual nos muestra las siguientes tres operaciones:

(1) Intercambio: el orden de las ecuaciones puede cambiarse.
(2) Escalado: multiplicar una ecuación por una constante no nula.
(3) Sustitución: una ecuación puede ser reemplazada por la suma de ella misma más un múltiplo de otra ecuación.

Una forma eficiente de trabajar es almacenar todas las constantes del
Sistema lineal AX=B en una matriz de orden N * (N+1) que se obtiene añadiendo a la matriz A una columna, la columna (N+1)-ésima, en la que se almacenan los términos de B, cada fila de esta matriz, que se llama matriz ampliada del sistema y se denota por [A|B], contiene toda la información necesaria para representar la correspondiente ecuación del sistema línea:

$$[A|B] = \left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1N} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2N} & b_2 \\ \vdots & \vdots & & \vdots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} & b_N \end{array}\right].$$

II. Condiciones de entrada

Un sistema de ecuaciones lineales de la forma AX=B con N ecuaciones y N incógnitas.

Se utilizará el algoritmo de pivoteo para resolver la matriz de manera secuencial, hasta convertirla en una matriz triangular superior. Luego, se utilizará el método de sustitución para resolver la matriz triangular superior.

III. Algoritmo de eliminación Gaussiana (secuencial):

```
For i = 1:N-1
        For r = 2:N
                m = a(ri)/a(ii)
                a(ri) = 0
                for c = i+1:N+1
                   a(rc)=a(rc)+a(ic)*m
              end
       end
end
```

En el anterior pseudocódigo se muestra el algoritmo de pivoteo.

1. En el primer paso se toma el primer elemento de la diagonal principal.
2. Con el primer elemento de la segunda fila se halla el coeficiente m dividiendo tal elemento con el elemento tomado en el paso 1.
3. Como el resultado de la suma entre la segunda fila y la primera fila ya multiplicada por el coeficiente hallado en el paso 2 siempre da cero, el primer elemento de la segunda fila se sustituye por cero.
4. Se realiza la suma la segunda fila con la primera fila ya multiplicada por el coeficiente hallado en el paso 2 para cada elemento de la segunda fila.

Este proceso se realiza para cada fila hasta conseguir que la primera columna está conformada por ceros a excepción del primer elemento.
Seguidamente se realizará el mismo proceso iniciando desde la siguiente fila y siguiente columna, tomando así solo los elementos de la submatriz [i:n]x[i:n], sabiendo que i se incrementa desde 1 hasta n.

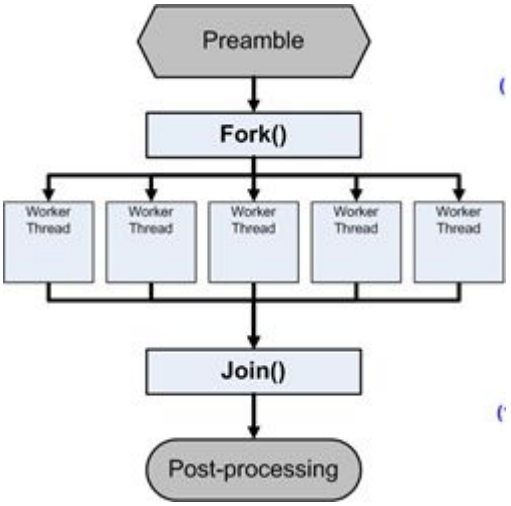
Por último se realizará un proceso de sustitución de abajo hacia arriba, para determinar los valores finales de cada incógnita.

NOTA: Todo elemento de la diagonal principal debe ser diferente de cero. Si existe algún elemento igual a cero, se debe realizar un intercambio con otra fila.

IV. Intuición previa para la programación en paralelo:

Para paralelizar el anterior proceso en base a lo aprendido, evitaremos la dependencia de datos paralelizando el proceso antes de realizar el siguiente pivoteo, porque entre cada pivoteo se modifican los valores de la diagonal principal los cuales son estrictamente necesarios para el siguiente pivoteo. Se dividirá la matriz en filas para operarlas en procesos separados, y a medida que el algoritmo se ejecuta, el número de filas y columnas a operar se irá reduciendo, disminuyendo la carga computacional.

V. ¿Por qué OpenMP?



OpenMP es una API para la programación multiproceso de **memoria compartida** en múltiples plataformas, permitiendo añadir concurrencia a los programas escritos en C, C++ y Fortran sobre la base de modelo de ejecución Fork-Join como se observa en la figura anterior.

Se decidió utilizar OpenMP porque este método de paralelización es mucho más sencillo que MPI y además de eso, el código ya paralelizado es mucho más entendible, organizado y fácil de leer con OpenMP.

En los resultados obtenidos en las pruebas se ve la diferencia entre el tiempo de ejecución del algoritmo codificado de forma secuencial y codificado de forma paralela.

Cabe notar que el algoritmo funciona con matrices cuadradas (nxn dimensiones), también que a medida que se paralelizaba el algoritmo se perdía un poco de precisión numérica, y algo muy importante, que se notó al realizar las pruebas es que, localmente la diferencia entre el tiempo de ejecución del código secuencial y el código paralelo se notaba bastante, pero a la hora de ejecutar este algoritmo en GUANE, esta diferencia no se veía tan clara.

VI. Implementación

Tanto la implementación de algoritmo de eliminación gaussiana se usó el lenguaje c++, el código sé testeo en una máquina con un procesador i5 con cuatro hilos y dos cores , y en el nodo 13 de el supercomputador guane variando el número de hilos (2, 4, 8, 16, 24), en todos los casos de generaba una matriz de números aleatorios de tamaño n x n+1 la cual variaba desde n = 100 hasta n = 3000, cada caso se testeto tres veces:

un ejemplo de la salida de ejecución es para el código paralelo con un n=1000 en el nodo número 13 de guane con 4 hilos :

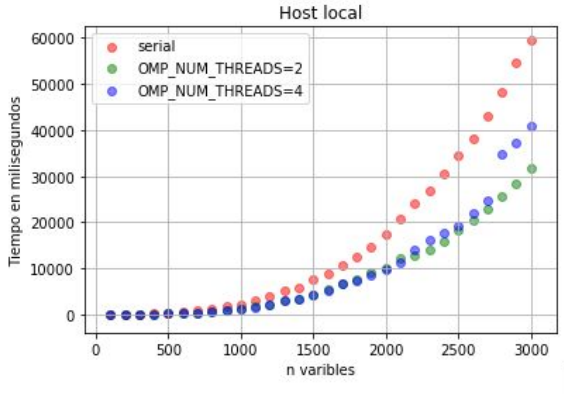
```
n=1000 hilos=4
3382.38
3422.67
3436.97
```

queda por aclarar que no se tuvieron en cuenta excepciones como sistemas de ecuaciones sin soluciones o que algún elemento de la diagonal pudiese en momento puesto que se priorizo mirar los resultados basados en el tiempo de ejecución del algoritmo.

VII. Resultados

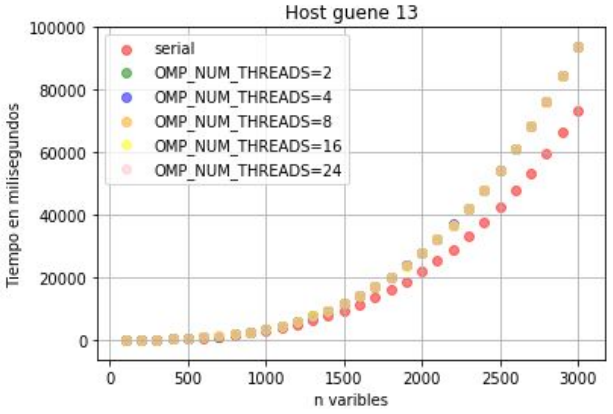
Local

en la computadora local se puede apreciar un cambio significativo en la velocidad al ejecutar código en en forma secuacien el parallela.



al mismo tiempo nos percatamos de que no hay cambios significativos en usar 2 o cuatro hilos creamos que se debe a que la quina en que se estaba ejecutando contraria al tiempo otras aplicaciones.

Guane



la ejecución en el supercomputador nos muestra resultados contradictorios puesto que el código paralelizado muestra un significante aumento en en el tiempo de ejecución.

como se puede observar la los el código en en no presenta una mejora incremental dependiendo de los nodos.

en la siguiente gráfica se puede apreciar mejor que todos los resultados del código paralelizado se solan

VIII. Conclusiones

OpenMP es un excelente herramienta que permite paralelizar codigo rápido y eficientemente.

La ejecución en guane no presenta resultados deseados hay que analizar que sucede al momento de la ejecución y confirmar que se usan eficazmente los hilos del nodo.

Por la pérdida de precisión al momento de la división, sería interesante usar una representación fraccionaria para evitar la pérdida de precisión.

IX. References

[1] S.F.McGinn and R.E.Shaw, Parallel Gaussian Elimination Using OpenMP and MPI

[2] Métodos Numéricos con Matlab John H. Mathews.