ANALISIS NODAL EN CIRCUITOS RESISTIVOS A TRAVES DE MAQUINAS DE ESTADO FINITAS

Edward Duarte, Andrés Ramírez, Nicolas Pedraza, David Orozco
Ingeniería Electrónica, Departamento de Ingeniería Pontificia Universidad Javeriana, Bogotá D.C, Colombia
edwardduarte@javeriana.edu.co
camilo_ramirez@javeriana.edu.co
pedraza_n@javeriana.edu.co
davidorozco@javeriana.edu.co

Abstract—The purpose of this project was to solve any circuit by means of a nodal analysis programmed in C, so that the value of the voltage found in each node is returned. The basis of the program consists of receiving the netlist and by means of the programming carried out to return the result of the voltage in the node through the gauss and Back Substitution process, this process is carried out by means of dynamic memory.

For the solution of the netlist, it is analyzed if it is current or resistance respectively in order to place it in the matrix, after this the input and output nodes are analyzed in order to locate on the diagonal those nodes where it belongs with its positive inverse value In the same way, its negative inverse value is located in those nodes where it is present and if it is not found, it takes the value of 0.

With this matrix obtained, the Gaussian elimination and the Back Substitution carried out by finite states continue to be realized by which the value of the voltage at the node and the time it takes to run the program are returned.

I. Introducción

El propósito planteado para este proyecto fue resolver cualquier circuito eléctrico planteado con resistencias y corrientes mediante un análisis nodal programado en lenguaje C, de manera que se retorne el valor del voltaje hallado en cada nodo.

La base del programa consiste en recibir la netlist (archivo que proporciona el nombre del elemento, nodos de entrada/salida y valores de ellos) y mediante la programación realizada devolver el resultado del voltaje en cada uno de los nodos mediante el proceso de gauss y Back Substitution. Este proceso es realizado mediante memoria dinámica para optimizar el código. Cabe resaltar que para hacer uso de memoria dinámica se requieren los apuntadores como base de este proceso.

Para la solución de la netlist, se analiza si es corriente o resistencia respectivamente, esta solución se hace efectiva con la identificación de la primera letra de los parámetros de la

netlist. Para así pasar a ubicarla en la matriz, luego de esto se analiza las corrientes de entrada y salida en cada uno de los nodos para así ubicar en la diagonal aquellas resistencias que tienen interacción en un nodo especifico. Esto se ubica con su valor inverso y positivo, de la misma forma se ubica su valor inverso negativo en aquellos nodos donde también está presente. Y si no se encuentra toma el valor de 0 en la matriz.

Con esta matriz obtenida se prosigue a realizar la eliminación gaussiana y el Back Substitution realizada en el código por estados finitos. Donde se retorna el valor del voltaje en el nodo y se identifica el tiempo de duración de la ejecución de este programa.

II. Directrices para la preparación de los manuscritos

La elaboración de este proyecto fue asesorada e instruida por el ingeniero y docente de la Pontificia Universidad Javeriana, Juan Carlos Giraldo, ingeniero electrónico, profesor de planta de la Pontificia Universidad Javeriana de Bogotá, del departamento de ingeniería electrónica, facultad de ingeniería

III. Objetivo general

Realizar un programa en lenguaje C el cual reciba un circuito eléctrico por medio de una netlist y le devuelva al usuario el valor del voltaje en los nodos del circuito.

IV. Objetivos específicos

- Realizar los procesos de gauss elimination y back Substitution por medio de estados finitos.
- Analizar un archivo txt (netlist) y resolver con los valores entregados en él.
- Acomodar de manera correcta los valores de los inversos de las resistencias en la matriz.
- Retornar los voltajes correctos de cada uno de los nodos del circuito eléctrico.

V. Algunos errores comunes

La diagonal de una matriz nodal debe ser positiva para una mayor comodidad a la hora de encontrar los voltajes- El análisis de nodos se realiza mediante las corrientes que entran al nodo no por los voltajes.

VI. Contexto

El análisis de nodos consiste en hallar los voltajes en cada uno de sus nodos por medio de sus corrientes entrantes a cada uno de los nodos. Es decir, analiza las corrientes entrantes y salientes del nodo, y despeja su valor de voltaje.
A continuación, se presenta:
- Estado del arte
- Marco Teórico
- Diagrama de flujo(anexo).
- Diagrama de estados
- Descripción código.
- Funcionamiento notas en el código.
- Conexiones.
- Conclusiones participantes

VII. Estado del arte

En la actualidad se evidencia que muchos problemas dados para ingenieros como para científicos son posibles de solucionar mediante sistemas de ecuaciones lineales, de manera que es vital la creación de programas que den una solución rápida y oportuna a dichos sistemas. De esta manera podemos evidenciar en el artículo "Using gauss - Jordan elimination method with CUDA for linear circuit equation systems" publicado en 2011, de PROCEDIA TECHNOLOGY como se presenta una guía de solución a un sistema ecuaciones lineales por medio de programación en lenguaje C. En este artículo se describe la solución del Sistema de ecuación de circuito lineal para una matriz "n x n", basado en circuitos que incluyen resistencia, fuentes de corriente independientes y fuente de voltaje DC. Por esta razón se detalla el proceso de eliminación gaussiana y eliminación hacia atrás (Back Substitution) de manera que se genere una diagonal con los resultados que se desean.[1]

VIII. Marco teórico

Análisis Nodal: El análisis de nodos, o método de tensiones nodales es un método para determinar la tensión (diferencia de potencial) de uno o más nodos.

El análisis de nodos es posible cuando todos los nodos tienen conductancia. Este método produce un sistema de ecuaciones [2]. El procedimiento de un análisis nodal es el siguiente:

1. Localice los segmentos de cable conectados al circuito. Estos serán los nodos que se usarán para el método.
2. Seleccione un nodo de referencia (polo a tierra). Se puede elegir cualquier nodo ya que esto no afecta para nada los cálculos; pero elegir el nodo con más conexiones podría simplificar el análisis.
3. Identifique los nodos que están conectados a fuentes de voltaje que tengan una terminal en el nodo de referencia. En estos nodos la fuente define la tensión del nodo. Si la fuente es independiente, la tensión del nodo es conocida. En estos nodos no se aplica la LCK.
4. Asigne una variable para los nodos que tengan tensiones desconocidas. Si la tensión del nodo ya se conoce, no es necesario asignarle una variable.
5. Para cada uno de los nodos, se plantean las ecuaciones de acuerdo con las Leyes de Kirchhoff. Básicamente, sume todas las corrientes que pasan por el nodo e iguálelas a 0. Si el número de nodos es n, el número de ecuaciones será por lo menos n-1 porque siempre se escoge un nodo de referencia al cual no se le elabora ecuación.
6. Si hay fuentes de tensión entre dos tensiones desconocidas (entre dos nodos desconocidos), unos esos dos nodos como un supernodo, haciendo el sumatorio de todas las corrientes que entran y salen en ese supernodo. Las tensiones de los dos nodos simples en el supernodo están relacionadas por la fuente de tensión intercalada.
7. Resuelva el sistema de ecuaciones simultáneas para cada tensión desconocida [3].

➢ Programación en c:

C es un lenguaje de programación (considerado como uno de lo más importantes en la actualidad) con el cual se desarrollan tanto aplicaciones como sistemas operativos a la vez que forma la base de otros lenguajes más actuales como Java, C++ o C#. [4]
Fue creado por Brian Kernighan y Dennis Ritchie a mediados de los años 70. La primera implementación de este la realizó Dennis Ritchie sobre un computador DEC PDP-11 con sistema operativo UNIX.[5]

Máquina de Estados finitos: Una Máquina de Estado Finito (Finite State Machine), llamada también Autómata Finito es una abstracción computacional que describe el comportamiento de un sistema reactivo mediante un número determinado de Estados y un número determinado de Transiciones entre dicho Estados.[6]

Método de Gauss: El método de Gauss consiste en transformar un sistema de ecuaciones en otro equivalente de forma que éste sea escalonado.[7]

Back Substitution: Sustitución hacia atrás (back -substitution): Consiste en reducir la matriz característica del sistema en una matriz triangular superior, para después sustituir una por una las variables así despejadas.[8]

Netlist: Es el término utilizado para describir un archivo generado por los programas de diseño electrónico (CAD), que contiene un listado de las conexiones o *Nets* eléctricos que existen entre los terminales de los componentes que pertenecen a un circuito esquemático o un PCB.[9]

Memoria dinámica: Es memoria que se reserva en tiempo de ejecución. Su principal ventaja frente a la estática es que su tamaño puede variar durante la ejecución del programa. (En C, el programador es encargado de liberar esta memoria cuando no la utilice más). El uso de memoria dinámica es necesario cuando no conocemos el número de datos/elementos a tratar; sin embargo, es algo más lento, ya que el tiempo ejecución depende del espacio que se vaha usar.[10]

Resistencia: es una medida de la oposición al flujo de corriente en un circuito eléctrico. Se mide en ohmios, que se simbolizan con la letra griega omega (Ω). Se denominaron ohmios en honor a Georg Simon Ohm (1784-1854), un físico alemán que estudio la relación entre voltaje corriente y resistencia.[11]

Corriente: es la velocidad a la que un flujo de electrones pasa por un punto de un circuito eléctrico completo. En otras palabras, corriente=flujo.[12]

Voltaje: es la magnitud que da cuenta de la diferencia en el potencial eléctrico entre dos puntos determinados. También llamado diferencia de potencial o tensión eléctrica. [13]

IX. DIAGRAMA DE FLUJOS

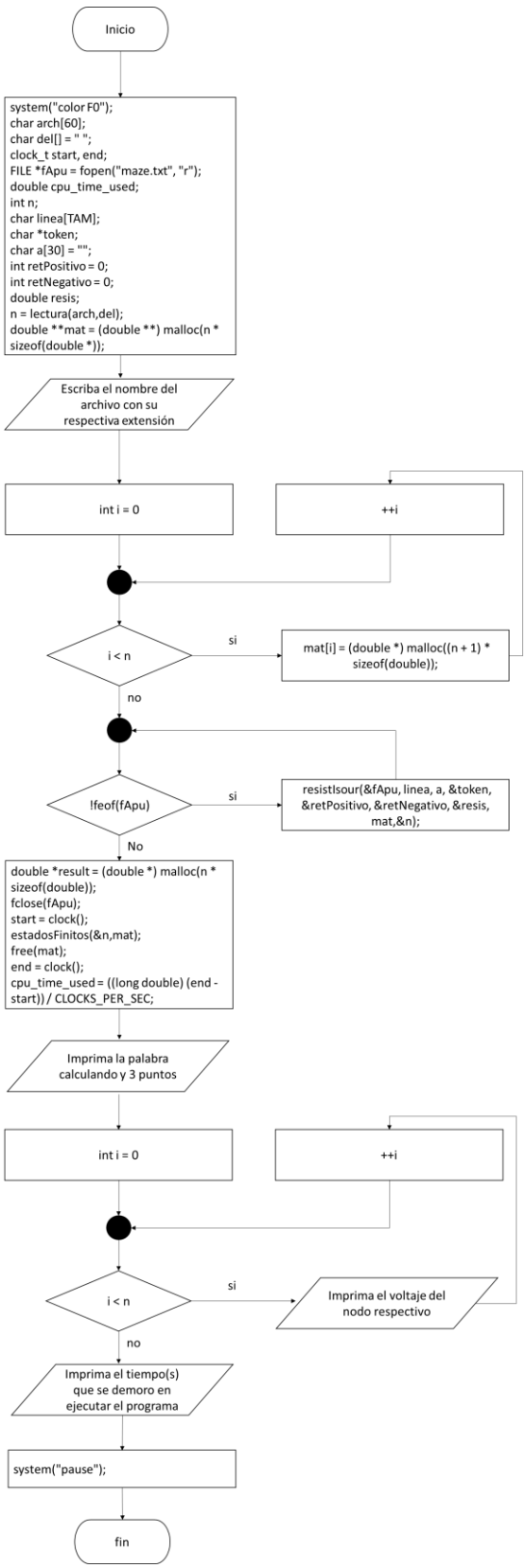


Fig. 1. Diagrama de flujos

X. DESCRIPCIÓN CÓDIGO

Inicialmente, se realizan las importaciones de librerías requeridas para el correcto funcionamiento del código

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
```

Dentro de sus funcionalidades principales se encuentran la utilización de funciones de comparación de caracteres, su asignación, la medición en el tiempo de procesamiento de cierta sección del código, hacer operaciones, estándar, de entrada-salida, gestión de memoria dinámica, entre otras.

Con ello claro, se procede a incluir los .h realizados.

```
#include "lectura.h"
#include "iSource.h"
#include "resistor.h"
#include "resistIsour.h"
#include "estadosFinitos.h"
```

A continuación se procede a explicar el funcionamiento básico de cada uno de los include mostrados previamente.

Lectura.h:

Lectura.h es una extensión del código que almacena una variable global llamada tamaño. De la misma manera, se almacena una función llamada lectura que retorna un entero con la cantidad de filas que debe tener la matriz.
Para su funcionamiento se realiza un apuntador hacia el archivo que contiene la información de los nodos. En dicho apuntador, se lee la información línea por línea y se guarda en una cadena de caracteres con el tamaño definido al inicio del código. Con la información almacenada se tokeniza con un separador de espacio " ". Con ello, se lee la segunda columna de la información que hace referencia a el nodo positivo y de esa manera se extraen los valores de los nodos. Finalmente, a medida que va leyendo las líneas va almacenando el valor del nodo mas grande que haya encontrado hasta que el archivo finalice y retorne el valor más grande encontrado.

iSource.h:

iSource.h es una extensión del código encargada de almacenar una función para desarrollar la lógica en el caso que se encuentre que la línea de lectura es una fuente de corriente('I'). Para el caso, se determinan dos condicionales. El primero indaga si el nodo positivo de la fuente es diferente de 0 y realiza la asignación de la sumatoria del valor de corriente en cuestión en la posición requerida. El segundo condicional indaga si el negativo de la fuente es diferente de 0, para poder restar su valor en la posición respectiva. Todos estos cambios, se llevan al cabo en la matriz "mat" por medio de un apuntador que ingresa por parámetro. En resumen, esta función modifica la ultima fila de la matriz en donde se encuentran los valores de corriente.

resistor.h:

resistor.h es una extensión del código encargada de almacenar una función para desarrollar la lógica en el caso que se encuentre que la línea de lectura es una resistencia('R'). Para el caso, se determinan tres condicionales. El primero indaga si el nodo positivo de la resistencia es diferente de 0 y realiza la asignación de la sumatoria de 1 sobre el valor de resistencia en la posición requerida. El segundo condicional indaga si el negativo de la fuente es diferente de 0, para poder sumar uno sobre su valor en la posición respectiva. El ultimo condicional revisa y entra si únicamente los 2 puntos de referencia de la resistencia son negativos. Para este caso los valores se restan respectivamente. En resumen, esta función llena la totalidad de la matriz dejando la diagonal positiva, y el resto de los valores con el inverso negativo o 0 respectivamente (en el nodo que se encuentre).

resistIsour.h:

resistor.h es una extensión del código encargada de almacenar una función para desarrollar la lógica que permite comparar línea por línea el archivo y determinar que función utilizar. Para ello se utilizar un comparador al primer carácter del primer token en cada línea. Para el caso que sea 'I', se llama la función "iSource" declarada en 'resistor.h'(cabe aclarar que le ingresan por parametro valores necesarios para su funcionamiento). Si el primer carácter del token llega a ser 'R', se llama la función "resistor" declarada en 'resistor.h'(de igual manera, se le ingresan parámetros para su funcionamiento).

estadosFinitos.h:

estadosFinitos.h es una extensión del código encargada de operar la matriz generada previamente para solucionar y encontrar los valores de voltaje de los nodos. Para ello utiliza máquinas de estados finitos mediante un método de gauss y back-sustitución.
En primera instancia se opera las filas de la matriz para modificar la matriz formando una escalera de ceros, esto permitirá mediante el back sustitución encontrar los valores en base a los calculados previamente. Esto se realizo en un total de 7 estados que equivalen en código estándar a 8 for y un if para poder realizar las mismas modificaciones a la matriz.

Con todos los .h definidos se procede a realizar la explicación del main respectivamente.

Las primeras líneas de código se encargan de crear variables para el correcto funcionamiento del código. Dentro de los resaltables se encuentran: el apuntador que almacena el archivo de entrada, variable que almacena el tiempo de ejecución, variable del tamaño de las filas de la matriz, el apuntador del token, el vector char que almacena línea por línea, etc.
Con el valor del tamaño almacenado en n, se crea la matriz de manera dinámica a través de la función malloc.
Con ello, se llega a un while que se realiza hasta que llegue al final del archivo. Dentro del while se encuentra la función "resistIsour" con todos los parámetros que le entran para su funcionamiento. Luego de ello se cierra el archivo y se inicia la

toma del tiempo para determinar cuándo dura realizarle gauss y back-substitution a la matriz. Para ello, se llama a la función de los estados finitos y se termina la toma de tiempo. Finalmente, se libera memoria de la matriz dinámica y se imprime en consola el tiempo que tardó.

XI. RESULTADOS

Para este proyecto se decide realizar varias pruebas del código con diferentes componentes (hardware) y encontrar una relación entre el tiempo de ejecución vs el hardware utilizado.
El primer factor para analizar es la memoria RAM en la ejecución del código.
Para ello, se comienza realizando pruebas para determinar en qué medida el ejecutable requiere de memoria RAM. Con el fin de resolver dicha duda, se hace uso del administrador de tareas y se monitorea la cantidad de RAM utilizada una vez se ejecuta el código. En un inicio, se tienen en uso 6.5gb de RAM con únicamente los procesos básicos de Windows. Pero, una vez se ejecuta el código “electricalMaze.c”, se observa un aumento a 6.6 Gb. Por lo cual, es uso de memoria para el programa es muy pequeño y la cantidad de memoria de los computadores donde se realizan las pruebas no influye en los resultados.
Sabiendo esto, se procede a analizar la frecuencia de la memoria. Para este estudio, se toma un mismo sistema y se le realizan las pruebas con una frecuencia de 2138 MHZ. En donde se obtiene la tabla 1.

2138 MHz	
Lento	Rapido
12,665	8,01
12,84	8,015
12,632	8,1
12,68	8,01
12,70425	8,03375

Tab. 1. Resultados con RAM a 2138MHz.

En dicha tabla, se observan los diferentes modos de realizar la ejecución del código. El primero de estos, determinado por la palabra “lento”, hace referencia a: gcc electricalMaze.c -o ejec.exe. Mientras que el modo” Rápido”, hace referencia a: gcc -Ofast electricalMaze.c -o ejec.exe.

Luego, se le realiza ”Overclock” a la misma RAM para darle una velocidad estable de 3600 MHz. De esta manera se sacan los siguientes valores:

3600 MHz	
lento	rapido
13,25	7,93
13,25	7,942
13,27	7,93
13,25	7,9
13,255	7,9255

Tab. 2. Resultados con RAM a 3600MHz.

Con ello, es posible hallar la diferencia porcentual de la mejora en los tiempos de compilación.

% Diferencia	
Modo Estandar	-4,33%
Modo Rapido	1,30%

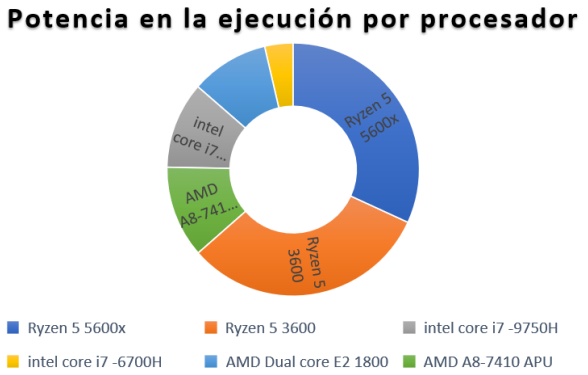
Tab. 3. Diferencia porcentual de los tiempos de ejecución con diferentes valores en la frecuencia de la RAM.

Con ello, se procede a realizar el análisis del siguiente componente que se pensó podría influir en el tiempo de ejecución. El procesador del computador.
Para realizar pruebas de ello, se decidió realizar una metodología en la cual se ejecutará el código en diferentes procesadores con diferente cantidad de núcleos. Para ello, se realiza un total de 10 pruebas por computador y se tabula el promedio en la tabla 4. Adicionalmente, se decidió realizar 2 pruebas de ejecución por computador. La primera con el comando "gcc electricalMaze.c -o ejec.exe", y la segunda "gcc -Ofast electricalMaze.c -o ejec.exe". Con ello, se halla la diferencia porcentual en los modos rápido y estándar para el código con estados finitos. De esta manera, se obtiene que:

	Modo Estandar		Modo Rapido		
	Sin E.F.(s)	Con E.F. (s)	Sin E.F.(s)	Con E.F.(s)	Dif. Modo(%)
Ryzen 5 5600x	9,5875	12,7	3,716	8,03	37%
Ryzen 5 3600	14,502	18,3398	4,998	10,216	44%
intel core i7 -9750H	16,449	410,91	6,287	328,854	20%
intel core i7 -6700H	17,8	490,8	6,6	443,72	10%
AMD Dual core E2 1800	84,21	401,092	29,84	345,6962	14%
AMD A8-7410 APU	58,94	358,83	21,35	318,52	11%

Tab. 4. Resultados de Tiempo por computador.

Para una mejor comprensión de la diferencia entre tiempo de ejecución en los procesadores se realiza la grafica 2. En donde, se observa la potencia en el modo rápido con máquina de estados finitos.



Graf.1. Diferencias entre las potencias de los procesadores.

Finalmente, se realiza la indagación de la cantidad de núcleos por procesador, obteniendo que:

- Ryzen 5 5600x = 6 núcleos
- Ryzen 5 3600 = 6 núcleos
- Intel Core i7 -9750H = 6 núcleos
- Intel Core i7 -6700H = 4 núcleos
- AMD Dual Core E2 1800 = 2 núcleos
- AMD A8-7410 APU= 4 núcleos

XII. ANALISIS

Para experimentar los efectos que generan los componentes, la velocidad de estos, y el tiempo de procesamiento. Se realiza una mejora en las frecuencias de la memoria RAM. Esto, con el objetivo de determinar en que influye la velocidad de la memoria a la ejecución del código.
Con lo anteriormente mencionado, se realiza la prueba y se toma una tabla con los resultados encontrados. Allí, se puede observar que los valores de las 2 velocidades tienen un cambio insignificante. Sin embargo, el comando "Rápido" aumento en su porcentaje mientras que el comando "Estandar" disminuyó la eficiencia. Esto, resalta como el cambiar la velocidad de la memoria en un sistema aislado, no genera un cambio muy significativo en las velocidades de procesamiento.

En base en la investigación anterior al procedimiento, se determina que acelerar el proceso de ejecución genera un retardo en el proceso de compilación. Esto, conlleva a un aumento en los requerimientos de la cantidad de memoria. Lo cual, se ve reflejado al momento de ejecutar el código. En donde, se genera una diferencia menor entre el porcentaje de diferencia en el modo estándar vs el rápido. Este resultado, demuestra que el método rápido requiere una mayor y mejor cantidad de memoria. Por lo cual, es posible afirmar que, en este modo, de no tener la suficiente cantidad de memoria podría no tener un cambio tan significativo en los resultados.

Por otra parte, se analiza los resultados con respecto al procesador. En donde a medida que el procesador aumenta su potencia, menor es el tiempo de ejecución. De allí, es posible resaltar, como la diferencia entre el comando "Ofast" y la ejecución predeterminada aumenta significativamente en los procesadores más potentes. Por lo general, se esperaría que la mejora tendiera a ser constante. Sin embargo, y en base a las investigaciones, se estima que esto se debe a que la ejecución del código no realiza un estrés del 100% en los núcleos del procesador. Por lo cual, se comprende que la consola de comandos tiene una cantidad de procesamiento reservada y limitada para la ejecución de los códigos.

Cuando se ejecuta el código "Ofast", se requiere de una mayor cantidad de procesamiento en un menor tiempo. Lo cual, en términos generales beneficia a los procesadores con una mayor eficiencia en el desarrollo de operaciones. Mientras, que los procesadores de generaciones pasadas deben realizar mayor trabajo respecto a los anteriores. Ello, genera que acelerar el proceso de ejecución genere un cambio más significativo en el tiempo a medida que el procesador es más potente.

También, se puede observar que realizar el código mediante máquinas de estados finitos ralentiza gran parte del proceso. Esto, se puede explicar debido a que las máquinas de estados finitos son procesos que buscan realizar múltiples tareas en un mismo ciclo. Por lo cual, cada una de las acciones de procesamiento tome más tiempo en ser realizada. La lógica, detrás de las maquinas de estados finitos difiere a la realización de un for. El cual, prioriza la tarea a realizar con respecto a las demás. Generando, que la velocidad de ejecución sea mucho menor a la que se tendría con máquinas de estados finitos.

XIII. CONCLUSIÓN

A. En base a lo investigado y observado en los tiempos de ejecución, se puede concluir que el comando -Ofast a cambio de minimizar tiempos de ejecución aumenta tiempos de compilación y memoria que se debe utilizar. Ello, permite que códigos que requieren de una única ejecución presenten ventajas con respecto al modo de compilación estándar. En cambio, sí el código se ejecuta en un microcontrolador que tiene poca capacidad de memoria, es preferible aumentar tiempos de ejecución a favor de minimizar el uso de memoria.

B. La cantidad de núcleos que tiene el procesador no es un factor que influya en gran medida los tiempos de ejecución. Sin embargo, factores como las arquitecturas de los procesadores (Zen 2 a Zen 3 para procesadores AMD), las frecuencias de cada procesador (MHz), los hilos por núcleo, y el tamaño de sus memorias caché son determinantes al momento de la ejecución del código y son los responsables en su mayoría del tiempo gastado en el procesamiento.

C. Las maquinas de estados finitos son una estructura que permiten la división de trabajo de procesamiento. Esto quiere decir que, a través de este tipo de códigos, es posible disminuir la carga de procesamiento instantánea, realizando varias acciones a la par y permitiendo que pequeños procesadores puedan resolver procesos complejos con mejores temperaturas a cambio de un mayor tiempo en la ejecución.

XIV. CÓDIGO DEL PROYECTO

Este proyecto es de código abierto y cualquier persona puede acceder a él. A continuación, se adjunta el link que contiene la totalidad de códigos.

https://livejaverianaedu-my.sharepoint.com/:u:/g/personal/edwardduarte_javeriana_edu_co/Ec2hCCiRKqdBg2ffQHccCIABdXEijG--yoFM7oM64RJvXA?e=Xmmy82

XV. REFERENCIAS

[1] I. "Using gauss - Jordan elimination method with CUDA for linear circuit equation systems", PROCEDIA TECHNOLOGY, 2020. [Online]. Available: https://reader.elsevier.com/reader/sd/pii/S2212017312000096?token=64B83D08156ABD5CBC9BEE398BDEE8F3F5E5AB454A17AC01224B669C11A4A294083255F8D6075BC2AC858C089CD691ED&originRegion=us-east-1&originCreation=20210516181856 [Accessed: 12- May- 2021].

[2] "análisis NODAL" SCHLUMBERGER, oilfield Glossary, 2020 [Online]. Available: https://www.glossary.oilfield.slb.com/es/terms/n/nodal_analysis. [Accessed: 12- Mayo- 2021].

[3] I. "Análisis de Nodos y Mallas", profe.uniandes, 2020. [Online]. Available: http://wwwprof.uniandes.edu.co/~ant-sala/cursos/FDC/Contenidos/03_Analisis_por_Nodos_y_Mallas.pdf. [Accessed: 12- May- 2021].

[4] "Que es C? Programacion y sintaxis", OpenWebinars, 2021. [Online]. Available: https://openwebinars.net/blog/que-es-c/ [Accessed: 12- May- 2021].

[5] "LENGUAJE C"c", *informatica*, 2021. [Online]. Available: https://informatica.uv.es/estguia/ATD/apuntes/laboratorio/Lenguaje-C.pdf. [Accessed: 12- May- 2021].

[6] "INTRODUCCION A LAS MAQUINAS DE ESTADOS FINITOS", *Tecbolivia*, 2021. [Online]. Available: http://tecbolivia.com/index.php/articulos-y-tutoriales-microcontroladores/13-introduccion-a-las-maquinas-de-estado-finito. [Accessed: 12- May- 2021].

[7] "Que significa el método de gauss en matemáticas?", superprof, 2021 [Online]. Available: https://www.superprof.es/diccionario/matematicas/algebra/metodo-gauss.html [Accessed: 12- May- 2021].

[8] "Metodos de eliminacion", *Aprende en linea*, 2021. [Online]. Available: http://aprendeenlinea.udea.edu.co/lms/moodle/mod/page/view.php?id=24487#:~:text=Sustituci%C3%B3n%20hacia%20atr%C3%A1s%20(back%20%2Dsubstitution,una%20las%20variables%20as%C3%AD%20despejadas. [Accessed: 12- May- 2021].

[9] "NETLIST", *Microensamble*, 2021. [Online]. Available: http://microensamble.com/glosario/netlist/ [Accessed: 12- May- 2021].

[10] "Memoria Dinamica", *sladeshare*, 2021. [Online]. Available: https://es.slideshare.net/gusolis93/memoria-dinamica-15706001 [Accessed: 12- May- 2021].

[11] "¿Que es la Resistencia?", *fluke*, 2021. [Online]. Available: https://www.fluke.com/es-co/informacion/blog/electrica/que-es-la-resistencia [Accessed: 14- May- 2021].

[12] "¿Que es la Corriente?", *fluke*, 2021. [Online]. Available: https://www.fluke.com/es-co/informacion/blog/electrica/que-es-la-corriente [Accessed: 14- May- 2021].

[13] "Concepto de Voltaje", *concepto.de*, 2021. [Online]. Available: https://concepto.de/voltaje/ [Accessed: 14- May- 2021].

XVI. ANEXOS

XVII. DIAGRAMA DE ESTADOS FINITOS

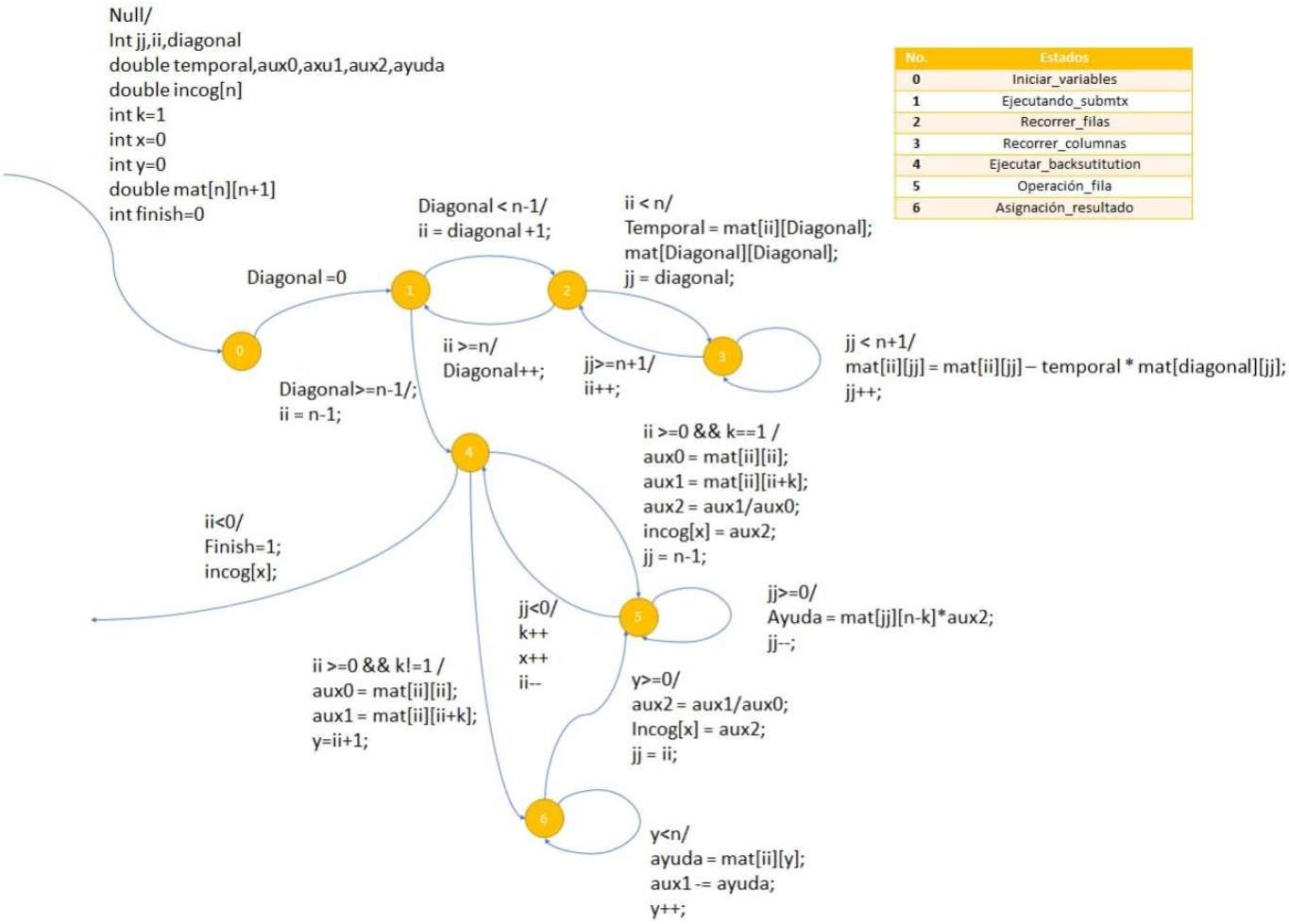


Fig. 3. Diagrama de estados finitos.