UNIVERSIDAD MICHOACANA DE SAN NICOLAS DE HIDALGO
FACULTAD DE INGENIERÍA ELÉCTRICA
DIVISIÓN DE ESTUDIOS DE POSGRADO

TÉCNICAS NUMÉRICAS Y COMPUTACIONALES APLICADAS A LA SOLUCIÓN EN ESTADO ESTACIONARIO
PERIÓDICO DE REDES ELÉCTRICAS

TESIS

Que para obtener el Grado de
DOCTOR EN CIENCIAS EN Ingeniería ELÉCTRICA
OPCIÓN SISTEMAS ELÉCTRICOS

Presenta

Ernesto Magaña Lemus

J. Aurelio Medina Ríos
Director de Tesis

Antonio Ramos Paz
Co-Director de Tesis

MORELIA, MICHOACÁN FEBRERO 2016

TÉCNICAS NUMÉRICAS Y COMPUTACIONALES APLICADAS A LA SOLUCIÓN EN ESTADO ESTACIONARIO PERIÓDICO DE REDES ELÉCTRICAS

TESIS

Que para obtener el Grado de

DOCTOR EN CIENCIAS EN INGENIERÍA ELÉCTRICA

OPCIÓN SISTEMAS ELÉCTRICOS

presenta

Ernesto Magaña Lemus

J. Aurelio Medina Rios

Director de Tesis

Antonio Ramos Paz

Co-Director de Tesis

Universidad Michoacana de San Nicolás de Hidalgo

FEBRERO 2016

Lista de Publicaciones

Congresos Internacionales ISI Thomson

1. Ernesto Magaña, Aurelio Medina y Antonio Ramos. *"Periodic Steady State Solution of Power Systems by Selective Transition Matrix Identification, LU Decomposition and Graphic Processing Units"*, Power & Energy Society General Meeting, 2015 IEEE, Págs. 1-5, Denver, Colorado, Estados Unidos de América, 2015.

2. Ernesto Magaña, Aurelio Medina y Antonio Ramos *"Periodic Steady State Solution of Power Systems by Selective Transition Matrix Identification and Graphic Processing Units"* Power, Electronics and Computing (ROPEC), 2014 IEEE International Autumn Meeting on, Págs. 1-6,Ixtapa, Guerrero, México, 2014.

3. Ernesto Magaña, Aurelio Medina, Antonio Ramos y Víctor Hugo Montesinos *"Periodic Steady State Determination of Power Systems Using Graphics Processing Units"* Electrical Engineering, Computing Science and Automatic Control (CCE), 2013 10th International Conference on, Págs. 274-279, Distrito Federal, México, 2013.

Otros Congresos Internacionales

1. Ernesto Magaña, Aurelio Medina y Antonio Ramos *"Periodic Steady State Solution of Power Systems by Selective Transition Matrix Identification"* Vigesimoséptima Reunión Internacional de Verano de Potencia, Aplicaciones Industriales y Exposición Industrial RVP-AI/2014. Acapulco, Guerrero, México. Julio de 2014.

2. Víctor H. Montesinos, Ernesto Magaña, Antonio Ramos y Aurelio Medina *"Application of Graphics Processing Units in the Fast Steady State Solution of Electric Power Systems"*, 4º Congreso Internacional de Supercómputo en México (ISUM 2013), Manzanillo, Colima, México. 2013.

3. Víctor H. Montesinos, Ernesto Magaña, Antonio Ramos y Aurelio Medina *"Rápida Solución en Estado Estacionario Periódico de Sistemas Eléctricos No- Lineales Usando Unidades de Procesamiento Gráfico"*, Reunión de Otoño de Potencia, Electrónica y Computación (ROPEC 2012), Colima, Colima, México, Noviembre 2012.

Resumen

En esta tesis se introduce el método de DN selectivo con el propósito de reducir el esfuerzo computacional del método de DN. Se muestra que siempre hay una reducción de cálculos al utilizar este método. También se muestra que se puede aplicar procesamiento paralelo al método de integración de Runge Kutta de cuarto orden, teniendo la posibilidad de usar hasta n^2 elementos de proceso (donde n es el número de variables de estado) para el cálculo de la matriz de identificación en el método DN.

En el método DN se mostró que la inversa de la matriz es un elemento importante que requiere de gran esfuerzo computacional cuando se tienen gran cantidad de variables de estado. Se logra una reducción en el tiempo de solución sin llegar a reducir la complejidad computacional usando procesamiento paralelo.

En esta tesis se presenta una propuesta del análisis de complejidad computacional al método Newton de Diferenciación Numérica (DN) para obtener el estado estacionario periódico de un sistema eléctrico. El método de DN requiere de gran esfuerzo computacional que no había sido cuantificado por una ecuación. Con la ecuación de complejidad obtenida del método de DN se propusieron algunas alternativas de solución para reducir el esfuerzo computacional y para distribuir el esfuerzo computacional entre varios elementos de proceso y con ello obtener la solución en un menor tiempo. En cada una de las propuestas de solución se realiza un análisis de complejidad computacional.

Palabras clave: Diferenciación Numérica Selectivo, Estado Estacionario Periódico, Procesamiento en Paralelo, Unidades de Procesamiento Gráfico, Matriz de Identificación.

Abstract

This thesis the ND method selective is introduced in order to reduce the computational effort of the DN method. It shows that there is always a reduction of calculations when you use this method. Also, shows that parallel processing can be applied using the integration of fourth order Runge Kutta method, having the possibility of using up to n^2 elements of process (where n is the number of state variables) for the calculation of the identification matrix in the ND method.

The DN method demonstrated that the matrix inverse is an important element that requires large computational effort when you have large number of state variables. A reduction is achieved in the time of solution without reducing the computational complexity using parallel processing.

In this thesis a proposal of analysis computational complexity to the Newton of numerical differentiation (DN) method for periodic steady state of an electrical system, is presented. The DN method requires large computational effort that had not been quantified by an equation. With the equation obtained from the ND method complexity have been proposed some alternative solutions to reduce the computational effort and to distribute the computational effort between various elements of process and thereby obtain the solution in less time. In each of the proposals solving an analysis of computational complexity is performed.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Símbolos

$\mathbf{\Phi}$	Matriz de transición de estados o matriz de identificación
$\mathbf{I}$	Matriz unitaria
$\mathbf{J}$	Matriz Jacobiana
$\mathbf{x}_\infty$	Variables de estado
$\mathbf{x}_\infty$	Variables de estado en el ciclo límite
GPU	Unidad de procesamiento gráfico
CPU	Unidad central de proceso
EDO	Ecuación diferencial ordinaria
UPFC	controladores unificados de flujos de potencia
MPI	Interfaz de paso de mensajes
PVM	Máquina virtual paralela
SVS	Compensadores estáticos de potencia reactiva
IEEE	Instituto de ingenieros en eléctrica y electrónica
ρ	Plano de Poincaré
DN	Diferenciación numérica
DNS	Diferenciación Numérica Selectiva
DGJ	Diferenciación Numérica con matriz de identificación constante
RK4	Runge Kutta de cuarto orden
C	Lenguaje de programación C
CUDA	Plataforma de computación y y programación paralela
SM	Multiprocesador de flujo en la arquitectura Fermi
SMX	Multiprocesador de flujo en la arquitectura Kepler
SMM	Multiprocesador de flujo en la arquitectura Maxwell
ALU	Unidad aritmética de números enteros
FPU	Unidad aritmética de punto flotante
T	Periodo
$\int$	Integral
Δ	Variación
D	Derivada

Capítulo 1

Introducción

En este capítulo se describe de manera concisa el estado del arte alcanzado de los métodos en los dominios de la frecuencia y el tiempo para la determinación del estado estacionario periódico de sistemas eléctricos y el estado del arte del procesamiento paralelo basado en Unidades de Procesamiento Gráfico (GPUs). Además del estado del arte se presentan el planteamiento del problema, los objetivos de la tesis, la metodología a seguir, las aportaciones del trabajo y la descripción de los capítulos.

1.1. Estado del Arte

En el estado del arte y en contenido de la tesis, se emplean términos eficiencia relativa, ganancia relativa y complejidad computacional. Estos términos se definen a continuación:

Eficiencia relativa La eficiencia relativa es una relación métrica que puede algunas veces proveer de una cantidad más conveniente para medir el desempeño de un algoritmo paralelo y se puede obtener como [Foster,1994]:

$$E_{relativa} = \frac{t_s}{t_p} \tag{1.1}$$

Donde

- t_s es el tiempo de ejecución del programa con un solo elemento de proceso
- t_p es el tiempo de ejecución del programa con p elementos de proceso.

Ganancia relativa o simplemente ganancia, se utilizará para comparar el tiempo de ejecución de dos programas que obtienen la misma solución o una solución aproximada y se puede obtener como:

$$g = \frac{programa1}{programa2} \tag{1.2}$$

Si $g > 1$, el $programa2$ se ejecuta en menor tiempo que el $programa1$ y es g veces mejor que el $programa1$; si $g = 1$, ambos programas se ejecutan en el mismo tiempo y si $g < 1$, el $programa2$ se ejecuta en mayor tiempo.

Complejidad computacional es el campo de la teoría de la computación que estudia teóricamente la complejidad para la solución de un problema. Los elementos que normalmente se estudian son:

El tiempo es una aproximación al número y patrón de pasos de ejecución de un algoritmo para solucionar algún problema.

El espacio es una aproximación de la cantidad de memoria que requiere para solucionar algún problema.

1.1.1. Técnicas de Solución del Estado Estacionario Periódico de un Sistema Eléctrico

El análisis y planeación de un sistema eléctrico requiere de modelos matemáticos de sus elementos, algoritmos y recursos computacionales importantes. La naturaleza de los sistemas eléctricos no permiten muchas veces su experimentación práctica y se tiene que recurrir a su análisis mediante simulación digital que debe estar basada en modelos adecuados de sus componentes y metodologías precisas y eficientes. Considerando el último punto es que se ha identificado al uso de sistemas de computo con suficientes elementos de proceso como una opción atractiva de análisis del comportamiento dinámico de la red eléctrica bajo diferentes escenarios de operación. Por ejemplo, variación de los parámetros, condiciones de falla, incorporación de nuevos dispositivos, entre otros.

Estas simulaciones permiten analizar las redes eléctricas que tienen un funcionamiento continúo a fin de tomar las mejores decisiones para el diseño y la operación del sistema eléctrico.

Debido al crecimiento y la complejidad de los sistemas eléctricos y del uso de nuevas tecnologías, las simulaciones de los sistemas eléctricos han incrementado su complejidad. Para desarrollar las simulaciones de forma eficiente, se han desarrollado herramientas numéricas y técnicas computacionales que ayuden a entender, planear y controlar el funcionamiento del sistema eléctrico.

Para analizar una red eléctrica se requiere modelarla de acuerdo al estudio que pretende de ella. El modelo esta basado en un conjunto de ecuaciones diferenciales ordinarias (EDOs) en el dominio del tiempo o linealizadas alrededor de un punto de operación, en el dominio de la frecuencia. Una vez que se tiene el conjunto de ecuaciones que describen la dinámica del sistema eléctrico se puede utilizar alguno de los siguientes marcos de referencia para analizar su comportamiento dinámico; es decir: dominio de la frecuencia, dominio del tiempo y dominio híbrido frecuencia-tiempo, respectivamente [Medina et. al 2013].

Métodos en el dominio de la frecuencia

Los métodos disponibles en el marco de referencia de la frecuencia se dividen en método directo, análisis armónico iterativo y método de flujo de potencia armónica. En el método directo la solución se basa en solución del sistema de ecuaciones $\mathbf{Yv} = \mathbf{i}$, donde $\mathbf{Y}$ es la matriz de admitancias del sistema, $\mathbf{v}$ es el vector de voltajes de los nodos e $\mathbf{i}$ es el vector de inyección de corrientes.

Método directo en este método se tiene el método de inyección de corrientes que utiliza el marco de componentes de secuencia inyectando una fuente de corriente ideal al sistema eléctrico [Mahmoud, 1982]. En otra contribución, la solución se obtiene en el dominio de fase para un sistema trifásico desbalanceado [Demsem, 1984].

Análisis armónico iterativo el analisis armónico iterativo está basado en el proceso iterativo siguiente:

1. El dispositivo que produce los armónicos es modelado como un suministro de voltaje dependiente de una fuente de corrriente, representada en cada iteración fuente de corriente de armónicos fija.
2. El problema es resolver primeramente una estimación del suministro de voltaje para obtener las corrientes armónicas. Las corrientes armónicas se usan para obtener los voltajes armónicos.

3. Los voltajes armónicos permiten el cálculo de corrientes armónicas más precisas. El proceso iterativo se termina hasta que las corrientes armónicas son suficintemernte pequeñas [Dommel, 1986], [Callaghan, 1990] y [Callaghan, 1989].

El Método de flujo de potencia armónica toma en cuenta la naturaleza del voltaje dependiente de los componentes del sistema. Las ecuaciones de voltaje y corriente armónica se resuelven de forma simultánea usando un algoritmo tipo Newton [Sharma, 1991], [Acha, 1988], [Acha, 1989], [Xia, 1982], [Xu, 1991], [Arrillaga, 1995], [Arrillaga, 2004] y [Bathurst, 2000].

Métodos en el dominio del tiempo

En el marco de referencia del tiempo; en principio, la solución del estado estacionario periódico se puede obtener en el dominio del tiempo mediante la integración de las ecuaciones diferenciales que describen el sistema una vez que ha pasado la respuesta transitoria. Los métodos que se han utilizado en el marco de referencia del tiempo es el método convencional de fuerza bruta que basa su solución en un método de integración numérica que puede Runge Kutta o la regla trapezoidal y un método para la rápida solución en estado estacionario periódico que basa su solución en determinar el valor de las variables de estado en el ciclo límite. La ventaja de trabajar el el dominio del tiempo es que se tienen todo es espectro armónico.

En [Medina et. al 2013] se detallan las contribuciones que se han realizado en el marco del dominio del tiempo.

Métodos en el marco de referencia híbrido

En el marco de referencia híbrido los componentes del sistema son representados en su marco de referencia natural, es decir, en el marco de referencia de la frecuencia los componentes lineales y en marco de referencia del tiempo los no lineales. Los voltajes de los nodos de carga donde están conectados los componentes no lineales se pueden obtener mediante un proceso iterativo [Semlyen y Medina, 1995].

En esta tesis se utiliza el marco de referencia del tiempo para determinar el estado estacionario periódico de un sistema eléctrico.

Uno de los problemas de particular interés en el análisis de redes eléctricas con componentes no lineales y/o variantes en el tiempo es la rápida determinación de estado estacionario periódico. Para resolver este problema se han propuesto algunos métodos, que se le ha denominado técnicas de acercamiento rápido. En [Chua y Ushida, 1981] los métodos de acercamiento rápido al estado estacionario periódico son clasificados en métodos de: fuerza Bruta, Balance Armónico y Disparo.

En esta tesis se trabajó en un método de disparo propuesto en [Semlyen y Medina, 1995]. El objetivo de este método es encontrar una condición inicial del vector $x(0)$ tal que cuando se integre el sistema de EDOs ($x = f(x,t)$), sobre un periodo completo de tiempo T, a partir de la condición inicial $x(0)$ se obtenga $x(T) = x(0)$. En [Semlyen y Medina, 1995] se proponen tres métodos Newton de extrapolación al Ciclo Límite basados en el plano de Poincaré [Parker y Chua et. al 1989] que son: Diferenciación Numérica (DN), Aproximación Directa y Matriz Exponencial.

El método de DN propuesto en [Semlyen y Medina, 1995] se ha aplicado en diferentes estudios, por ejemplo:

En [Rodríguez y Medina, 2004] se aplica el método de DN para el análisis de estabilidad transitoria de una máquina síncrona.

En [García, 2005] se propone una técnica de procesamiento en paralelo basada en MPI para la solución del estado estacionario periódico de una red eléctrica.

En [García y Medina, 2003] se propone una aplicación del método de DN propuesto en [Semlyen y Medina, 1995] para obtener la solución del estado estacionario periódico en la solución de sistemas eléctricos que contienen compensadores de estáticos de potencia reactiva (SVSs).

En [Medina y Ramos, 2003] se propone el uso del método DN y una técnica de procesamiento en paralelo basado en PVM en la solución del estado estacionario periódico de un sistema eléctrico.

En [Peña y Medina,2010] se propone el método de DN para un estudio de la máquina de inducción.

En [Segundo y Medina, 2010a] se propone un proceso mejorado al convencional DN haciendo uso de la propiedad de simetría de media onda en el análisis de redes eléctricas en el dominio del

tiempo. El proceso mejorado se basa en el cálculo de medio ciclo (periodo) de integración, en vez del ciclo completo.

En [Segundo y Medina, 2010] se propone un método para la solución del estado estacionario en sistemas de potencia basado en un método de expansión exponencial. En esta propuesta reportan una eficiencia relativa de 37.5 comparado con el método de DN.

En [Segundo y Medina, 2008] se propone la rápida solución del estado estacionario periódico de sistemas eléctricos con controladores unificados de flujos de potencia (UPFC, por sus siglas en inglés) mediante el método de DN.

1.1.2. Técnicas de Procesamiento en Paralelo Basadas en Unidades de Procesamiento Gráfico

Las Unidades de Procesamiento Gráfico aparecieron en el mercado en el año 2006. Nvidia que es uno de los fabricantes más importantes de tarjetas gráficas; introdujo al mercado 5 nuevas arquitecturas de las unidades de procesamiento gráfico (GPUs) que son la G80, GT200, Fermi [Nvidia, 2009], Kepler [Nvidia, 2012] y Maxwell [Nvidia, 2015], siendo la más reciente la arquitectura Maxwell. Las arquitecturas de las GPUs que se utilizaron en esta tesis fueron la Tesla y la Kepler de la primera generación.

En el sitio de «IEEE Explore» existen 7723 trabajos científicos relacionados con las GPUs. Ello pone a las GPUs como dispositivos muy utilizados en la actualidad en las diferentes áreas del conocimiento. Difícilmente se podrán citar todos los artículos relacionados con las GPUs, por lo que a continuación se hace referencia a algunos trabajos especializados relacionados con esta tesis:

En [Jalili, 2012] se reporta el estudio de estabilidad transitoria a sistemas eléctricos de gran escala usando GPUs. Se utiliza un arreglo de 4 GPUs, lográndose eficiencias relativas de 10.44 en una parte del proceso. En este trabajo se divide el sistema eléctrico en subsistemas. Cada subsistema puede ser calculado por una GPU. Se reportan reducciones primeramente al dividir en subsistemas y luego al procesar los subsistemas con una GPU.

En [Maggioni, 2013] se hace un estudio del estado estacionario de la ecuación química maestra basando el procesamiento en GPUs.

En [Xiaoming, 2015] se hace un estudio de la simulación de circuitos eléctricos usando factorización LU y técnicas de dispercidad. Se reporta una eficiencia relativa promedio de 7.02 en la factorización LU. La eficiencia relativa se obtiene de forma experimental.

En [Neic,2012] se describen simulaciones realizadas en el área de la salud. Los resultados obtenidos generan eficiencia relativa en el cómputo de los modelos propuestos de 11.8 y 16.3, que se obtuvieron utilizando 20 GPUs. El mismo estudio requiere hasta 476 núcleos de la CPU.

En [García-Olmos, 2013] se propone una técnica basada en GPUs para calcular la matriz de identificación del método de DN propuesto en [Semlyen y Medina, 1995]; se reportan eficiencias relativas inferiores a 6 en el proceso completo del método DN.

Unidad de Procesamiento Gráfico

Una unidad de procesamiento gráfico GPU (del acrónimo "Graphics Processing Unit") es un procesador dedicado al procesamiento de gráficos. La GPU fue diseñada para ayudar al procesador central y está optimizada para el cálculo de operaciones de punto flotante; que son las operaciones que predominan en las funciones 3D.

La mayor parte de la información ofrecida en la especificación de una tarjeta gráfica se refiere a las características de la GPU, pues constituye la parte más importante de la tarjeta gráfica, así como es la principal determinante del rendimiento.

Tres de las características más importantes de la GPU son:

- La frecuencia de reloj del núcleo,
- El número de procesadores
- El número de multiprocesadores de flujo.

En la actualidad los dos fabricantes más importantes de GPUs son Nvidia y AMD (Advanced Micro Devices). Una de las características más importantes de las GPUs es el número de procesadores shaders (shaders=cuda en Nvidia y shaders=stream en AMD). En esta tesis se trabajó con las GPUs Nvidia por varias razones; algunas de ellas son las siguientes:

Nvidia desarrolló CUDA (Compute Unified Device Architecture-Arquitectura Unificada de Dispositivos de Cómputo) para la programación de GPUs, que es un entorno de programación parecido al lenguaje C.

Nvidia ha trabajado de forma muy importante en el desarrollo de nueva tecnología. Como resultado ha puesto en el mercado 5 arquitecturas de sus GPUs.

Existen mayores trabajos científicos de diferentes áreas que usan las GPUs Nvidia. En [Jalili, 2012] se reporta un estudio de estabilidad transitoria a sistemas eléctricos usando GPUs; en [Maggioni, 2013] se reporta un estudio del estado estacionario de la ecuación química maestra, en [Neic,2012] se describen simulaciones realizadas en el área de la salud.

Las mejores supercomputadoras del mundo [Top-500,2015] usan GPUs Nvidia para el procesamiento de la información.

Arquitecturas de las GPUs

Nvidia ha logrado introducir al mercado varios productor asociados a procesamiento gráfico. Una de las características que tienen las tarjetas gráficas es la cantidad importante de procesos que se pueden ejecutar de forma paralela; ese procesamiento en paralelo hace atractiva a una tarjeta gráfica para usarla en aplicaciones distintas al procesamiento gráfico.

Nvidia ha lanzado al mercado varios productos asociados a las tarjetas gráficas (estos se describen en [Nvidia-93-2016]). Los productos que interesan de Nvidia para el desarrollo de esta tesis son las GPUs y las herramientas para la programación de las aplicaciones en ellas. Algunos de los productos relacionados con las GPUs que Nvidia introdujo al mercado son los siguientes:

- En el año 1999 Nvidia inventa la primer GPU [Nvidia].
- En el año 2006 Nvidia introduce la plataforma de CUDA [CUDA,2015].
- En el año 2009 Nvidia presenta la arquitectura Fermi [Fermi,2009].
- En el año 2012 Nvidia introduce las primeras GPUs con arquitectura Kepler [Nvidia, 2012].
- En el año 2014 Nvidia introduce las primeras GPUs con arquitectura Maxwell [Nvidia, 2015].

En el año 2006, Nvidia introdujo la plataforma de computación CUDA. CUDA es una plataforma de propósito general para computo paralelo. El modelo de programación de CUDA aprovecha el motor de cálculo paralelo en la GPU Nvidia para resolver problemas computacionales complejos de manera más eficientes que una CPU.

CUDA contiene un entorno de software que le permite a los desarrolladores a usar el lenguaje C como un lenguaje de alto nivel. Otros lenguajes, interfaces de programación de aplicaciones o directivas también son soportadas por CUDA, tales como Fortran, DirectCompute y OpenACC.

Las arquitecturas que Nvidia introdujo al mercado en la última década son: la arquitectura G80, la GT200, la Fermi, la Kepler y la Maxwell. La arquitectura G80 fue la primera que soportó el lenguaje C; esta es una ventaja importante para los programadores que ya saben ese lenguaje. Las arquitecturas G80 y GT200 tienen un multiprocesador de flujo (Streaming Multiprocessor SM) de 8 núcleos. Las GPUs que se usaron en esta tesis usan la arquitectura Fermi y Kepler de la primera generación. Los aspectos importantes de la arquitectura Fermi se analizan en la siguiente sección.

Arquitectura Fermi

La arquitectura Fermi [Nvidia, 2009]) fue introducida por la compañía Nvidia en el año 2009. La primera GPU basada en la arquitectura Fermi constaba de 3.0 millones de transistores y contó con 512 núcleos CUDA. Un núcleo CUDA ejecuta una instrucción de enteros y una instrucción de punto flotante por ciclo de reloj por hilo. Los 512 núcleos CUDA están organizados en 16 SM de 32 hilos cada SM (ver Figura 1.1 [Nvidia, 2009]). La GPU cuenta con 6 partes de memoria de 64 bits cada una, para un total de 384 bits; que soporta hasta 6 GB de memoria GDDR5 DRAM.

En el bloque "Host Interface"de la Figura 1.1 se conecta la GPU a la CPU a través del puerto PCI- Express. El bloque "GigaThread" (de la Figura 1.1) es el planificador global y distribuye los bloques de hilos en los planificadores de hilos del SM; ver bloque "Warp Scheduler" de la Figura 1.2.

Cada SM tiene 32 núcleos CUDA [Nvidia, 2009], esto significa que tiene 4 veces más que la arquitectura G80. Cada núcleo CUDA tiene una unidad aritmética lógica de enteros (ALU) y una unidad aritmética de punto flotante (FPU) [Nvidia, 2009]. Las GPUs anteriores a la arquitectura Fermi utilizan el entandar IEEE 754-1985 [IEEE-754 Std.1985] para el manejo de la aritmética de puto flotante. La arquitectura Fermi implementa el estándar IEEE 754-2008 [IEEE-754 Std. 2008]

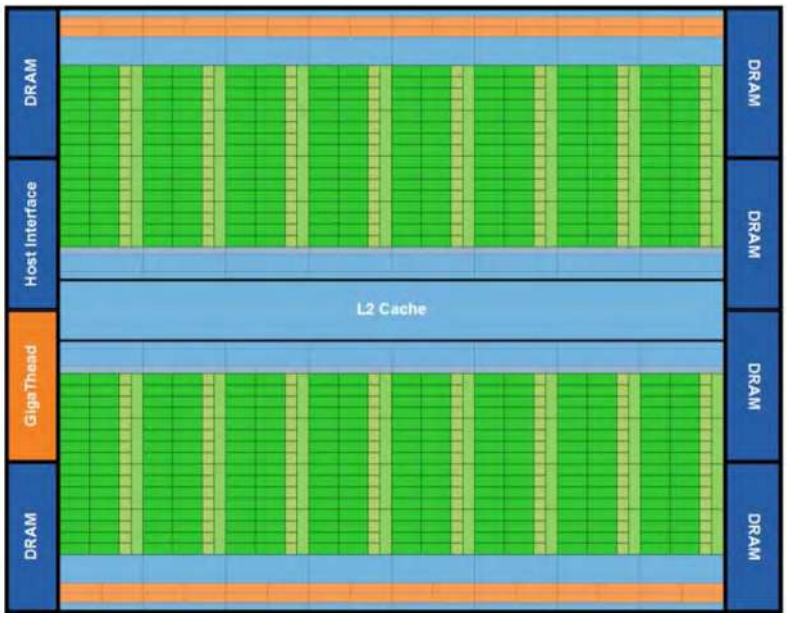


Figura 1.1: Arquitectura Fermi

de punto flotante; el estandar proporciona la instrucción fusionada multiplicación y suma para las aritméticas de precisión simple y doble [Nvidia, 2009].

En la Figura 1.2 [Nvidia, 2009] se muestra el contenido de un SM y el contenido de núcleo CUDA en la arquitectura Fermi. Las GPUs de la arquitectura Fermi tienen cierta cantidad de SM (la GPU Nvidia Tesla C2075 que se utilizó en esta tesis tiene 14 SM); cada SM tiene 32 núcleos CUDA. Los 32 núcleos del SM comparten una memoria de 64 KB y pueden acceder a la memoria global (memoria a la que pueden acceder todos los núcleos de la GPU).

Cada SM tiene 16 unidades de carga/almacenamiento (load/store (LD/ST)) [Nvidia, 2009] lo que permite identificar la dirección de origen y destino que tienen que ser calculados por los 16 hilos por ciclo de reloj. Las unidades de LD/ST soportan los datos de cada dirección de la memoria caché o de la memoria DRAM (memoria global).

La GPU tiene cuatro unidades para funciones especiales (Special Function Unit (SFU)) [Nvidia, 2009]. Estas unidades ejecutan instrucciones como el seno, coseno, reciproco y raíz cuadrada. Cada SFU ejecuta una instrucción por hilo por ciclo de reloj.

Tabla 1.1: Comparación de las arquitecturas G80, GT200 y Tesla

GPU	G80	GT200	Fermi
Transistores	681 millones	1.4 billones	3.0 billones
núcleos CUDA	128	240	512
Operaciones de precisión doble	No	30 FMA/reloj	256 FMA/reloj
Operaciones de precisión simple	128 MAD/reloj	240 MAD/reloj	512 FMA/reloj
SFUs/SM	2	2	4
Memoria compartida/SM	16 KB	16 KB	48 KB o 16KB

Arquitectura Kepler

La arquitectura Kepler fue introducida por Nvidia en el año 2012 [Nvidia, 2012]. Existen 3 versiones de las arquitecturas Kepler que son: GK104, GK110 y GK220. Las implementaciones completas de las dos últimas puen tener hasta 15 unidades SMX y seis controladores de memoria de 64 bits.

Las principales características que cambió esta arquitectura en comparación con la arquitectura Fermi son:

- La nueva arquitectura del multiprocesador de flujos para la arquitectura Kepler (SMX).
- Un subsistema de memoria mejorado.
- La arquitectura Kepler GK220 incrementa el archivo de registros y la memoria compartida por SMX

Uno de los elementos más importantes en arquitecturas de GPUs es el SM en las arquitecturas anteriores a la arquitectura Kepler. La evolución del SM ha sido muy importante, ya que en las arquitecturas G80 y GT200 cada SM tenía 8 núcleos CUDA y en la arquitectura Fermi tiene 32 núcleos CUDA. En la arquitectura Kepler el SM cambió a SMX; un SMX en la arquitectura Kepler tiene 192 núcleos CUDA, según se muestra en la Figura 1.3.

En la arquitectura Kepler de la segunda generación se introdujo la programación dinámica. La programación dinámica permite a un Kernel (función que se ejecuta en una GPU) llamar a otro

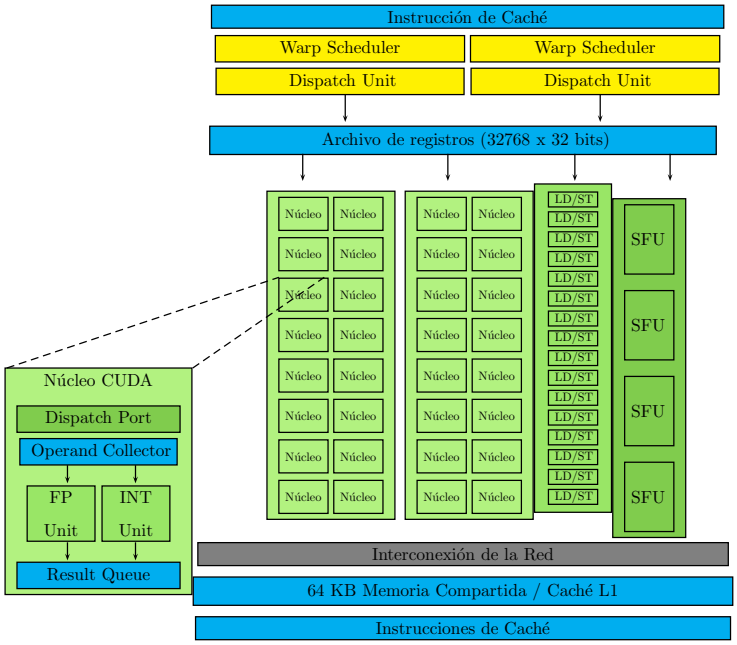


Figura 1.2: SM y esquema un núcleo cuda

kernel como lo hace una función que se ejecuta en la CPU. En esta tesis no se emplea la programación dinámica debido a que las GPUs que se tienen no soportan este tipo de programación.

Arquitectura Maxwell

Uno de los elementos mas importantes que ha evolucionado en las diferentes arquitecturas que ha presentado Nvidia en la última década ha sido el multiprocesador de Flujo SM. En la arquitectura Kepler se le llamó SMX y contiene 192 núcleos CUDA por SMX; en la arquitectura Maxwell, que se introdujo en el año 2014, el SM es denominado SMM y cada SMM está compuesto por cuatro bloques de 32 núcleos, para un total de 128 núcleos CUDA. La Figura 1.4 ilustra lo anterior.

1.1.3. Plataforma para el Procesamiento en Paralelo Basado en Unidades de Procesamiento Gráfico

En esta sección se analizan algunos elementos del modelo de programación CUDA introducido en [CUDA,2015] por la compañía Nvidia para la programación de sus GPUs.

Figura 1.3: SMX en la arquitectura Kepler

En la Figura 1.5 se presenta el modelo de programación de CUDA. Un programa puede tener procesos que se pueden calcular de forma secuencial y procesos que se ejecutan en paralelo. El proceso secuencial se puede ejecutar en una CPU debido a que la frecuencia a la que trabaja la CPU es mayor a la frecuencia a la trabajan los núcleos CUDA en una GPU. Al proceso en la CPU se le puede llamar Host y al proceso paralelo de puede ejecutar en una GPU se le puede llamar Device.

En CUDA se definen funciones que serán ejecutadas en la CPU y en la GPU. La funciones que son ejecutadas en la GPU toman el nombre del kernel. Cada kernel se ejecuta idealmente de

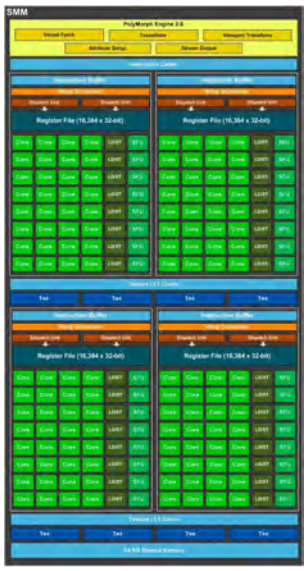


Figura 1.4: SMM en la arquitectura Maxwell

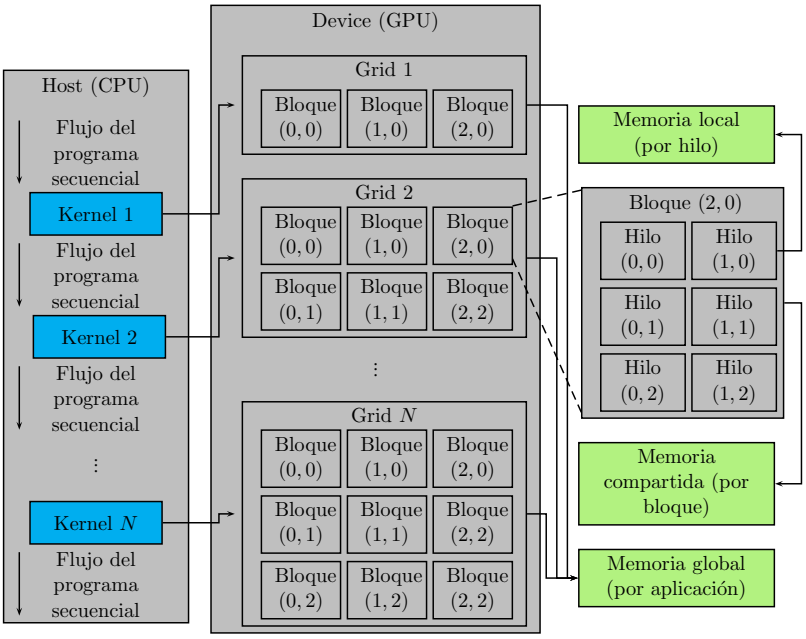


Figura 1.5: Esquema del modelo de programación en CUDA y tipos de memoria

forma simultánea por los diferentes hilos de la GPU. En la Figura 1.5 se observa que la ejecución de un programa puede iniciar con un proceso secuencial, luego sigue un proceso paralelo en la GPU al ejecutar el Kernel 1 en diferentes bloques de hilos. Una vez que termina la ejecución del kernel 1 en la GPU se procede a ejecutar un proceso secuencial en la CPU. El proceso sigue alternando una parte secuencial y una parte paralela hasta que termina la aplicación.

Los bloques de hilos en la GPU se pueden dividir en una rejilla (Grid) de bloques; que puede ser de una, dos o tres dimensiones. En la rejilla 2 de la Figura 1.5 hay 6 bloques; distribuidos en dos dimensiones (dos renglones de 3 bloques cada uno).

Los hilos en un bloque también pueden estar distribuidos en una, dos o tres dimensiones. En la Figura 1.5 se muestra que el bloque 2 ejecuta 6 hilos distribuidos en dos dimensiones (tres renglones y dos columnas).

Jerarquía de Memoria

En la GPU existen tres clases de memoria que son:

- Memoria local: este tipo de memoria solamente la puede usar un hilo. Los otros hilos no pueden hacer cambios en la memoria local al hilo.
- Memoria compartida (shared): este tipo de memoria la pueden usar todos los hilos del bloque. Los hilos de otros bloques no tienen acceso a esta memoria.
- Memoria global: este tipo de memoria la pueden usar todos los hilos de todos los bloques.

En la Figura 1.5 se observa de forma clara que el hilo puede ocupar una memoria local. En un bloque de hilos es necesario usar la memoria compartida entre los núcleos CUDA del mismo bloque y la memoria global se utiliza para compartir la información entre los diferentes hilos de los diferentes bloques.

1.2. planteamiento del problema

Cuando se requiere determinar el comportamiento de un sistema en tiempo real es necesario técnicas que reduzcan el esfuerzo computacional y plataformas de procesamiento paralelo que

ayuden a reducir el tiempo para obtener la solución. Determinar el estado estacionario periódico de un sistema es la base de otros estudios.

1.3. Justificación de la Tesis

En base a lo escrito en el estado del arte, hay muchos trabajos relacionados para mejorar y aplicar el método de DN propuesto en [Semlyen y Medina, 1995]. Ninguno de ellos hace un análisis de complejidad computacional del algoritmo de DN; todas las propuestas para mejorar el método de DN justifican sus métodos con casos de estudio y comparan el método propuesto con otros métodos ya probados.

Los trabajos que se han reportado para mejorar el método de DN propuesto en [Semlyen y Medina, 1995] usando procesamiento paralelo han concentrado su atención en el cálculo de la matriz de identificación o transición de estados y no consideraron o no tenían posibilidades de ahondar con el proceso paralelo que se puede lograr en el método de integración que se requiere para calcular cada columna de la matriz de identificación. Los estudios no indican el número de elementos de proceso que pueden usarse para procesar en paralelo el método DN.

Las GPUs son dispositivos que tienen muchos elementos de proceso que pueden trabajar de forma paralela; dichos elementos de proceso ayudan reducir los tiempos de solución de un problema que se puede dividir en tareas independientes. Las aplicaciones de la GPU se han extendido a todas las áreas donde se requieren cálculos masivos. Construir un equipo basado en GPUs es fundamental para probar las metodologías propuestas al método de DN propuesto en [Semlyen y Medina, 1995].

En [Segundo y Medina, 2010] se propone un método para integrar la mitad de un periodo en lugar de integrar un periodo completo; con esto se logra una eficiencia relativa cercana a 2 en el procesamiento del método convencional DN. Cuando se usa procesamiento en paralelo en el método de integración, la eficiencia relativa idealmente puede llegar a n, donde n es el número de variables de estado del sistema eléctrico. El procesamiento paralelo del método de integración y la integración del medio ciclo se pueden usar simultáneamente.

La inversa de una matriz, necesaria en el método DN es mencionada en [García, 2005]. En este trabajo indican que la inversa de la matriz reduce la eficiencia relativa del método de DN propuesto en [Semlyen y Medina, 1995]; la afirmación no la hacen en base a un estudio en el que

se determine el tiempo para calcular la inversa.

1.4. Objetivos de la Tesis

1.4.1. Objetivo General

Proponer y probar metodologías que ayuden a reducir el tiempo cómputo del método de DN propuesto en [Semlyen y Medina, 1995]. Las propuestas pueden ser basadas en reducciones de cálculos o procesamiento en paralelo. Las reducciones en el tiempo de cómputo se pueden obtener comparando la complejidad computacional de los métodos de DN propuestos con la complejidad del método DN tradicional o por medición del tiempo de ejecución de las propuestas del método DN con el tiempo de ejecución del método DN tradicional.

1.4.2. Objetivos Particulares

Realizar un análisis de complejidad computacional del método DN propuesto en [Semlyen y Medina, 1995] en donde se puedan verificar las oportunidades para reducir el tiempo para determinar el estado estacionario periódico de un sistema eléctrico. El estudio de complejidad computacional de un algoritmo aplica solamente a ese algoritmo.

1.5. Metodología

La metodología en que está basada esta tesis es la siguiente:

Para el estudio de complejidad del método de DN propuesto en [Semlyen y Medina, 1995], se proponen variables que dependen del tiempo debido a que en algunas expresiones no están definidas el número de operaciones que se tienen que procesar, debido a que dependen del sistema eléctrico.

Se proponen algunas alternativas de solución para reducir el tiempo para determinar el estado estacionario periódico de un sistema eléctrico. Las propuestas se basan en el método DN propuesto en [Semlyen y Medina, 1995]. Una de las propuestas reduce el número de cálculo y otras propuestas distribuyen los cálculos para ser procesados de forma paralela mediante unidades de procesamiento gráfico. En todas las propuestas se hace un análisis de complejidad computacional.

Construir un equipo basado en un procesador de 4 núcleos y tres unidades de procesamiento gráfico, para probar las metodologías propuestas.

Comparar las metodologías propuestas con el método de DN convencional para determinar las reducciones en el tiempo o el mejoramiento de la eficiencia relativa.

1.6. Aportaciones

Se propuso el método Diferenciación Numérica selectivo para determinar el estado estacionario periódico de un sistema eléctrico y su complejidad.

Se propuso una alternativa de paralelización del método de Runge Kutta de cuarto orden en el cálculo paralelo de la matriz de identificación. Se muestra que en el proceso para el cálculo de la matriz de identificación se pueden usar n^2 elementos de proceso. Cuando se tengan n^2 elementos de proceso, la complejidad computacional en el proceso de cálculo de la matriz de identificación pasa de cuadrática a lineal y finalmente a constante.

Se determinar la complejidad computacional del método de DN convencional. El resultado de la ecuación de complejidad computacional indica que la inversa de la matriz es el proceso que mayor esfuerzo computacional necesita después de una cantidad finita de variables de estado. En sistemas pequeños, la matriz de identificación es la que mayor esfuerzo computacional requiere en el método de DN.

Se propuso un procedimiento que permite reducir el tiempo de solución de la inversa de la matriz aplicando la factorización LU y un proceso paralelo en la sustitución hacia delante y hacia atrás. Se muestra que la eficiencia relativa en esta parte del proceso puede llegar a ser 4.

Se introduce al método de DN con matriz de identificación constante un proceso paralelo para calcular los periodos de integración en los ciclos iniciales, el ciclo base y de los periodos de integración que se requieren después de calcular la matriz de identificación.

1.7. Descripción de Capítulos

En el Capítulo 2 se hace una propuesta de complejidad computacional al método DN tradicional para obtener el estado estacionario periódico de un sistema eléctrico.

En el Capítulo 3 se presentan las propuestas del método de DN basadas en el método DN tradicional para determinar la solución en estado estacionario periódico de un sistema eléctrico.

En el Capítulo 4 se presentan los casos de estudio en que se muestran las ventajas de las propuestas del método de DN, con respecto al método de DN tradicional.

En el Capítulo 5 se presentan las conclusiones generales de la tesis y se proponen trabajos futuros a realizar en el mismo campo del conocimiento.

Capítulo 2

Propuesta de Complejidad al Método de Diferenciación Numérica para la Solución en Estado Estacionario Periódico de Redes Eléctricas

2.1. Introducción

El método de Diferenciación Numérica es un método Newton que se introdujo en [Semlyen y Medina, 1995]; con el propósito de obtener de manera eficiente la solución en estado estacionario de una red no lineal y/o variante en el tiempo. Se han propuesto modificaciones a su proceso de solución, con el propósito de incrementar su eficiencia y reducir el esfuerzo computacional asociado con la aplicación del método. En [Segundo y Medina, 2010] se logró reducir casi a la mitad el esfuerzo computacional asociado con el método al tomar en consideración la simetría de media onda en circuitos y redes eléctricas. En [García, 2005], [García-Olmos, 2013] y [Medina y Ramos, 2003] se logró reducir el tiempo de ejecución al aplicar tecnología de procesamiento en paralelo con diferentes plataformas computacionales. En esos trabajos la reducción de los tiempos se reportan por medio del concepto de eficiencia relativa.

En este capítulo se propone un análisis de la complejidad computacional del método

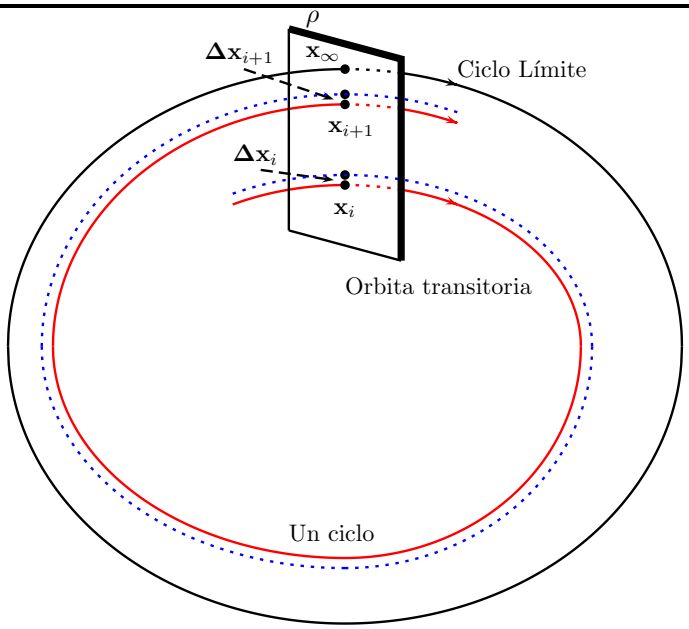


Figura 2.1: Orbita del vector de estados $\mathbf{x}$.

DN y se explica brevemente en la Sección 2.2. La ecuación que se espera obtener del análisis de complejidad servirá para enfocar el esfuerzo para reducir el tiempo para hallar la solución del estado estacionario periódico de un sistema eléctrico.

2.2. Método de Diferenciación Numérica

El comportamiento periódico de una red eléctrica puede ser determinado en el dominio del tiempo, mediante la integración del conjunto de Ecuaciones Diferenciales Ordinarias (EDO's) que describen su dinámica. Su descripción general en términos de una EDO es la siguiente:

$$x' = f(x,t) \tag{2.1}$$

Si la fuerza impulsora para el conjunto EDO's que describen la dinámica de un sistema es periódica, entonces $f(.,t)$ es un vector T-periódico. La solución en estado estable $x(t)$ es también periódica y puede ser representada como el Ciclo Límite [Parker y Chua et. al 1989] para x_k en términos de otro elemento periódico de x o en términos de una función T-periódica, por ejemplo $sen(wt)$, tal y como se muestra en la Figura 2.1. Otras variables tales como $i(t)$ se obtienen de ecuaciones algebraicas.

Antes de alcanzar el Ciclo Límite, los ciclos de una órbita transitoria están cercanos a éste. Su posición está apropiadamente descrita por su representación sobre el plano de Poincaré ρ. Un ciclo sencillo mapea su punto de inicio $\mathbf{x}_i$ a un punto final $\mathbf{x}_{i+1}$ y mapea un segmento de perturbación $\mathbf{\Delta x}_i$, tal y como se aprecia en la Figura 2.1. Todos los mapeos cerca de un Ciclo Límite son quasi-lineales, así que el método Newton puede ser usado para obtener el punto en el Ciclo Límite $\mathbf{x}_\infty$. Esto es independiente de su estabilidad [Semlyen y Medina, 1995].

Con el propósito de tomar ventaja de la linealidad en la vecindad de un Ciclo Base, se puede linealizar la Ecuación (2.1) alrededor de una solución de $x(t)$ desde t_i hasta t_{i+T}. Esto resulta en un problema variacional:

$$\mathbf{\Delta x}' = \Delta f(x(t), t) = D_x f(x, t)\mathbf{\Delta x} = \mathbf{J}(t)\mathbf{\Delta x} \tag{2.2}$$

Donde,

$\mathbf{J}(t)$ es la matriz Jacobiana (T-periódica). La condición inicial es:

$$\mathbf{\Delta x}(t_i) = \mathbf{\Delta x}_i \tag{2.3}$$

La Ecuación (2.3) es una EDO lineal, variante en el tiempo, con una solución de la forma cerrada:

$$\mathbf{\Delta x}(t) = \mathbf{\Delta x}_i e^{\int_a^b \mathbf{J}(t)dt} \tag{2.4}$$

La Ecuación (2.4) satisface la Ecuación (2.1). Para $t = t_i + T$ se tiene :

$$\mathbf{\Delta x}_{i+1} = \mathbf{\Phi}\mathbf{\Delta x}_i \tag{2.5}$$

$$\mathbf{\Phi} = e^{\int_{t_i}^{t_i+T} \mathbf{J}(t)dt} \tag{2.6}$$

Se puede ver que $\mathbf{\Phi}$ resulta casi la misma para cualquier t_i tal que el mapeo cerca del ciclo límite está cerca de la linealidad.

La Ecuación (2.4) muestra que los segmentos de entrada son mapeados para corresponder a los segmentos de salida por medio de la matriz $\mathbf{\Phi}$. Por el conjunto de relaciones en el plano de

Poincaré ilustrados por la Figura 2.1, se pretende identificar también la matriz $\mathbf{C}$, definida de la manera siguiente [Semlyen y Medina, 1995].

De la Figura 2.1 se tiene que $\mathbf{\Delta x}_i = \mathbf{x}_\infty - \mathbf{x}_i$, con lo que $\mathbf{\Delta x}_{i+1} = \mathbf{x}_\infty - \mathbf{x}_{i+1}$. Igualando $\mathbf{x}_\infty$ de las dos expresiones anteriores se tiene,

$$\mathbf{\Delta x}_i + \mathbf{x}_i = \mathbf{\Delta x}_{i+1} + \mathbf{x}_{i+1} \tag{2.7}$$

Sustituyendo la Ecuación (2.5) en la Ecuación (2.7) y despejando $\mathbf{\Delta x}_i$ se tiene,

$$\mathbf{\Delta x}_i = (\mathbf{I} - \mathbf{\Phi})^{-1} (\mathbf{x}_{i+1} - \mathbf{x}_i) \tag{2.8}$$

Sustituyendo la Ecuación (2.8) en $\mathbf{\Delta x}_i = \mathbf{x}_\infty - \mathbf{x}_i$ y despejando $\mathbf{x}_\infty$ se obtiene,

$$\mathbf{x}_\infty = \mathbf{x}_i + (\mathbf{I} - \mathbf{\Phi})^{-1} (\mathbf{x}_{i+1} - \mathbf{x}_i) \tag{2.9}$$

La Ecuación (2.9) se puede escribir como:

$$\mathbf{x}_\infty = \mathbf{x}_i + \mathbf{C} (\mathbf{x}_{i+1} - \mathbf{x}_i) \tag{2.10}$$

La cual es una estimación de la localización del Ciclo Límite con:

$$\mathbf{C} = (\mathbf{I} - \mathbf{\Phi})^{-1} \tag{2.11}$$

El proceso de solución del método DN se muestra en la Figura 2.2. Inicia con el vector de condiciones iniciales ($\mathbf{x}_0$); con el vector de condiciones iniciales se calculan algunos periodos de integración "ciclos iniciales"; el número de periodos iniciales de integración que se requieren depende de las características de sistema. En [Semlyen y Medina, 1995] se reporta que para sistemas bien amortiguados son suficientes 3 o 4 periodos de integración y en sistemas pobremente amortiguados 6 o 7. El resultado de los periodos iniciales de integración es el vector $\mathbf{x}_i$.

Con el vector $\mathbf{x}_i$ integra un periodo de tiempo (que es el ciclo base) para obtener el vector $\mathbf{x}_{i+1}$. Con los vectores $\mathbf{x}_i$ y $\mathbf{x}_{i+1}$ se obtiene un vector de errores ($\mathbf{err}$) que contiene el error de cada variable de estado; este vector se utiliza para calcular el error máximo ($Error$) que hay en una de las variables de estado del sistema. Si el $Error$ es mayor que una tolerancia (por lo general de

.

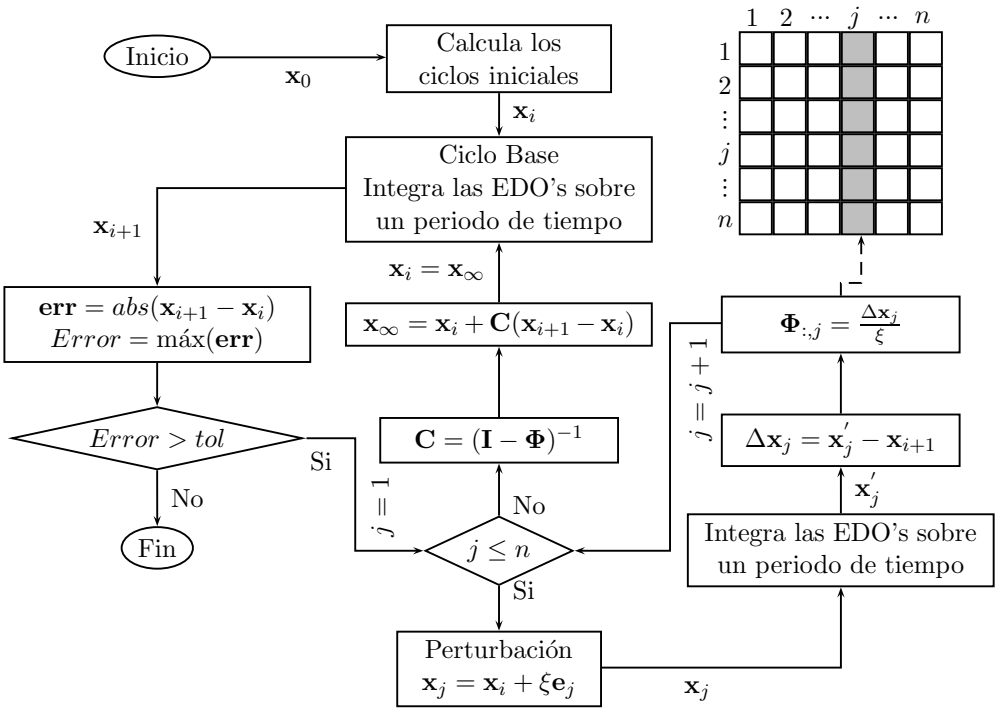


Figura 2.2: Método de DN tradicional.

10^{-10}) entonces, se procede a re-calcular la matriz $\mathbf{\Phi}$, de lo contrario termina el algoritmo con la solución que está en el vector $\mathbf{x}_\infty$.

Para calcular la matriz $\mathbf{\Phi}$ se realiza lo siguiente: para obtener la columna j de la matriz de identificación es necesario realizar una perturbación en la posición j del vector $\mathbf{x}_i$. Ésto se logra ejecutando la operación $\mathbf{x}_j = \mathbf{x}_i + \xi \mathbf{e}_j$. La perturbación ξ toma un valor muy pequeño (10^{-6}); $\mathbf{e}_j$ es la columna j de la matriz identidad. El siguiente paso es integrar un periodo de tiempo con los valores en las variables de estado que están en el vector $\mathbf{x}_j$; el resultado de la integración es $\mathbf{x}_j^{'}$. Con los vectores $\mathbf{x}_j^{'}$ y $\mathbf{x}_{i+1}$ se obtiene la columna j de matriz $\mathbf{\Phi}$ evaluando la ecuación $\mathbf{\Phi}_{:j} = \frac{\mathbf{x}_j^{'} - \mathbf{x}_{i+1}}{\xi}$.

Una vez realizado el proceso del cálculo de la matriz $\mathbf{\Phi}$ se procede a calcular la matriz $\mathbf{C}$, para ello se calcula $\mathbf{C} = (\mathbf{I} - \mathbf{\Phi})^{-1}$; con ella se obtienen los valores de las variables de estado en $\mathbf{x}_\infty$ mediante la Ecuación (2.10).

El nuevo vector $\mathbf{x}_i$ será el vector $\mathbf{x}_\infty$; con el nuevo vector $\mathbf{x}_i$ se integra un periodo de tiempo para obtener un nuevo vector $\mathbf{x}_{i+1}$. Con los vectores $\mathbf{x}_i$ y $\mathbf{x}_{i+1}$ se calcula el *Error* máximo que hay en el conjunto de variables de estado. El proceso de cálculo de: $\mathbf{\Phi}$, $\mathbf{C}$, $\mathbf{x}_i = \mathbf{x}_\infty$, $\mathbf{x}_{i+1}$ y *Error* se repite mientras el *Error* sea mayor a la tolerancia especificada.

2.3. Propuesta de la Complejidad Computacional del Método de Diferenciación Numérica

El método de DN tiene cuatro elementos principales que son: los periodos iniciales de integración, el cálculo del ciclo base, el cálculo de la matriz de identificación y el cálculo de la matriz $\mathbf{C} = (\mathbf{I} - \mathbf{\Phi})^{-1}$. En los cálculos de los periodos iniciales de integración, del ciclo base y la matriz de identificación es necesario integrar a lo largo de un número determinado de periodos de tiempo. Se utilizó el método de Runge-Kutta de cuarto orden para el proceso de integración numérica del conjunto resultante de EDO's.

Para hacer un estudio de la complejidad computacional del método DN es necesario hacer un estudio de la complejidad computacional de cada uno de sus elementos; con la complejidad de sus elementos se puede obtener la complejidad global del método DN.

La complejidad computacional que se propone está basada en las operaciones de punto flotante (flops) que se requieren para su ejecución. Durante el proceso del análisis se harán simplificaciones.

2.3.1. Complejidad Computacional del Método de Runge Kutta de Cuarto Orden

El método de Runge Kutta de cuarto orden (RK4) se muestra en la Ecuación (2.12) y requiere de la Ecuación (2.13) (que se puede escribir con la Ecuación (2.14)).

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4) \tag{2.12}$$

$$\begin{aligned} \mathbf{k}_1 &= f\left(t, \mathbf{x}_i\right) \\ \mathbf{k}_2 &= f\left(t + \frac{1}{2}h, \mathbf{x}_i + \frac{1}{2}\mathbf{k}_1 h\right) \\ \mathbf{k}_3 &= f\left(t + \frac{1}{2}h, \mathbf{x}_i + \frac{1}{2}\mathbf{k}_2 h\right) \\ \mathbf{k}_4 &= f\left(t + h, \mathbf{x}_i + \mathbf{k}_3 h\right) \end{aligned} \tag{2.13}$$

$$
\begin{aligned}
\mathbf{k}_1 &= f\left(t, \mathbf{x}_i\right) \\
\mathbf{y}_1 &= \mathbf{x}_i + \frac{h}{2}\mathbf{k}_1 \\
\mathbf{k}_2 &= f\left(t + \frac{h}{2}, \mathbf{y}_1\right) \\
\mathbf{y}_2 &= \mathbf{x}_i + \frac{h}{2}\mathbf{k}_2 \\
\mathbf{k}_3 &= f\left(t + \frac{h}{2}, \mathbf{y}_2\right) \\
\mathbf{y}_3 &= \mathbf{x}_i + h\mathbf{k}_3 \\
\mathbf{k}_4 &= f\left(t + h, \mathbf{y}_3\right)
\end{aligned}
\tag{2.14}
$$

La Ecuación (2.14) contiene los vectores $\mathbf{k}_1$, $\mathbf{k}_2$, $\mathbf{k}_3$ y $\mathbf{k}_4$ que son necesarios para calcular la Ecuación (2.12). Para obtener los valores de los vectores $\mathbf{k}_1$, $\mathbf{k}_2$, $\mathbf{k}_3$ y $\mathbf{k}_4$ es necesario evaluar una función f que recibe dos argumentos: el tiempo y un vector que usarán las EDO's para su evaluación. Los vectores $\mathbf{y}_1$, $\mathbf{y}_2$, $\mathbf{y}_3$ son vectores auxiliares necesarios para el cálculo de los vectores $\mathbf{k}_2$, $\mathbf{k}_3$ y $\mathbf{k}_4$.

2.3.2. Tiempo para la ejecución del método de Runge Kutta de cuarto orden

La representación de un sistema eléctrico de potencia requiere de n ecuaciones y n variables de estado. Para ejecutar el método de RK4 es necesario el Algoritmo 1 que se describe a continuación:

- En las líneas 3, 7, 11 y 15 se evalúan las EDO's para encontrar los vectores $\mathbf{k}_1$, $\mathbf{k}_2$, $\mathbf{k}_3$ y $\mathbf{k}_4$, respectivamente. Si para evaluar una ecuación es necesario de un tiempo t_1, para evaluar n ecuaciones se requiere un tiempo nt_1. Para ejecutar n veces las líneas 3, 7, 11 y 15 es necesario un tiempo $4nt_1$.

- Con las líneas 5, 9 y 13 se calculan los vectores $\mathbf{y}_1$, $\mathbf{y}_2$ y $\mathbf{y}_3$, respectivamente. Cada línea realiza una multiplicación y una suma; si para realizar la multiplicación y la suma se requiere de un tiempo t_2, entonces se requiere nt_2 para ejecutar n veces la línea 5 y el mismo tiempo para ejecutar n veces las líneas 9 y 13; por lo tanto, para ejecutar n veces las líneas 5, 9 y 13 se requiere de un tiempo $3nt_2$.

- La línea 17 se procesa n veces, por lo tanto, si para procesarla una vez es necesario de un tiempo t_3, entonces para procesarla n veces será necesario nt_3. Para la evaluación de la línea 17 es necesario tres multiplicaciones y cuatro sumas.

Algoritmo 1 Método de Runge Kutta de cuarto orden ejecutado de forma secuencial

1: **function** RUNGEK4($\mathbf{f}, \mathbf{x}, n, h, t$)
2: **for** $j = 1 : n$ **do**
3: $\mathbf{k}_1[j] =$ EVALÚA($\mathbf{f}[j], t, \mathbf{x}$)
4: **for** $j = 1 : n$ **do**
5: $\mathbf{y}_1[j] = \mathbf{x}(j) + \frac{h}{2}\mathbf{k}_1(j)$
6: **for** $j = 1 : n$ **do**
7: $\mathbf{k}_2[j] =$ EVALÚA($\mathbf{f}[j], t + \frac{h}{2}, \mathbf{y}_1)$)
8: **for** $j = 1 : n$ **do**
9: $\mathbf{y}_2[j] = \mathbf{x}(j) + \frac{h}{2}\mathbf{k}_2(j)$
10: **for** $j = 1 : n$ **do**
11: $\mathbf{k}_3[j] =$ EVALÚA($\mathbf{f}[j], t + \frac{h}{2}, \mathbf{y}_2)$)
12: **for** $j = 1 : n$ **do**
13: $\mathbf{y}_3[j] = \mathbf{x}(j) + h\mathbf{k}_3(j)$
14: **for** $j = 1 : n$ **do**
15: $\mathbf{k}_4[j] =$ EVALÚA($\mathbf{f}[j], t + h, \mathbf{y}_3)$)
16: **for** $j = 1 : n$ **do**
17: $\mathbf{y}[j] = \mathbf{x}[j] + \frac{h}{6}\left(\mathbf{k}_1[j] + 2\mathbf{k}_2[j] + 2\mathbf{k}_3[j] + \mathbf{k}_4[j]\right)$
18: **return y**

Es importante mencionar que no todas las ecuaciones tomarán el mismo tiempo para su evaluación, eso de debe a que el modelo de cada elemento del sistema eléctrico requiere ecuaciones diferentes. Para el análisis de complejidad computacional se tomará en cuenta que cada ecuación se evalúa en un tiempo t_1. También se debe considerar que no se puede conocer el número de operaciones de punto flotante que necesitará la evaluación de cada ecuación, debido a que algunas ecuaciones dependen de la conectividad del sistema y del modelo empleado para su descripción matemática.

El resultado de un análisis de complejidad computacional es una expresión que está en términos sus operaciones. En este análisis se usarán algunas variables que involucran tiempos, debido a que hay al menos un elemento del algoritmo en el que no se conoce el número de operaciones necesarias para resolver su tarea. Los tiempos t_2 y t_3 se pueden expresar en función del número de

operaciones de punto flotante que se realizan en un tiempo t_2 o en un tiempo t_3. El tiempo t_1 no puede expresarse en función del número de operaciones debido a que t_1 representa el tiempo para evaluar una EDO y las EDO's no tienen el mismo número de operaciones. Por lo anterior, durante el análisis de complejidad se usarán las variables t_1, t_2 y t_3 para expresar el tiempo (t_{rks}) para ejecutar el método de RK4. Sumando los tiempos que se obtuvieron del Algoritmo 1 y son: $4nt_1$, $3nt_2$ y nt_3 se tiene:

$$t_{rks} = (4t_1 + 3t_2 + t_3)n \tag{2.15}$$

El tiempo en la ejecución del método de RK4 es lineal, por lo tanto, la complejidad del algoritmo para ejecutar el método de RK4 es lineal.

2.3.3. Tiempo para la Integración de un Periodo de Tiempo

Para integrar un periodo de tiempo es necesario ejecutar el procedimiento que se muestra en el Algoritmo 2; en él se observa que:

- La línea 3 procesa m veces el Algoritmo 1 en un tiempo mt_{rks}, donde m es el número de divisiones del periodo.
- La línea 4 también se ejecuta m veces; esta línea incrementa en h el tiempo. El tiempo para procesar la línea 4 (considerando que se procesa en un tiempo cercano a t_2) se puede despreciar, ya que si se compara con el tiempo para integrar un periodo de tiempo (ver Ecuación (2.15)), se tiene que $t_2 \ll (4t_1 + 3t_2 + t_3)n$.

Por lo anterior, el tiempo t_p para integrar un periodo de tiempo es el que se muestra a continuación:

$$t_p = m(t_{rks}) = mn(4t_1 + 3t_2 + t_3) \tag{2.16}$$

2.3.4. Tiempo para los ciclos iniciales

Antes de comenzar con el método DN se requiere integrar un número inicial de periodos de tiempo. Para realizar esta tarea se realiza el Algoritmo 3. Las líneas 3 y 4 se procesan varias

Algoritmo 2 Método para integrar un periodo de tiempo

```
1: function INTEGRA-PERIODO(f, x, n, h, t, m)
2:     for j = 1 : m do
3:         x=RUNGEK4(f, x, n, h, t)
4:         t=t+h
5:     return x
```

veces (indicadas en la variable *periodos*). Si b es el número de periodos que se deben de integrar y despreciando el tiempo de la ejecución de la línea 4, entonces el Algoritmo 3 se ejecuta en un tiempo t_c, que se obtiene de multiplicar b por el tiempo t_p (que es el tiempo para obtener un periodo de integración) y su expresión es la siguiente:

$$t_c = b(t_p) = bmn(4t_1 + 3t_2 + t_3) \tag{2.17}$$

La complejidad para procesar los ciclos iniciales es lineal.

Algoritmo 3 Método para calcular los ciclos iniciales

```
1: function CICLOS-INICIALES(f, x, n, h, t, m, periodos, T)
2:     for j = 1 : periodos do
3:         x=INTEGRA-PERIODO(f, x, n, h, t, m)
4:         t = t + T
5:     return x
```

2.3.5. Tiempo para Calcular la Matriz de Identificación

Para calcular la matriz $\mathbf{\Phi}$ es necesario:

- Integrar n periodos de tiempo (ver línea 5 del Algoritmo 4). Para procesar la integración de los n periodos de tiempo es necesario un tiempo $nt_p = n^2m(4t_1 + 3t_2 + t_3)$.
- La línea 7 se evalúa n^2 veces, por lo tanto, si para evaluar un vez esa línea es necesario de un tiempo aproximado de t_2, entonces para evaluarla n^2 veces es necesario de un tiempo aproximado de n^2t_2. Este tiempo se puede despreciar dado que $n^2t_2 \ll n^2m(4t_1 + 3t_2 + t_3)$.
- Las líneas 4 y 8 se evalúan n veces en un tiempo aproximado de $2nt_2$, por lo tanto, se puede despreciar el tiempo de su evaluación.

Considerando lo anterior, el tiempo para procesar la matriz $\mathbf{\Phi}$ es:

$$t_m = n(t_p) = mnn(4t_1 + 3t_2 + t_3) = n^2 m(4t_1 + 3t_2 + t_3) \tag{2.18}$$

La complejidad del Algoritmo 4 para hallar la matriz $\mathbf{\Phi}$ es cuadrática.

Algoritmo 4 Método para calcular la matriz $\mathbf{\Phi}$ de forma secuencial

```
1: function MATRIZ-IDENTIFICACIÓN(f, x, x1, n, h, t, m)
2:     ξ = 1 × 10^−6
3:     for j = 1 : n do
4:         x[j] = x[j] + ξ
5:         y = INTEGRA-PERIODO(f, x, n, h, t, m)
6:         for k = 1 : n do
7:             Φ[k][j] = (y[k] − x1[k]) / ξ
8:         x[j] = x[j] − ξ
9:     return Φ
```

2.3.6. Tiempo para Calcular la Inversa de una Matriz Usando la Factorización LU

Para calcular la inversa de la matriz se puede usar la factorización LU descrita en [Chapra, 2006] y posteriormente un proceso de sustitución hacia adelante y hacia atrás. En el Algoritmo 5 se detalla el proceso de la factorización LU, el cual supone que los pivotes son diferentes de cero.

Para encontrar las operaciones (de punto flotante o flops) que utiliza el algoritmo y con ello la complejidad en función del número de operaciones se tiene:

La línea 4 del algoritmo se ejecuta $\sum_{i=1}^{n-1} i$ veces y la línea 7 lo hace $\sum_{i=1}^{n-1} i^2$ veces; esas dos líneas son las que procesan operaciones de punto flotante y tomando en cuenta que: $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$ y $\sum_{i=1}^{n-1} i^2 = \frac{n(n-1)(2n-1)}{6}$; el algoritmo para la factorización LU necesita N_c operaciones. Como se indica en la Ecuación (2.19).

$$N_c = \sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} i^2 = \frac{n(n-1)}{2} + \frac{n(n-1)(2n-1)}{6} = \frac{n^3}{3} - \frac{n}{3} \tag{2.19}$$

El término cúbico de la Ecuación (2.19) se obtiene al ejecutar la línea 7; esta línea realiza una multiplicación y una resta (dos operaciones), por lo tanto, se puede expresar el tiempo necesario para la factorización LU en función de t_2 y es: $t_{LU} = \frac{n^3}{3} t_2 - \frac{n}{3} t_2$.

Algoritmo 5 Factorización LU

```
1: function LU(A, n)
2:     for p = 1 : n − 1 do
3:         for r = p + 1 : n do
4:             factor = A[r][p]/A[p][p]
5:             A[r][p] = factor
6:             for c = p + 1 : n do
7:                 A[r][c] = A[r][c] − factor ∗ A[p][c]
8:     return A
```

En el Algoritmo 6 se presenta la sustitución hacia adelante y hacia atrás para encontrar la inversa de la matriz. Para el algoritmo es necesario tener la factorización LU en la matriz **A** y la matriz identidad en la matriz **X**. La inversa de la matriz **A** quedará en la matriz **X**; el algoritmo hace uso de un vector **b**.

En el algoritmo 6, el ciclo que está en la línea 2 ejecuta el código n veces, por lo tanto, el número de operaciones que se tengan en el resto del algoritmo se multiplica por n para obtener el número total de operaciones.

A continuación se indica el número de veces que se efectúan las operaciones de punto flotante.

- La línea 8 se procesa $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$ veces
- La línea 10 se ejecuta 1 vez
- La línea 14 se ejecuta $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$ veces
- La línea 15 se ejecuta $n - 1$ veces

Sumando todas las operaciones se tiene $(n-1) + 2\frac{n(n-1)}{2} + 1 = n^2$ y multiplicando por n esta última ecuación se obtiene el número de operaciones totales para la sustitución hacia adelante y hacia atrás para hallar la inversa de la matriz y es:

$$N_s = n^3 \tag{2.20}$$

En la línea 8 se ejecuta una multiplicación y una resta (dos operaciones); en la línea 14 se ejecuta una multiplicación y una suma (dos operaciones) y en la línea 15 se ejecuta una resta y una división (dos operaciones). Por lo tanto, el tiempo t_{sust} requerido para el proceso de sustitución hacia adelante y hacia atrás es $t_{sust} = n^3 t_2$.

Algoritmo 6 Sustitución hacia adelante y hacia atrás

```
1:  function SUSTITUCIÓN(A, I, n)
2:      for c = 1 : n do
3:          for k = 1 : n do
4:              b[k] = I[k][c]
5:          for i = 2 : n do
6:              suma = b[i]
7:              for j = 1 : i − 1 do
8:                  suma = suma − A[i][j] ∗ b[j]
9:              b[i] = suma
10:         X[n][c] = b[n]/A[n][n]
11:         for i = n − 1 : 1 do
12:             suma = 0
13:             for j = i + 1 : n do
14:                 suma = suma + A[i][j] ∗ X[j][c]
15:             X[i][c] = (b[i] − suma)/A[i][i]
16:     return X
```

Para obtener la inversa de la matriz usando la factorización LU se utiliza el proceso que se indica en el Algoritmo 7. La complejidad computacional del algoritmo se obtiene al sumar las expresiones (2.19) (al procesar la línea 9) y (2.20) (al procesar la línea 10) y es:

$$N_{inv} = N_c + N_s = \frac{n^3}{3} - \frac{n}{3} + n^3 = \frac{4n^3}{3} - \frac{n}{3} \tag{2.21}$$

Cuando n es muy grande, el término $\frac{n}{3}$ comparado con el término $\frac{4n^3}{3}$ es muy pequeño, por lo tanto, se puede despreciar y la complejidad para calcular la inversa está dada por:

$$N_{inv} = \frac{4n^3}{3} \tag{2.22}$$

Escribiendo la Ecuación (2.22) en función del tiempo t_2 se tiene que: $t_{inv} = \frac{4n^3}{3} t_2$. La complejidad para obtener la inversa de una matriz usando la factorización LU y la sustitución hacia adelante y hacia atrás es cúbica.

Algoritmo 7 Inversa de una matriz usando la factorización LU

```
1:  function UNITARIA(n)
2:      for r = 1 : n do
3:          for c = 1 : n do
4:              I[r][c] = 0
5:          I[r][r] = 1
6:      return I
7:  function INVERSA(A, n)
8:      I=UNITARIA(n)
9:      A=LU(A, n)
10:     X=SUSTITUCION(A, I, n)
11:     return X
```

2.3.7. Tiempo para Ejecutar el Método de Diferenciación Numérica

En el método de DN son necesarias otras rutinas que se trataron en las secciones anteriores, una de ellas es la rutina para encontrar el error máximo dentro de las variables de estado y otra es la rutina para evaluar la Ecuación (2.11), que será necesaria para calcular las variables de estado en el Ciclo Límite con la evaluación de la Ecuación (2.10).

El procedimiento que se muestra en el Algoritmo 8 encuentra el error máximo en una de las variables de estado del sistema. El número de operaciones de punto flotante es n, ya que la línea 2 se ejecuta solamente una vez y la línea 4 lo hace $n-1$ veces, que en suma son n operaciones.

Algoritmo 8 Cálculo del error máximo en las variables de estado

```
1:  function ERROR(x, x1, n)
2:      error = ABSOLUTO(x[0] − x1[0])
3:      for j = 2 : n do
4:          e = ABSOLUTO(x[j] − x1[j])
5:          if e > error then
6:              error = e
7:      return error
```

En el procedimiento que se muestra en el Algoritmo 9, que evalúa la Ecuación (2.10). Para ello es necesario:

- Procesar la operación $\mathbf{B} = \mathbf{I} - \mathbf{\Phi}$. Esta rutina se ejecuta en la línea 6 y comienza su definición en la línea 1. Para procesar esta operación se requiere de n operaciones de punto flotante, que es el número de veces que se ejecuta la línea 3.

- Ejecutar la rutina para obtener la inversa de una matriz, según se describe en el Algoritmo 7. La complejidad para obtener la inversa usando factorización LU es la definida por la Ecuación (2.21), que se puede aproximar mediante la Ecuación (2.22) y es $N_{inv} = \frac{4}{3}n^3$.

- Una vez que se tiene la matriz $\mathbf{C}$ es necesario evaluar la Ecuación (2.10), para ello es necesario el proceso que está de la línea 8 a la línea 12. Se observa que la línea 11 se procesa n^2 veces y la línea 12 lo hace n veces, por lo tanto, es necesario de $n^2 + n$ operaciones para evaluar la Ecuación (2.10).

El Algoritmo 9 requiere de n operaciones para ejecutar $\mathbf{B} = \mathbf{I} - \mathbf{\Phi}$, de $\frac{4n^3}{3} - \frac{n}{3}$ operaciones para obtener $\mathbf{C} = \mathbf{B}^{-1}$ y de $n^2 + n$ para evaluar $\mathbf{x}_\infty = \mathbf{x}_i + \mathbf{C}\left(\mathbf{x}_{i+1} - \mathbf{x}_i\right)$. El número de operaciones que son necesarias para calcular el valor de las variables de estado en el Ciclo Límite (conociendo la matriz $\mathbf{\Phi}$) es el siguiente:

$$Op_{lim} = n^2 + \frac{4n^3}{3} - \frac{n}{3} + n^2 + n = \frac{4n^3}{3} + 2n^2 + \frac{2n}{3} \tag{2.23}$$

Cuando n es muy grande, el término cuadrático y lineal se pueden eliminar, por lo que la Ecuación (2.23) se podrá aproximar como se muestra a continuación:

$$Op_{lim} \approx \frac{4n^3}{3} \tag{2.24}$$

Algoritmo 9 Calcula variables de estado en el ciclo límite

```
1:  function RESTA(I, Φ, n)
2:      for c = 1 : n do
3:          X[r][r] = I[r][r] − Φ[r][r]
4:      return X
5:  function CICLO-LIMITE(Φ, I, n, x, x1)
6:      B = RESTA(I, Φ)
7:      C = INVERSA(B, n)
8:      for r = 1 : n do
9:          suma = 0
10:         for c = 1 : n do
11:             suma = suma + C[r][c](x1[c] − x[c])
12:         x∞[r] = x[r] + suma
13:     return x∞
```

El Algoritmo 10 describe el proceso completo del método DN. De la línea 2 a la línea 7 se definen algunas variables importantes que necesita el método DN. Las variables son:

- n representa el número de ecuaciones y el número de variables de estado.
- T representa un periodo.
- m indica el número de divisiones del periodo.
- h es el paso de integración.
- t el el tiempo, en donde $t = 0$ representa el tiempo inicial.
- *periodos* representa el número de periodos que se integran inicialmente.

Algoritmo 10 Método de Diferenciación Numérica

```
1:  function DIFERENCIACIÓN-NUMÉRICA(f, x0)
2:      n=número-de-variables-de-estado
3:      T = 1/60
4:      m = 512
5:      h = T/m
6:      t = 0
7:      periodos = 8
8:      x =CICLOS-INICIALES(f, x0, n, h, t, m, periodos, T)
9:      x1 = INTEGRA-PERIODO(f, x, n, h, t, m)
10:     error = ERROR(x, x1, n)
11:     I = UNITARIA(n)
12:     while error > tolerancia do
13:         Φ = MATRIZ-IDENTIFICACIÓN(f, x, x1, n, h, t, m)
14:         x = CICLO-LIMITE(Φ, I, n, x, x1)
15:         x1 = INTEGRA-PERIODO(f, x, n, h, t, m)
16:         error = ERROR(x, x1, n)
17:     return x1
```

En el algoritmo 10:

- La línea 8 calcula los periodos iniciales de integración y requiere de un tiempo $t_c = bmn(4t_1 + 3t_2 + t_3)$;
- La línea 9 integra un periodo de tiempo (que es el ciclo base) y necesita un tiempo $t_p = mn(4t_1 + 3t_2 + t_3)$.

- La línea 10 encuentra el error máximo que hay en una de las variables de estado y el número de operaciones de punto flotante es n.
- Después de estas líneas sigue un ciclo que se analiza más adelante.

Para poder sumar los tiempos hasta este punto del algoritmo se hace la siguiente consideración: para hallar el error máximo, cada operación requiere una resta y el cálculo del valor absoluto de un número (dos operaciones) y el tiempo t_2 es el tiempo para realizar una multiplicación y una suma (dos operaciones), por lo tanto, se puede considerar que para encontrar el error máximo se necesita un tiempo nt_2.

El tiempo t_{ac} de ejecución del algoritmo hasta antes del ciclo de la línea 12 es $bmn(4t_1 + 3t_2 + t_3) + mn(4t_1 + 3t_2 + t_3) + nt_2$, que se puede escribir como $mn(b+1)(4t_1 + 3t_2 + t_3) + nt_2$. El término nt_2 se puede despreciar debido a que su valor es muy pequeño comparado con el resto de la expresión, por lo tanto, el tiempo t_{ac} se puede expresar como:

$$t_{ac} = (b+1)mn(4t_1 + 3t_2 + t_3) \tag{2.25}$$

El tiempo hasta entes del ciclo de la línea 12 se redujo al tiempo necesario para calcular $b+1$ periodos de integración.

En la línea 12 comienza un ciclo que ejecuta las líneas:

- 13: calcula la matriz $\mathbf{\Phi}$ con un tiempo de ejecución $t_m = n^2m(4t_1 + 3t_2 + t_3)$.
- 14: calcula las variables de estado en el Ciclo Límite con la cantidad de operaciones de punto flotante que se puede calcular con la ecuación $Op_{lim} = \frac{4n^3}{3} + 2n^2 + \frac{2n}{3}$ que para n muy grande se puede aproximar como $Op_{lim} \approx \frac{4n^3}{3}$. La última expresión es número de operaciones para calcular la inversa de la matriz y para sumar con el resto de los elementos que están en el ciclo se considera que cada operación requiere un tiempo t_2. El tiempo para obtener el Ciclo Límite es $t_{lim} = \frac{4n^3}{3}t_2 + 2n^2t_2 + \frac{2n}{3}t_2$ que con n muy grande se tiene $t_{lim} = \frac{4n^3}{3}t_2$.
- 15: ejecuta un periodo de integración con un tiempo de ejecución de $t_p = mn(4t_1 + 3t_2 + t_3)$.
- 16: calcula el error máximo de una de las variables de estado con un número de operaciones de punto flotante igual a n y un tiempo nt_2.

Considerando que el ciclo que inicia en la línea 12 se ejecuta p veces, entonces la suma de los tiempos de las líneas 13, 14, 15 y 16 se multiplica por p.

El tiempo de ejecución de una iteración del ciclo es $n^2m(4t_1+3t_2+t_3)+\frac{4n^3}{3}t_2+mn(4t_1+3t_2+t_3)+nt_2$. El último término se puede eliminar debido a que su contribución en el tiempo comparado con los demás términos es insignificante. El tiempo t_{dc} para ejecutar las p iteraciones del ciclo es:

$$t_{dc}=p\left(n^2m\left(4t_1+3t_2+t_3\right)+\frac{4n^3}{3}t_2+mn\left(4t_1+3t_2+t_3\right)\right) \tag{2.26}$$

El tiempo total para ejecutar el método DN es la suma de los tiempos $t_{ac}+t_{dc}=(b+1)mn(4t_1+3t_2+t_3)+p\left(n^2m\left(4t_1+3t_2+t_3\right)+\frac{4n^3}{3}t_2+mn\left(4t_1+3t_2+t_3\right)\right)$ que se puede escribir como:

$$t_{dn}=(b+1+p)\,mn\left(4t_1+3t_2+t_3\right)+p\left(n^2m\left(4t_1+3t_2+t_3\right)+\frac{4n^3}{3}t_2\right) \tag{2.27}$$

El término cúbico para hallar la inversa de la matriz hace que el algoritmo del método DN tenga una complejidad cúbica.

La Ecuación (2.27) se puede expresar en función del tiempo para integrar un periodo de tiempo $t_p=mn\left(4t_1+3t_2+t_3\right)$, y es:

$$t_{dn}=(b+1+p)\,t_p+p\left(nt_p+\frac{4n^3}{3}t_2\right) \tag{2.28}$$

La Ecuación (2.27) se puede escribir en términos del tiempo t_p para el cálculo de un periodo de integración y del tiempo para calcular la inversa de una matriz (t_{inv}), y es:

$$t_{dn}=(b+1+p)\,t_p+p\left(nt_p+t_{inv}\right)=(b+1+p(n+1))\,t_p+pt_{inv} \tag{2.29}$$

De la Ecuación (2.29) se observa lo siguiente:

1. El tiempo para ejecutar el algoritmo de DN y con ello llegar a la Solución del Estacionario Periódico del sistema, se reduce a calcular $b+1+p(n+1)$ periodos de integración y a calcular p veces la inversa de una matriz.

2. Al incrementar el valor de n se incrementa el número de periodos de tiempo a integrar y con ello se incrementa el tiempo de solución. El valor de n depende del número de variables de estado del sistema.

3. Dado que b es constante y puede tomar un valor de 8, conforme se incrementa n el término $b+1$ en la Ecuación (2.29) deja de ser importante, por el contrario, cuando n es pequeño, el término $b+1$ no se puede despreciar.

4. Cuando p (que indica las veces que se identifica la matriz $\mathbf{\Phi}$ y se calcula la inversa) se incrementa, el tiempo de solución también incrementa.

5. El tiempo para integrar un periodo de tiempo es muy importante, dado que si disminuye el tiempo mencionado, se reduce el tiempo en el término $(b+1+p(n+1))\,t_p$ de la Ecuación (2.29) y con ello se reduce el tiempo para el cálculo del método DN.

Para analizar el comportamiento de los elementos que contribuyen en la Ecuación 2.27, se hacen las siguientes consideraciones:

- t_2 realiza 2 operaciones y t_3 realiza 7 operaciones, por lo tanto, $t3 = 3.5t_2$,
- En el mejor de los casos $t_1 = t_2$, en otro caso $t_1 > t_2$. Para hacer una reducción en la Ecuación (2.27) se tomará el tiempo $t_1 = t_2$ (con ello se obtiene el mejor de los casos). El peor de los casos se presenta cuando se tiene la EDO que más operaciones necesita para su evaluación y todas las ecuaciones se consideren con el mismo número de operaciones.

Sustituyendo $t_3 = 3.5t_2$ y $t_1 = t_2$ en la Ecuación (2.27) se obtiene:

$$t_{dn} = (b+1+p)\,mn\,(4t_2+3t_2+3.5t_2) + p\left(n^2m\,(4t_2+3t_2+3.5t_2) + \frac{4n^3}{3}t_2\right) \tag{2.30}$$

Simplificando la Ecuación (2.30) se tiene:

$$t_{dn} = 10.5\,(b+1+p)\,mnt_2 + p\left(10.5mn^2t_2 + \frac{4n^3}{3}t_2\right) \tag{2.31}$$

Cuando $t_1 > t_2$, la constante 10.5 que está en la Ecuación (2.31) tomará un valor mayor.

La Ecuación (2.31) es imprecisa al considerar que $t_1 = t_2$, aunque aun así proporciona elementos importantes a considerar:

- El término $10.5\,(b+1+p)\,mnt_2$ tiene una complejidad lineal, el término $10.5mn^2t_2$ tiene una complejidad cuadrática y el término $\frac{4n^3}{3}t_2$ una complejidad cúbica.

- Si n incrementa (por que se analiza un sistema con más variables de estado), entonces los elementos que tienen complejidad cuadrática y cúbica ocuparán la mayor cantidad de tiempo.

- El término cuadrático $10.5mn^2t_2$ es el tiempo necesario para identificar la matriz $\mathbf{\Phi}$ y el término cúbico $\left(\frac{4n^3}{3}t_2\right)$ es el tiempo necesario para ejecutar el procedimiento para hallar la inversa de una matriz.

- Se puede revisar la contribución en el tiempo debido al término cuadrático y al término cúbico. Es claro que el término cúbico crecerá más rápido que el término cuadrático, pero debe haber un punto en que ambos términos son iguales, por lo tanto, si igualamos ambos términos se tiene que $10.5mn^2t_2 = \frac{4}{3}n^3t_2$. En $n = 7.875m$ las dos expresiones son iguales y como $m = 512$, entonces en $n = 4032$ las dos expresiones son iguales; para $n > 4032$ el término cúbico requerirá más tiempo que el término cuadrático y para valores $n < 4032$ el término cuadrático requiere mayor tiempo de proceso.

  Considerando el tiempo para calcular la inversa de la matriz y el tiempo para procesar la matriz $\mathbf{\Phi}$, se obtiene el tiempo t_{mds} de los elementos que mayor tiempo requieren en el método DN y es:

$$t_{mdn} = \frac{4}{3}t_2n^2\,(7.875m+n) \tag{2.32}$$

La Ecuación (2.32) clarifica de forma importante la contribución de la matriz de identificación en el tiempo total y la contribución de la inversa. En sistemas pequeños, el tiempo para el cálculo de la inversa es despreciable, por ejemplo para un sistema con 50 variables de estado $n = 50$ y $m = 512$. El factor $(7.875m + n)$ indica que la matriz de identificación contribuye en $7.875m = 4032$ mientras que la inversa contribuye con un valor de 50. Esto significa que el tiempo para procesar la matriz $\mathbf{\Phi}$ es 80.64 veces el tiempo para procesar la inversa de la matriz.

Para validar la Ecuación (2.27), se realizaron estudios de seis sistemas de prueba; los sistemas son los sistemas del IEEE modificados y contienen:

- primer caso: 14 nodos con 56 variables de estado

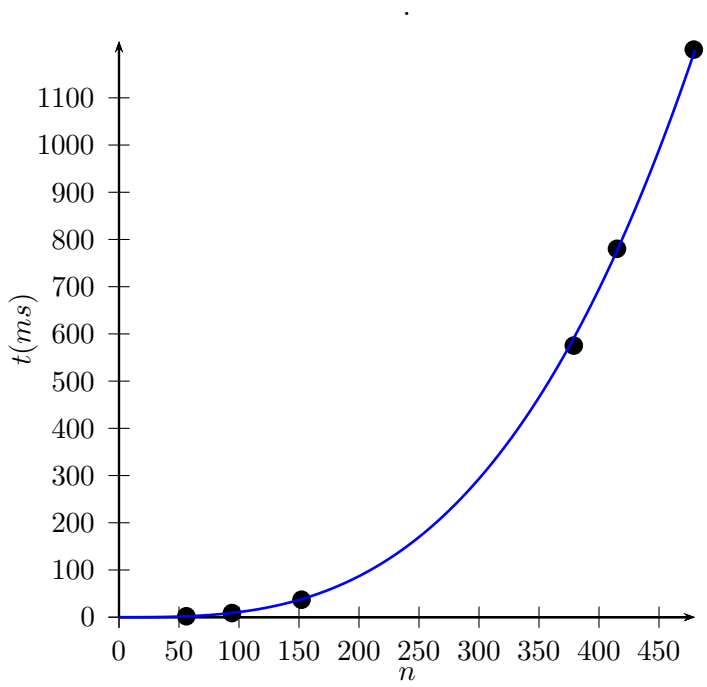


Figura 2.3: Tiempo requerido para obtener la inversa de la matriz.

- segundo caso: 30 nodos con 94 variables de estado
- tercer caso: 57 nodos con 152 variables de estado
- cuarto caso: 118 nodos con 379 variables de estado
- quinto caso: 118 nodos con 415 variables de estado
- sexto caso: 118 nodos con 479 variables de estado

Para verificar el comportamiento de la inversa de la matriz, se obtuvo el tiempo de la inversa (por medición) de los 6 casos de prueba. El resultado de esas mediciones está en los puntos de la gráfica de la Figura 2.3. Los puntos de la gráfica de la Figura 2.3 se comparan con la gráfica de la ecuación $t_{inv} = \frac{4}{3}n^3 t_2$, que es el tiempo para ejecutar la inversa de la matriz. El tiempo t_2 (que depende de la velocidad del elemento de proceso) se obtuvo por medición y se encontró que tiene un valor $t_2 = 8.14654535 \times 10^{-6} ms$. Por lo tanto, la ecuación que se graficó es: $t_{inv} = \frac{4}{3}\left(8.14654535 \times 10^{-6}\right) n^3$. En la Figura 2.3 se observa que los puntos (obtenidos por medición) coinciden con la gráfica de la ecuación $t_{inv} = \frac{4}{3}\left(8.14654535 \times 10^{-6}\right) n^3$.

Para comparar el comportamiento completo del método de DN con mediciones y con la Ecuación (2.27), es necesario obtener por medición algunas operaciones que son necesarias en la ecuación. El valor de la expresión $4t_1 + 3t_2 + t_3$ es necesaria en la Ecuación (2.27).

La Tabla 2.1 muestra información importante para comparar los tiempos del método DN usando la Ecuación 2.27 y usando mediciones; la tabla contiene la siguiente información:

- columna 1: número de nodos del sistema de prueba del IEEE.
- columna 2: número de variables de estado (n).
- columna 3: valor de la expresión $4t_1+3t_3+t_3$ (en milisegundos); este valor se obtuvo mediante mediciones.
- columna 4: valor del tiempo t_1 para cada uno de los sistemas de prueba del IEEE. Este tiempo se había considerado que sería diferente para cada sistema y en esta columna se confirma.
- columna 5: comparación de tiempo t_1 con el tiempo t_2 para cada sistema; en esta columna se observa que para cada sistema t_1 es diferente, por lo tanto, $\frac{t_1}{t_2}$ va a ser diferente y en todos los casos $t_1 > t_2$. Para obtener la Ecuación (2.30) se consideró que en el mejor de los casos $t_1 = t_2$, que en los seis casos de prueba no se cumple.
- columna 6: muestra el tiempo en milisegundos del proceso completo DN. El tiempo se obtuvo mediante medición.
- columna 7: muestra el tiempo en milisegundos del proceso completo DN. El tiempo se obtuvo evaluando la Ecuación (2.27), la cual se muestra a continuación para facilitar la lectura.

$$t_{dn} = (b + 1 + p)\, mn\, (4t_1 + 3t_2 + t_3) + p\left(n^2 m\, (4t_1 + 3t_2 + t_3) + \frac{4n^3}{3} t_2\right)$$

Donde,

$b = 8$, $m = 512$, $t_2 = 8.14654535 \times 10^{-6}$, $p = 4$ en el sistema de 118 nodos con 479 variables de estado y $p = 2$ en los otros sistemas.

Los valores que muestran las dos últimas columnas de la Tabla 2.1 difieren en promedio 0.56 %. Este valor muy pequeño.

Tabla 2.1: Comparación tiempos del método de DN usando la Ecuación (2.27) y mediante mediciones

Nodos	n	$(4t_1 + 3t_2 + t_3)(ms)$	$t_1(ms)$	$\frac{t_1}{t_2}$	$DN_L(ms)$	$DN_C(ms)$
14	56	$3.20123166 \times 10^{-4}$	$6.67926554 \times 10^{-5}$	8.19	1117	1132
30	94	$2.41598644 \times 10^{-4}$	$4.71615249 \times 10^{-5}$	5.79	2335	2331
57	152	$1.24860008 \times 10^{-4}$	$1.79768658 \times 10^{-5}$	2.20	3124	3137
118	379	$1.36747712 \times 10^{-4}$	$2.09487918 \times 10^{-5}$	2.57	21408	21588
118	415	$2.13485719 \times 10^{-4}$	$4.01332938 \times 10^{-5}$	4.92	39773	39701
118	479	$3.17296303 \times 10^{-4}$	$6.60859397 \times 10^{-5}$	8.11	154111	154882

La Figura 2.4 muestra la gráfica del tiempo para calcular la matriz $\mathbf{\Phi}$ y el tiempo para calcular la inversa de la matriz, necesaria para evaluar la ecuación $\mathbf{C} = (\mathbf{I} - \mathbf{\Phi})^{-1}$. La gráfica muestra los tiempos hasta 500 variables de estado; en la gráfica se observa que el tiempo para calcular la matriz $\mathbf{\Phi}$ es mucho mayor que el tiempo para calcular la inversa de la matriz.

Los puntos en la Figura 2.4 representan 6 casos de estudio; se observa que en 3 de ellos los puntos están sobre la gráfica del tiempo para encontrar la matriz $\mathbf{\Phi}$ y en los otros 3 casos los puntos difieren de la gráfica. Los puntos que no coinciden con la gráfica de tiempo para hallar $\mathbf{\Phi}$ se debe a que el término $4t_1 + 3t_2 + t_3$ que se consideró constante no lo es (ver Tabla 2.1).

En la Figura 2.6 se muestra la gráfica del tiempo para calcular la matriz $\mathbf{\Phi}$ y el tiempo para calcular la inversa con hasta 15000 variables de estado. La Ecuación (2.18) necesaria para hallar la matriz $\mathbf{\Phi}$ es $t_m = n^2 m(4t_1 + 3t_2 + t_3)$; donde m, n y $4t_1 + 3t_2 + t_3$ se consideran constantes. En la columna 3 de la Tabla 2.1 se observa que el tiempo de la expresión $4t_1 + 3t_2 + t_3$ no es constante, varía debido a que t_1 que es el tiempo para evaluar una ecuación dentro del conjunto de EDO's y las ecuaciones no se evalúan en el mismo tiempo, no es constante.

En las Figuras 2.5 y 2.6 se muestran cuatro gráficas; tres de ellas para el cálculo de $\mathbf{\Phi}$ (uno por cada valor de $4t_1 + 3t_2 + t_3$ en los sistemas de 118 nodos) y una para el cálculo de la inversa; con $4t_1 + 3t_2 + t_3 = 1.36747712 \times 10^{-4}$ se obtiene la gráfica $\mathbf{\Phi}_1$; con $4t_1 + 3t_2 + t_3 = 2.13485719 \times 10^{-4}$ se obtiene la gráfica $\mathbf{\Phi}_2$ y con $4t_1 + 3t_2 + t_3 = 3.17296303 \times 10^{-4}$ la gráfica $\mathbf{\Phi}_3$. Los puntos en la gráfica de la Figura 2.5 representan 6 casos de estudio y se observa que los 6 están en una de las tres gráficas de $\mathbf{\Phi}$.

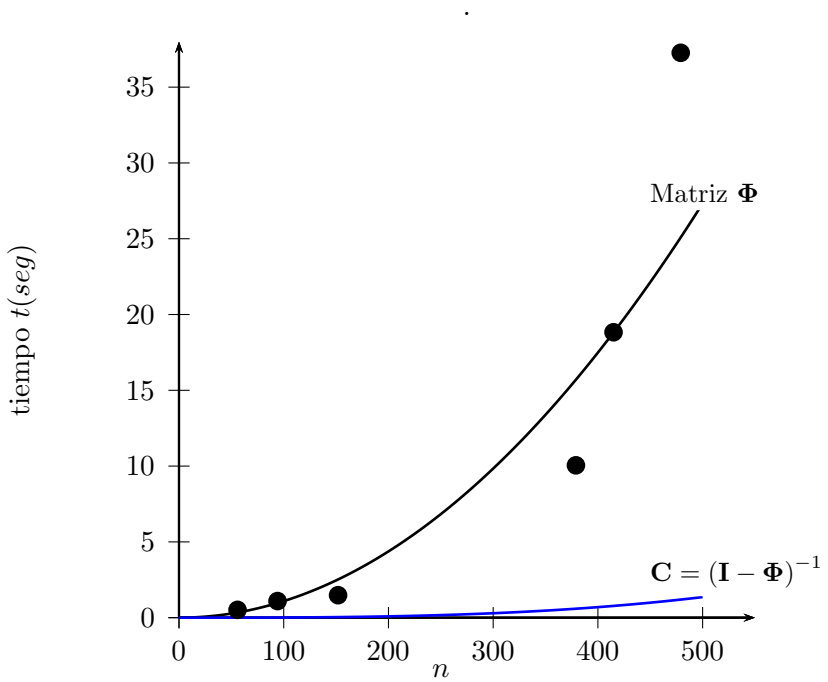


Figura 2.4: Tiempo para calcular la inversa de la matriz y tiempo para determinar $\mathbf{\Phi}$ hasta 500 variables de estado

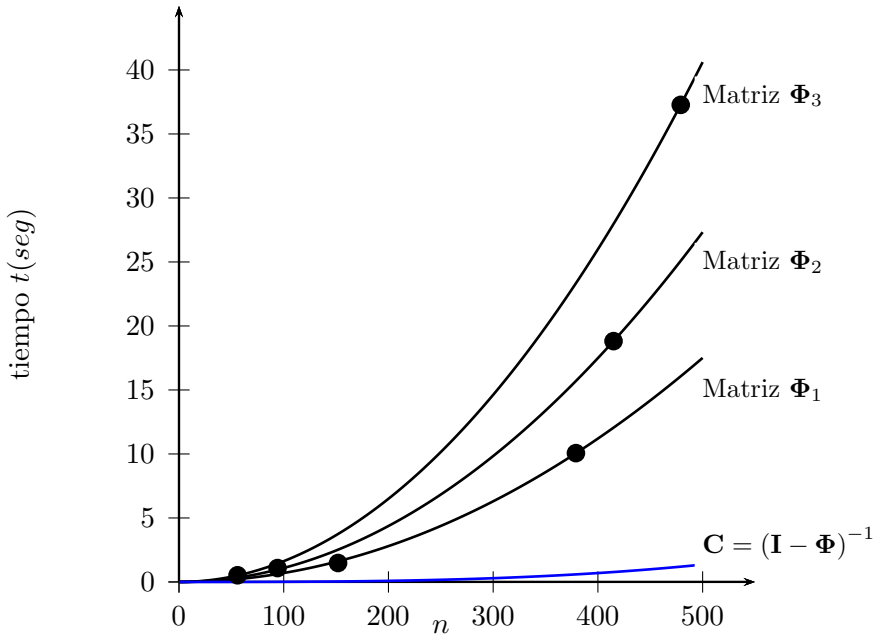


Figura 2.5: Tiempo para calcular la inversa de la matriz y tiempo para determinar $\mathbf{\Phi}$ hasta 500 variables de estado con diferentes tiempos en la ejecución de un periodo de integración (t_p)

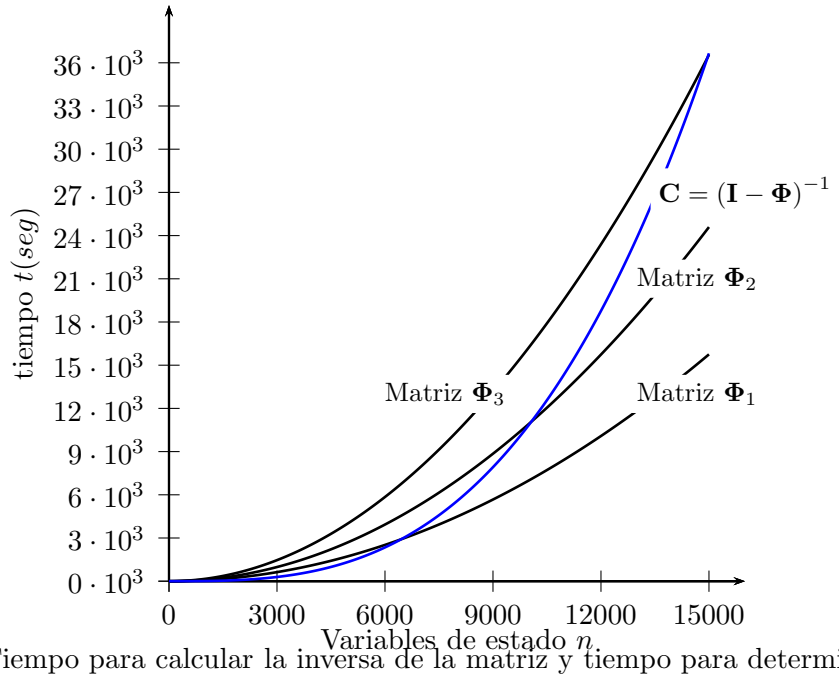


Figura 2.6: Tiempo para calcular la inversa de la matriz y tiempo para determinar $\mathbf{\Phi}$ hasta 15000 variables de estado

La Figura 2.6 muestra que conforme se incrementa el número de variables de estado en el sistema, el tiempo para calcular la inversa de la matriz toma mayor importancia, llegando a ser más el tiempo más importante del proceso completo.

2.4. Conclusiones

El resultado de la propuesta del análisis de complejidad computacional del método DN es la Ecuación (2.27) (que se escribe a continuación para facilitar la lectura). La ecuación nos proporciona elementos importantes para reducir el esfuerzo computacional o distribuir el esfuerzo computacional.

$$t_{dn} = (b + 1 + p)\, mn\, (4t_1 + 3t_2 + t_3) + p\left(n^2 m\, (4t_1 + 3t_2 + t_3) + \frac{4n^3}{3} t_2\right) \tag{2.33}$$

La inversa de la matriz proporciona un término cúbico en la Ecuación (2.27); este término cúbico hace que la complejidad computacional del método de DN sea cúbica.

El término $n^2 m\,(4t_1 + 3t_2 + t_3)$ en la Ecuación (2.27) representa el tiempo de proceso para el cálculo de la matriz $\mathbf{\Phi}$ y es el tiempo más importante hasta un número finito de variables de estado. En uno de los casos de prueba se observa (ver Figura 2.6) que ese número finito de variables de estado está entre 6000 y 7000.

Capítulo 3

Propuestas para Incrementar el Desempeño Computacional del Método de Diferenciación Numérica

3.1. Introducción

En el Capítulo 2 se realizó un análisis de complejidad del algoritmo DN; el resultado de ese análisis se resume en la Ecuación (2.27) y se repite en (3.1) para facilitar la lectura.

$$t_{dn} = (b+1+p)\,mn\,(4t_1+3t_2+t_3) + p\left(n^2 m\,(4t_1+3t_2+t_3) + \frac{4n^3}{3}t_2\right) \tag{3.1}$$

Donde,

- n es el número de variables de estado del sistema
- m es el número de pasos de integración en que se divide un periodo de tiempo T.
- b es el número de periodos de integración previos al ciclo base.
- p es el número de veces que es necesario identificar la matriz $\mathbf{\Phi}$.
- t_1 es el tiempo para evaluar una ecuación del conjunto EDO's.

- t_2 es el tiempo para realizar una multiplicación y una suma; cálculos necesarios para ejecutar el método de RK4.
- t_3 es el tiempo para realizar tres multiplicaciones y cuatro sumas (siete operaciones)en el método de RK4.
- $mn\,(4t_1 + 3t_2 + t_3)$ es el tiempo para integrar un periodo (T) de tiempo.
- $n^2 m\,(4t_1 + 3t_2 + t_3)$ es el tiempo necesario para identificar la matriz $\mathbf{\Phi}$.
- $\frac{4n^3}{3}t_2$ es el tiempo necesario para calcular la inversa de una matriz.

En este capítulo se presentan algunas propuestas que ayuden a reducir el tiempo para encontrar el estado estacionario periódico de un sistema eléctrico; mediante el método DN; se justifican con la Ecuación (3.1).

Se han desarrollado algunos trabajos relacionados al método DN con el propósito de reducir el tiempo de cómputo requerido para obtener la solución en estado estacionario periódico:

- En [Segundo y Medina, 2009] se logró reducir al utilizar medio periodo ($\frac{T}{2}$), en vez de integrar el periodo completo T. Una de las conclusiones de este trabajo es que se logró mejorar la eficiencia del método en un 100 %. Esta conclusión es correcta para sistemas donde la inversa de la matriz no contribuye de forma significativa en el tiempo de solución. Al usar el medio periodo de integración, el tiempo de solución estará dado por:

$$t_{dn} = (b + 1 + p)\,\frac{m}{2} n\,(4t_1 + 3t_2 + t_3) + p\left(\frac{m}{2} n^2\,(4t_1 + 3t_2 + t_3) + \frac{4n^3}{3} t_2\right) \tag{3.2}$$

- En [García, 2005] se procesa en paralelo la matriz $\mathbf{\Phi}$ usando la plataforma MPI. En este trabajo se reporta el caso de estudio de un sistema trifásico de 118 nodos con 3423 EDO's, reportando un speedup (ganancia) cercana a 110 usando 256 procesadores. En este trabajo se menciona (sin hacer un análisis de complejidad) que la inversa de la matriz, necesaria en el proceso de DN, contribuye a disminuir la ganancia del método DN.
- En [García-Olmos, 2013], se reportan resultados disminuyendo también el tiempo para calcular la matriz $\mathbf{\Phi}$, ahora usando procesamiento paralelo con Unidades de Procesamiento Gráfico.

Para un sistema de 3 nodos con 7 variables de estado, las ganancias obtenidas son muy cercanas a 7 en el proceso de cálculo de la matriz $\mathbf{\Phi}$ y en el proceso completo del método DN, inferior a 6.

- En [Medina y Ramos, 2003] se aplica procesamiento en paralelo a la matriz $\mathbf{\Phi}$ mediante la plataforma PVM obteniendo ganancias cercanas a 3.0.
- En [Peña y Medina,2010] se aplica el método DN al análisis de un sistema eólico de generación conectado al sistema externo. Se reporta una eficiencia relativa cercana a 4, en un estudio de 32 turbinas de viento. En el caso de estudio se compara el método DN usando procesamiento en paralelo con fuerza bruta.

En trabajos previos que han aplicado procesamiento en paralelo al método DN, las columnas de la matriz $\mathbf{\Phi}$ se obtienen a la vez de manera simultánea, ya que se pueden obtener de manera independiente unas de otras [García, 2001]. Unas metodologías tienen mejores resultados que otras, debido a los diferentes tiempos asociados con la transferencia de la información y la sincronización de todos los dispositivos involucrados en el procesamiento paralelo.

En este capítulo se proponen metodologías para reducir el tiempo de cómputo requerido para obtener la solución del estado estacionario periódico de sistemas eléctricos mediante el método DN.

3.2. Propuesta de Complejidad del Método de DN Aplicando Procesamiento Paralelo en la Matriz $\mathbf{\Phi}$

En el Algoritmo 11 se muestra el procedimiento para identificar la matriz $\mathbf{\Phi}$ en paralelo. En la línea 11 se preparan los elementos de proceso que estarán involucrados en el cálculo de las columnas. En la línea 12 se invoca el proceso que calcula las columnas de la matriz $\mathbf{\Phi}$ con n elementos de proceso; cada uno de los n elementos de proceso calcula una columna de la matriz $\mathbf{\Phi}$. En la línea 2 se detecta qué elemento procesará la columna j; la línea 5 copia del vector $\mathbf{x}$ al vector $\mathbf{z}$ (vector local al elemento de proceso j). En la línea 6 realiza una perturbación en la posición j del vector $\mathbf{z}$ y realiza la integración de un periodo de tiempo con el vector $\mathbf{z}$. Finalmente, la línea 9 calcula la columna j de la matriz $\mathbf{\Phi}$.

La línea 13 es fundamental, ya que sincroniza todos los elementos de proceso para que ningún elemento de proceso siga procesando hasta que todos hayan terminado hasta este punto. Una vez que se han calculado todas las columnas, se regresa la matriz $\mathbf{\Phi}$.

Para encontrar la complejidad del algoritmo de DN procesando las columnas de la matriz $\mathbf{\Phi}$ en forma paralela, se analizan las líneas en las que se realizan operaciones de punto flotante.

- La línea 6 se ejecuta una vez en el tiempo $\frac{t_2}{2}$
- La línea 7 se ejecuta una vez; el tiempo de ejecución es $mn(4t_1 + 3t_2 + t_3)$
- La línea 9 se ejecuta n veces en el tiempo nt_2
- El tiempo de sincronización dependerá de los elementos de proceso utilizados. Para efectos del estudio de complejidad se despreciará ese tiempo.

El tiempo (t_{mp}) para realizar el proceso de cálculo de $\mathbf{\Phi}$ en paralelo es el tiempo requerido para obtener una columna con un elemento de proceso. La Ecuación (3.3) determina el tiempo de cómputo asociado con el proceso. Despreciando los términos nt_2 y $\frac{t_2}{2}$ resulta en la Ecuación (3.4).

$$t_{mp} = mn(4t_1 + 3t_2 + t_3) + nt_2 + \frac{t_2}{2} \tag{3.3}$$

$$t_{mp} = mn(4t_1 + 3t_2 + t_3) \tag{3.4}$$

En la Ecuación (3.4) se observa de forma clara que la complejidad para identificar la matriz $\mathbf{\Phi}$ es lineal. La complejidad para hallar la matriz $\mathbf{\Phi}$ cambió de cuadrática (ver Ecuación (2.18) a lineal.

Modificando la línea 13 del Algoritmo 10 para que se ejecute en paralelo la identificación de $\mathbf{\Phi}$ se tiene una modificación en la complejidad del algoritmo de DN; el tiempo t_{dnp} es el que se muestra en la Ecuación (3.5). Se observa en esta ecuación que el término cuadrático para identificar $\mathbf{\Phi}$ se eliminó, ahora n tiene exponente 1.

$$t_{dnp} = (b + 1 + p)\, mn\, (4t_1 + 3t_2 + t_3) + p\left(nm\,(4t_1 + 3t_2 + t_3) + \frac{4n^3}{3}t_2\right) \tag{3.5}$$

Algoritmo 11 Método para calcular la matriz $\mathbf{\Phi}$ de forma paralela

```
1: function PROCESA-COLUMNA(f, x, x1, n, h, t, m, Φ)
2:     j =Elemento de proceso
3:     ξ = 1x10^-6
4:     for k = 1 : n do
5:         z[k] = x[k]
6:     z[j] = z[j] + ξ
7:     y = INTEGRA-PERIODO(f, z, n, h, t, m)
8:     for k = 1 : n do
9:         Φ[k][j] = (y[k] − x1[k]) / ξ
10: function MATRIZ-IDENTIFICACIÓN(f, x, x1, n, h, t, m)
11:     Prepara n elementos de proceso
12:     Paralelo <<< n >>>PROCESA-COLUMNA(f, x, x1, n, h, t, m, Φ)
13:     Sincroniza
14:     return Φ
```

La Ecuación (3.5) se puede escribir en función del tiempo requerido para procesar un periodo, según se muestra en la Ecuación (3.6). Esta ecuación es válida cuando se tienen n elementos de proceso (un elemento de proceso calcula una columna de $\mathbf{\Phi}$).

$$t_{dnp} = (b + 1 + p)\, t_p + p \left(t_p + \frac{4n^3}{3} t_2 \right) \tag{3.6}$$

Cuando se tienen q elementos de proceso se tienen 2 casos:

1. Cuando $q >= n$, el tiempo de procesamiento de $\mathbf{\Phi}$ es el tiempo en que un elemento de proceso identifica una columna.

2. Cuando $q < n$, algunos elementos de proceso identificarán $\frac{n}{q}$ y otros $\frac{n}{q} + 1$ columnas; la operación $\frac{n}{q}$ toma la parte entera.

El tiempo para obtener la matriz $\mathbf{\Phi}$ será el tiempo para procesar $\frac{n}{q} + 1$ columnas. El tiempo t_{mpq} para calcular la matriz $\mathbf{\Phi}$ usando procesamiento paralelo con el número de elementos de proceso menor al número de columnas de $\mathbf{\Phi}$ se muestra en las Ecuaciones (3.7) y (3.8). El tiempo para procesar el método DN usando procesamiento paralelo lo determina la Ecuación (3.9).

$$t_{mpq} = \frac{n^2 m}{q} (4t_1 + 3t_2 + t_3) + nm (4t_1 + 3t_2 + t_3) \tag{3.7}$$

$$t_{mpq} = \left(\frac{n}{q} + 1\right) mn\,(4t_1 + 3t_2 + t_3) \tag{3.8}$$

$$t_{dnp} = (b + 1 + p)\, mn\,(4t_1 + 3t_2 + t_3) + p\left(\left(\frac{n}{q} + 1\right) nm\,(4t_1 + 3t_2 + t_3) + \frac{4n^3}{3}t_2\right) \tag{3.9}$$

Tomando en cuenta que el tiempo t_p para integrar un periodo de tiempo es $t_p = mn\,(4t_1 + 3t_2 + t_3)$, la Ecuación 3.9 puede escribirse como:

$$t_{dnp} = (b + 1 + p)\, t_p + p\left(\left(\frac{n}{q} + 1\right) t_p + \frac{4n^3}{3}t_2\right) \tag{3.10}$$

3.3. Propuesta del Método de Diferenciación Numérica Selectivo

En el algoritmo DN convencional, es necesario calcular repetidamente la matriz $\mathbf{\Phi}$ durante el proceso de solución para determinar el estado estacionario periódico de la red eléctrica. El cálculo de la matriz $\mathbf{\Phi}$ requiere de gran esfuerzo computacional, para su determinación es necesario integrar n periodos de tiempo. En esta sección se propone una alternativa para reducir el cálculo de los n periodos de tiempo para identificar la matriz $\mathbf{\Phi}$.

Se debe recordar la matriz $\mathbf{\Phi}$ se identifica por columnas, de acuerdo al procedimiento descrito en la Sección 2.2. Una vez calculada la matriz $\mathbf{\Phi}$ se calcula la matriz $\mathbf{C}$ y el valor de las variables de estado en el ciclo límite $\mathbf{x}_\infty$ que en su caso se convierte en el vector $\mathbf{x}_i$. Una vez que se tiene el vector $\mathbf{x}_i$, se calcula un periodo de integración con las condiciones que se tienen en el vector $\mathbf{x}_i$ obteniendo el vector $\mathbf{x}_{i+1}$. Con los vectores $\mathbf{x}_{i+1}$ y $\mathbf{x}_i$ se obtiene un vector de error **err**, para las variables de estado.

En el vector **err** se puede encontrar que hay variables que tienen un error menor a una tolerancia de convergencia. Cuando se aplica el método DN convencional se observa que las columnas asociadas a las variables que tienen un error menor a la tolerancia establecida no sufren modificaciones importantes (se puede decir que las columnas permanecen prácticamente constantes).

Por lo anterior, se puede reducir el tiempo de ejecución del método de DN si no se calculan todas las columnas de la matriz de identificación después de calcularla por primera vez. La regla para calcular o no una columna de la matriz $\mathbf{\Phi}$ se muestra a continuación:

"Las columnas de la matriz $\mathbf{\Phi}$ que serán calculadas son aquellas asociadas a las variables que no han llegado al estado estacionario periódico y son aquellas en que el *error* asociado a la variable es mayor al pre-establecido para convergencia".

En la Figura 3.1 se muestra el algoritmo para encontrar la solución del estado estacionario periódico usando una matriz de identificación en la que no se calculan todas las columnas (matriz de identificación selectiva). El algoritmo comienza con el cálculo de los periodos iniciales de integración que toma el vector $\mathbf{x}_0$ y obtiene el vector $\mathbf{x}_i$, enseguida se calcula un periodo de integración para obtener el vector $\mathbf{x}_{i+1}$. Con los vectores $\mathbf{x}_i$ y $\mathbf{x}_{i+1}$ se obtiene un vector **err** que contiene la información el error de cada una de las variables de estado y el *Error* máximo (que es el valor del error asociado a la variable con mayor error). Cuando *Error* mayor que uno tolerable se procede a calcular la matriz $\mathbf{\Phi}$ y de lo contrario significa que todas las variables han llegado al estado estacionario periódico.

Una vez que se ha revisado que es necesario calcular la matriz de identificación, debido a que se cumple $Error > tol$, se procede a calcular $\mathbf{\Phi}$. Para ello es necesaria la condición $j = 1, 2, ..., n$ que indicará que j tomará los valores desde 1 hasta n. Luego de esta condición, es necesario la condición que indicará si la columna j será o no calculada.

La condición para calcular o no una columna de la matriz $\mathbf{\Phi}$ es una condición compuesta de dos condiciones:

1. La primera condición es $CDN = 0$; esta condición se cumple cuando se va a calcular $\mathbf{\Phi}$ por primera vez, lo que significa que todas las columnas de la matriz $\mathbf{\Phi}$ se calculan.

2. La segunda condición es $\mathbf{err}[j] > tolerancia$; la condición $\mathbf{err}[j] > tolerancia$ indica que el error asociado al número de variable de estado j es mayor a una tolerancia y se debe calcular la columna j de $\mathbf{\Phi}$.

Si no se cumple una de las dos condiciones anteriores, entonces no se calcula la columna j de la matriz $\mathbf{\Phi}$ y se usa la columna j de la matriz $\mathbf{\Phi}$ de la iteración anterior.

Después de calcular la matriz $\mathbf{\Phi}$, se procede a calcular $\mathbf{C} = (\mathbf{I} - \mathbf{\Phi})^{-1}$ y $\mathbf{x}_\infty = \mathbf{x}_i + \mathbf{C}(\mathbf{x}_{i+1} - \mathbf{x}_i)$. Para continuar con el proceso, se realiza la operación $\mathbf{x}_i = \mathbf{x}_\infty$, luego se calcula un periodo de integración para encontrar nuevamente el vector $\mathbf{x}_{i+1}$ para encontrar nuevamente el vector **err**

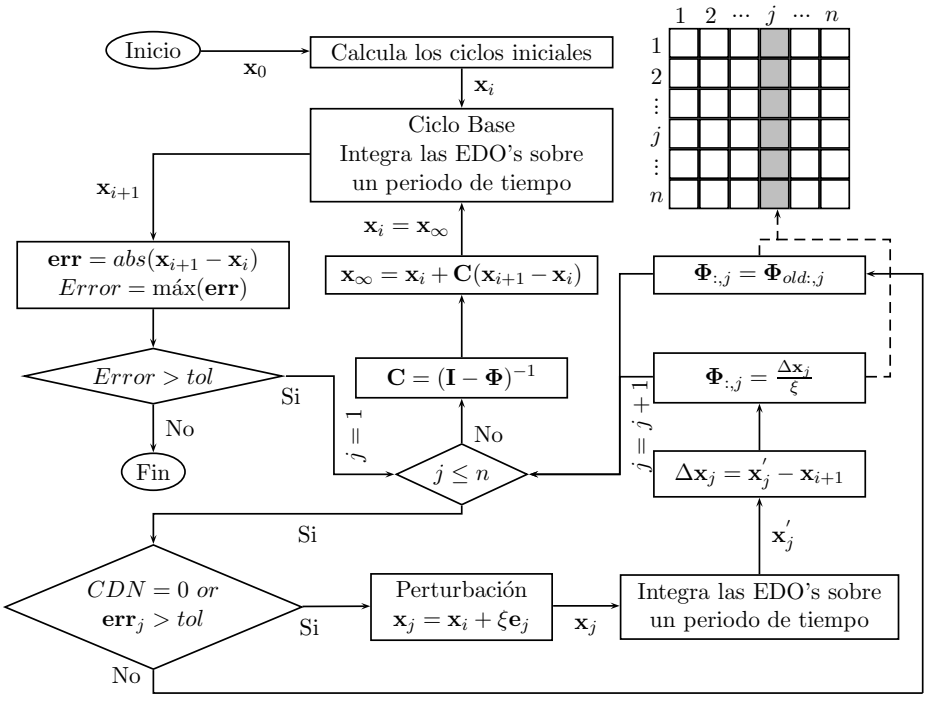


Figura 3.1: Método de diferenciación numérica selectivo.

y la variable $Error$. El proceso sigue hasta que todas las variables alcancen al estado estacionario periódico.

3.3.1. Complejidad del Método de Diferenciación Numérica Selectivo

En el Algoritmo 12 se muestra el procedimiento para obtener la matriz $\mathbf{\Phi}$. La función para obtener $\mathbf{\Phi}$ definida en el algoritmo recibe tres variables adicionales a las que se reciben en la función que se define en el Algoritmo 3 (que también encuentra la matriz $\mathbf{\Phi}$); las variables adicionales son: $\mathbf{\Phi}$, **err** y CDN. La variable $\mathbf{\Phi}$ contiene la matriz $\mathbf{\Phi}$ que se calculó en la iteración anterior (o contiene cualquier valor en su primera iteración); la variable **err** contiene el error de cada una de las variables de estado y la variable CDN contiene el número de veces que se ha calculado la matriz $\mathbf{\Phi}$ (su valor inicia en cero).

Las líneas 5, 6, 8 y 9 del Algoritmo 12 ejecutan operaciones de punto flotante. La línea 6 es la que requiere más tiempo (que es el tiempo para integrar un periodo de tiempo: $t_p = mn(4t_1 + 3t_2 + t_3)$); los tiempos para ejecutar las otras líneas se pueden despreciar, ya que requieren un tiempo muy pequeño comparado con el tiempo en que se ejecuta la línea 6.

La primera vez que se ejecuta la función definida en el Algoritmo 12 requiere un tiempo

de ejecución igual al que se obtuvo del Algoritmo 3 y es el que se muestra en la Ecuación (2.18); para facilitar la lectura la Ecuación (2.18) se escribe en 3.11. Las siguientes veces que se ejecuta la función del Algoritmo 12 el tiempo (t_{ms}) de ejecución será el que se muestra en la Ecuación (3.12). La variable k_p es el número de columnas que no se calcularon; cuando se calculan todas las columnas de $\mathbf{\Phi}$, entonces $k_p = 0$ y las expresiones 3.12 y 3.11 serán equivalentes.

$$t_m = n(t_p) = mnn(4t_1 + 3t_2 + t_3) = n^2 m(4t_1 + 3t_2 + t_3) \tag{3.11}$$

$$t_{ms} = n(t_p) = (n - k_p)mn(4t_1 + 3t_2 + t_3) \tag{3.12}$$

Algoritmo 12 Método para calcular la matriz $\mathbf{\Phi}$ para el método DNS

```
1:  function MATRIZ-IDENTIFICACIÓN(f, x, x1, n, h, t, m, err, Φ, CDN)
2:      ξ = 1x10^-6
3:      for j = 1 : n do
4:          if CND = 0 or err[j] > tolerancia then
5:              x[j] = x[j] + ξ
6:              y = INTEGRA-PERIODO(f, x, n, h, t, m)
7:              for k = 1 : n do
8:                  Φ[k][j] = (y[k] − x1[k]) / ξ
9:              x[j] = x[j] − ξ
10:     return Φ
```

El Algoritmo 13 muestra el procedimiento para encontrar el estado estacionario periódico de un sistema eléctrico usando el método DN y el cálculo de la matriz $\mathbf{\Phi}$ de forma selectiva. La diferencia con el Algoritmo 10 está en tres elementos; el primero es que en el Algoritmo 13 es necesario el vector **err** que contiene el error de cada una de las variables de estado (se calcula en las líneas 21 y 28 y se ocupa en las líneas 24 y 29 para hallar el error máximo y en la línea 25 para calcular $\mathbf{\Phi}$); el segundo es la forma en que se calcula la matriz $\mathbf{\Phi}$ (línea 25) y el tercer elemento es la variable CDN que cuenta las veces que se ha calculado $\mathbf{\Phi}$.

La variante en el tiempo de ejecución de los Algoritmos 10 y 13 está en la forma en que se obtiene $\mathbf{\Phi}$, por lo tanto, si en la Ecuación (3.1) se modifica el tiempo que se invierte en el cálculo de $\mathbf{\Phi}$ (de forma regular) por el tiempo de ejecución en su forma selectiva se tendrá el tiempo total de ejecución del algoritmo de DN de forma selectiva (DNS).

De la Ecuación (3.1) se obtiene el tiempo t_{pm} necesario para calcular p veces $\mathbf{\Phi}$ en el método de DN y es:

$$t_{pm} = pn^2m\left(4t_1 + 3t_2 + t_3\right) \tag{3.13}$$

En el Algoritmo 12 no se calculan todas las columnas de $\mathbf{\Phi}$, posterior a la primera iteración. El número de columnas que no se calculan depende de la dinámica del sistema eléctrico, por lo tanto, no se puede indicar el número columnas que no se calculan en cada iteración. Para hallar una ecuación en términos de variables se propone que la serie c_2, c_3, c_4,...,c_p sea el número de columnas que no se calculan en las iteraciones: $2, 3, 4...p$, respectivamente. El tiempo para calcular la matriz $\mathbf{\Phi}$ en cada iteración se muestra a continuación:

- Iteración 1: $n\left(nm\left(4t_1 + 3t_2 + t_3\right)\right)$
- Iteración 2: $\left(n - c_2\right)\left(nm\left(4t_1 + 3t_2 + t_3\right)\right)$
- Iteración 3: $\left(n - c_3\right)\left(nm\left(4t_1 + 3t_2 + t_3\right)\right)$
- ...
- Iteración p: $\left(n - c_p\right)\left(nm\left(4t_1 + 3t_2 + t_3\right)\right)$

Sumando los tiempos de la lista anterior, se tiene la Ecuación (3.14) que determina el tiempo t_{pms} para calcular p veces $\mathbf{\Phi}$ de forma selectiva.

$$\begin{aligned} t_{pms} = {} & n\left(nm\left(4t_1 + 3t_2 + t_3\right)\right) + \\ & \left(n - c_2\right)\left(nm\left(4t_1 + 3t_2 + t_3\right)\right) + \\ & \left(n - c_3\right)\left(nm\left(4t_1 + 3t_2 + t_3\right)\right) + \\ & ...+ \\ & \left(n - c_p\right)\left(nm\left(4t_1 + 3t_2 + t_3\right)\right) \end{aligned} \tag{3.14}$$

Simplificando la Ecuación (3.14) se obtiene:

$$t_{pms} = pn^2m\left(4t_1 + 3t_2 + t_3\right) - \left(c_2 + c_3 + ... + c_p\right)nm\left(4t_1 + 3t_2 + t_3\right) \tag{3.15}$$

La Ecuación (3.15) se puede escribir como:

$$t_{pms} = pn^2m\left(4t_1 + 3t_2 + t_3\right) - \sum_{j=2}^{n} c_j nm\left(4t_1 + 3t_2 + t_3\right) \tag{3.16}$$

Comparando las expresiones 3.13 y 3.16 se observa que 3.16 contiene la Ecuación (3.13) menos el término $\sum_{j=2}^{n} c_j nm\left(4t_1 + 3t_2 + t_3\right)$.

De la Ecuación (2.16) se tiene que un periodo de tiempo ocupa un tiempo $t_p = nm\left(4t_1 + 3t_2 + t_3\right)$; por lo tanto, la Ecuación (3.1) se puede escribir como se muestra en la Ecuación (3.17); y las (3.13) y (3.16) de acuerdo a las Ecuaciones (3.18) y (3.19), respectivamente,

$$t_{dn} = \left(b + 1 + p\right)t_p + p\left(nt_p + \frac{4n^3}{3}t_2\right) \tag{3.17}$$

$$t_{pm} = pnt_p \tag{3.18}$$

$$t_{pms} = pnt_p - \sum_{j=2}^{p} c_j c_p \tag{3.19}$$

El tiempo para encontrar la solución en estado estacionario periódico usando la matriz de identificación de forma selectiva se determina mediante la Ecuación (3.20). El término $\sum_{j=2}^{p} c_j t_p$ es la reducción que se logra del método DN al procesar $\mathbf{\Phi}$ de forma selectiva.

$$t_{dns} = \left(b + 1 + p\right)t_p + p\left(nt_p + \frac{4n^3}{3}t_2\right) - \sum_{j=2}^{p} c_j t_p \tag{3.20}$$

La Ecuación (3.20) se puede escribir como:

$$t_{dns} = \left(b + 1 + p\right)t_p + p\frac{4n^3}{3}t_2 + \sum_{j=1}^{p}\left(n - c_j\right)t_p \tag{3.21}$$

Donde, $c_1 = 0$ y c_j es el número de columnas que no se calculan en la j-ésima iteración.

El método DNS mínimo es el método de DNS que calcula solamente una columna de la matriz $\mathbf{\Phi}$ en cada una de las iteraciones que le suceden a la primera.

En la Figura 3.2 se ven las gráficas del tiempo en que se ejecuta el método de DN tradicional, método de DN selectivo y el método DN selectivo mínimo. En las gráficas se consideró lo siguiente:

Algoritmo 13 Método de diferenciación numérica selectivo

```
1: function ERROR2(err, n)
2:     error = err[1]
3:     for j = 2 : n do
4:         if err[j] > Error then
5:             error = err[j]
6:     return error
7: function ERROR-VARIABLES(x, x1, n)
8:     for j = 1 : n do
9:         err= ABSOLUTO(x[j] − x1[j])
10:    return err
11: function DIFERENCIACIÓN-NUMÉRICA(f, x0)
12:    n=número-de-variables-de-estado
13:    T = 1/60
14:    m = 512
15:    h = T/m
16:    t = 0
17:    periodos = 8
18:    CDN = 0
19:    x =CICLOS-INICIALES(f, x0, n, h, t, m, periodos, T)
20:    x1 = INTEGRA-PERIODO(f, x, n, h, t, m)
21:    err = ERROR-VARIABLES(x, x1, n)
22:    error = ERROR2(err, n)
23:    I = UNITARIA(n)
24:    while error > tolerancia do
25:        Φ = MATRIZ-IDENTIFICACIÓN(f, x, x1, n, h, t, m, err, Φ, CDN)
26:        x = CICLO-LIMITE(Φ, I, n, x, x1)
27:        x1 = INTEGRA-PERIODO(f, x, n, h, t, m)
28:        err = ERROR-VARIABLES(x, x1, n)
29:        error = ERROR2(err, n)
30:        CND = CND + 1
31:    return x1
```

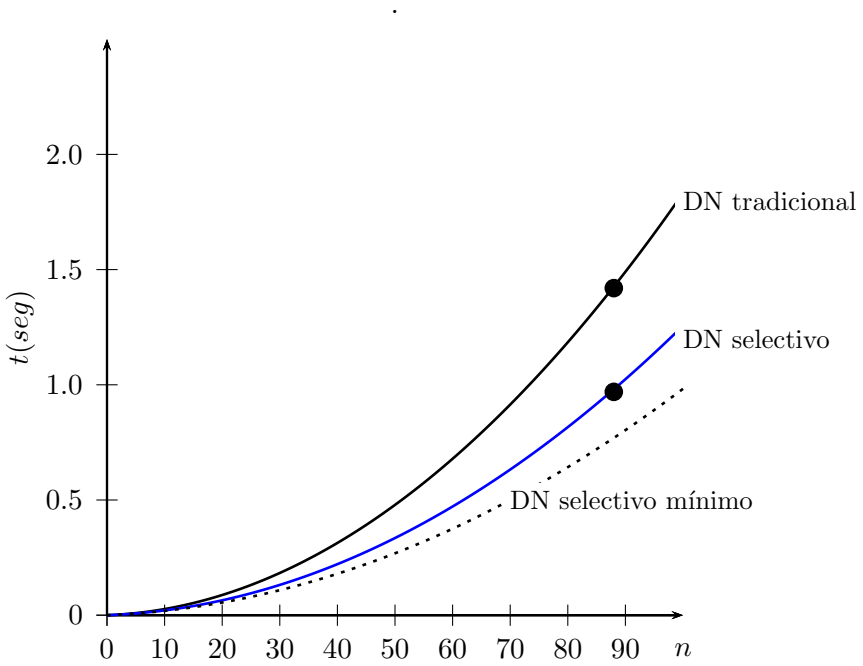


Figura 3.2: Tiempo para determinar el estado estacionario periódico con DN tradicional y con DNS hasta 100 variables de estado

- La matriz $\mathbf{\Phi}$ se calcula dos veces, por lo tanto, $p = 2$ en las Ecuaciones 3.20 y 3.10,
- $t_p = 1.67539 \times 10^{-4} ms$,
- $t_2 = 8.14654535 \times 10^{-6} ms$

Con las consideraciones anteriores se tiene un rango de posibilidades en el que puede trabajar el método de DN selectivo. Los límites de este rango son el mejor y peor caso, respectivamente, descritos a continuación:

- El mejor caso (método selectivo mínimo) es cuando se calcula solamente una columna de la matriz $\mathbf{\Phi}$ en la segunda iteración. La gráfica del tiempo de ejecución del método DN selectivo para este caso se presenta con línea punteada de la Figura 3.2.
- El peor caso se presenta cuando se tienen que calcular todas las columnas de la matriz $\mathbf{\Phi}$ en la segunda iteración. Este caso es igual al método DN tradicional.

En [Magaña y Medina,2015] se reportó un estudio donde se calcula el 33 % de las columnas en la segunda iteración para el cálculo de $\mathbf{\Phi}$ en un sistema con 88 variables de estado. En la Figura

3.2 se muestra el tiempo requerido por el método DN selectivo considerando un 33 % en el cálculo de las columnas de la matriz $\mathbf{\Phi}$.

Los puntos en la gráfica de la Figura 3.2 son las lecturas de la ejecución del algoritmo de DN tradicional y del algoritmo de DN selectivo en el sistema con 88 variables de estado; se observa que los puntos están en las gráficas que les corresponden.

La reducción de los tiempos al usar el método de DN selectivo depende del número de iteraciones en que se calcula la matriz $\mathbf{\Phi}$ y de las reducciones de los cálculos de las columnas en cada iteración. Cuando $p = 2$, en el mejor de los casos, el tiempo para llegar a la solución es cercana a la mitad del tiempo en que se ejecuta el método de DN tradicional y en el peor de los casos no hay reducción en el tiempo de solución. Cuando $p = 3$, en el mejor de los casos (que se presenta cuando se calcula una columna en la segunda y tercera iteración), el tiempo para llegar a la solución es cercana a un tercio del tiempo para ejecutar el método de DN tradicional. De forma general, el tiempo para ejecutar el método de DN selectivo (t_{dns}) está en un rango tiempos comparado con el tiempo para ejecutar el método de ND tradicional (t_{dn}). La relación de estos tiempos se presenta en la Ecuación (3.22).

$$t_{dns} = \left(\frac{1}{p}, 1\right) t_{dn} \tag{3.22}$$

La Figura 3.3 se presenta una comparación del método de DN tradicional y del método de DN selectivo. Las consideraciones son las siguientes:

- La matriz $\mathbf{\Phi}$ se calcula dos veces, por lo tanto, $p = 2$ en las Ecuaciones 3.20 y 3.10,
- $t_p = 0.82 \times 10^{-4} ms$,
- $t_2 = 8.14654535 \times 10^{-6} ms$

En la Figura 3.3 se observa que la línea punteada que representa el método de DN selectivo mínimo (que en el el caso de 2 iteraciones para hallar la matriz $\mathbf{\Phi}$ es el que solamente calcula una columna en la segunda iteración) esta muy cercana al método DN selectivo. Las dos gráficas están casi juntas debido que solamente se calculan 6 columnas de 357 posibles (que representa el 1.68 % del total de columnas).

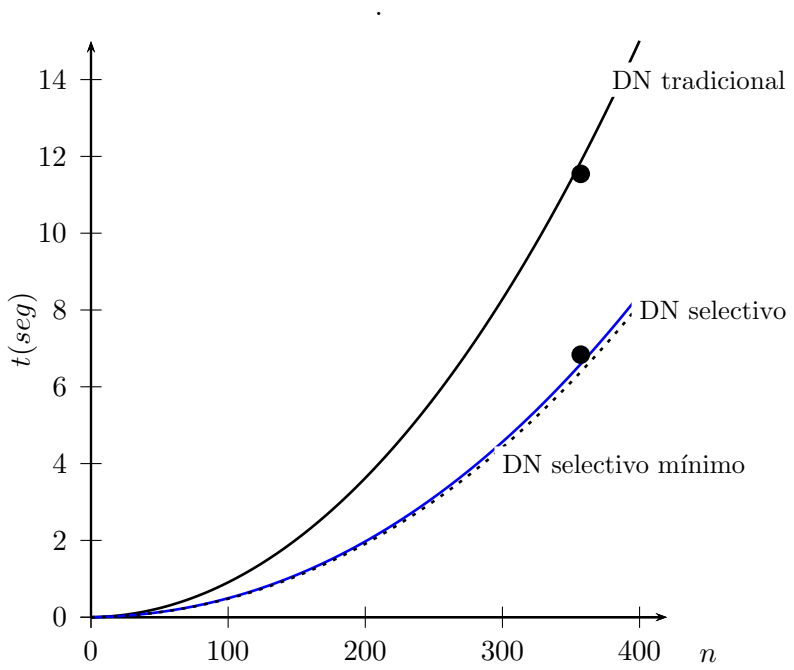


Figura 3.3: Tiempo para determinar el estado estacionario periódico con DN tradicional y con DNS hasta 400 variables de estado

Los puntos de la gráfica de la Figura 3.3 representan los tiempos para alcanzar la solución de un sistema con 357 variables de estado usando las metodologías DN tradicional y DN selectivo; se observa que ambos puntos están muy cercanos a las gráficas correspondientes.

Las Figuras 3.3 y 3.2 tienen como característica común lo siguiente:

- Ambas figuras tienen tres gráficas que representan: el tiempo de ejecución del método DN tradicional, el tiempo del método DN selectivo y del tiempo del método DN selectivo mínimo.
- La gráfica del método selectivo está entre la región del tiempo del método DN selectivo mínimo (línea punteada) y el tiempo del método de DN tradicional.

3.4. Propuesta de Diferenciación Numérica Selectiva usando Procesamiento en Paralelo

En el proceso del cálculo para la determinación del estado estacionario periódico, el elemento que requiere mayor esfuerzo computacional es el cálculo de la matriz $\mathbf{\Phi}$ (hasta cierta cantidad

de variables de estado). El proceso de cálculo de **Φ** se puede realizar en paralelo. La cantidad de elementos de proceso que se pueden usar de forma simultánea es n, donde n es el número de variables de estado del sistema. Para sistemas de gran escala, la cantidad de variables de estado es muy superior a los elementos de proceso disponibles en los sistemas de cómputo.

Cuando se calcula **Φ** de forma selectiva, se reduce el número de columnas que se tienen que calcular de la matriz **Φ**, dando como resultado una disminución del número de elementos de proceso necesarios para calcular **Φ**. Si en la iteración j del cálculo de la matriz de identificación no se calculan c_j columnas, entonces para identificar la matriz **Φ** se requieren $n - c_j$ elementos de proceso.

El proceso se puede realizar con cualquier arquitectura de procesamiento en paralelo, con el inconveniente técnico de los tiempos de la transferencia de la información entre los distintos dispositivos involucrados en el procesamiento en paralelo y de la sincronización de todos los elementos de proceso. De las expresiones 3.21 y 3.10 se puede obtener la Ecuación (3.23) para el proceso de DN usando matriz de identificación selectiva y procesamiento en paralelo.

$$t_{dns} = (b + 1 + p)\, t_p + \sum_{j=1}^{p} \left(\frac{n - c_j}{q} + 1 \right) t_p + p \frac{4n^3}{3} t_2 \tag{3.23}$$

Donde,

- $n - c_j$ es el número de columnas que se deben de calcular.
- q es el número de elementos de proceso.
- $\frac{n-c_j}{q}$ se toma la parte entera de la operación e indica el número de columnas que procesarán algunos procesadores, mientas que otros procesarán ese número entero mas uno. Cuando $n - c_j < q$, entonces cada procesador encontrará una columna de la matriz **Φ**.

3.5. Diferenciación Numérica con Matriz Φ Constante

En la Sección 3.3 se explicó la manera de reducir de forma significativa el cálculo de la matriz **Φ**. En esta sección se propone calcular solamente una vez la matriz **Φ** y usarla para el cálculo de **C** y durante el resto del proceso la matriz **C** permanece constante. Cuando la matriz

C permanece constante, la inversa de la matriz, necesaria en el proceso del método de DN en sus diferentes versiones se calcula solamente una vez.

En la Figura 3.4 se observa el algoritmo DN usando $\mathbf{\Phi}$ constante (DNJ), en ella se puede observar cuatro bloques importantes que son:

- Bloque 1. Calcula los periodos iniciales de integración y el Ciclo Base, además revisa si es necesario calcular $\mathbf{\Phi}$. En este bloque obtienen los vectores $\mathbf{x}_i$ y $\mathbf{x}_{i+1}$ y el *Error*.
- Bloque 2. Cálculo de $\mathbf{\Phi}$. Este bloque toma los vectores $\mathbf{x}_i$ y $\mathbf{x}_{i+1}$ y calcula $\mathbf{\Phi}$.
- Bloque 3. Cálculo de la matriz **C**. Este bloque evalúa $\mathbf{C} = (\mathbf{I} - \mathbf{\Phi})^{-1}$.
- Bloque 4. Cálculo de los ciclos finales. En este bloque se usa la matriz **C** en todos los ciclos necesarios para llegar a la solución. Cada ciclo debe de evaluar $\mathbf{x}_\infty = \mathbf{x}_i + \mathbf{C}(\mathbf{x}_{i+1} - \mathbf{x}_i)$, después de esa evaluación se realiza la operación $\mathbf{x}_i = \mathbf{x}_\infty$, posteriormente se integra un periodo de tiempo para obtener el nuevo vector $\mathbf{x}_{i+1}$. Con los vectores $\mathbf{x}_i$ y $\mathbf{x}_{i+1}$ se obtiene el *Error*; cuando el *Error* es mayor que un criterio de convergencia establecido se repite el proceso indicado en el bloque 4 y de lo contrario termina el proceso con la solución en x_{i+1}.

3.5.1. Propuesta de Complejidad Computacional del Algoritmo de Diferenciación Numérica con Φ Constante

Para analizar la complejidad computacional del procedimiento de DN usando la matriz $\mathbf{\Phi}$ constante y con ello revisar el desempeño de esta métodología, es necesario analizar el Algoritmo 16.

Algoritmo 14 Algoritmo para evaluar la Ecuación $\mathbf{C} = (\mathbf{I} - \mathbf{\Phi})^{-1}$

```
1: function OPERACION-RESTA-INVERSA(Φ, I, n)
2:     B = RESTA(I, Φ)
3:     C = INVERSA(B, n)
4:     return C
```

Los tiempos en cada una de las líneas del Algoritmo 16 donde es necesario operaciones de punto flotante son:

- En las primeras líneas (hasta la 7) los tiempos son despreciables.

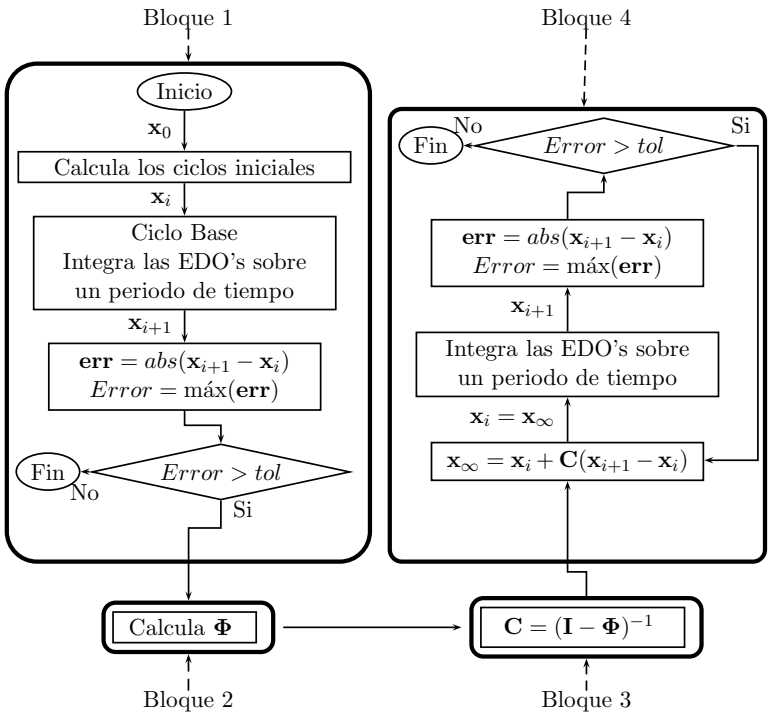


Figura 3.4: Método DN con **Φ** constante.

Algoritmo 15 Algoritmo para evaluar la Ecuación $\mathbf{x}_\infty = \mathbf{x}_i + \mathbf{C}\,(\mathbf{x}_{i+1} - \mathbf{x}_i)$

```
1: function CICLO-LIMITE-Φ-CONSTANTE(C, n, x, x1)
2:     for r = 1 : n do
3:         suma = 0
4:         for c = 1 : n do
5:             suma = suma + C[r][c](x1[c] − x[c])
6:         x∞[r] = x[r] + suma
7:     return x∞
```

Algoritmo 16 Método de Diferenciación Numérica usando $\mathbf{\Phi}$ constante

```
1:  function DIFERENCIACIÓN-NUMÉRICA-Φ-CONSTANTE(f, x0)
2:      n=número-de-variables-de-estado
3:      T = 1/60
4:      m = 512
5:      h = T/m
6:      t = 0
7:      periodos = 8
8:      x = CICLOS-INICIALES(f, x0, n, h, t, m, periodos, T)
9:      x1 = INTEGRA-PERIODO(f, x, n, h, t, m)
10:     error = ERROR(x, x1, n)
11:     if error > tolerancia then
12:         Φ = MATRIZ-IDENTIFICACIÓN(f, x, x1, n, h, t, m)
13:         I = UNITARIA(n)
14:         C = OPERACION-RESTA-INVERSA(Φ, I, n)
15:         do
16:             x = CICLO-LIMITE-Φ-CONSTANTE(C, n, x, x1)
17:             x1 = INTEGRA-PERIODO(f, x, n, h, t, m)
18:             error = ERROR(x, x1, n)
19:         while error > tolerancia
20:     return x1
```

- En la línea 8 es necesario integrar b periodos de tiempo, por lo tanto, es necesario un tiempo $t_b = bt_p = bmn(4t_1 + 3t_2 + t_3)$.

- La línea 9 integra un periodo de tiempo, por lo tanto, es necesario un tiempo t_p o $t_p = mn(4t_1 + 3t_2 + t_3)$.

- Las líneas 10 y 18 ejecutan la función para calcular el *error* máximo que se presenta en la variable de estado con mayor error. El tiempo de ejecución es nt_2; es muy pequeño comparado en tiempo necesario para integrar un periodo de tiempo que requiere un término de $3nmt_2$ (poco mas de 1500 veces mas grande, solamente en ese término).

- La línea 12 obtiene $\mathbf{\Phi}$ y el tiempo de ejecución en forma secuencial será $t_{ms} = n^2m(4t_1 + 3t_2 + t_3)$ o en función del tiempo para ejecutar un periodo, $t_{ds} = nt_p$.

- La línea 13 define una matriz unitaria, el tiempo de ejecución será despreciable comparado con la inversa de la matriz que se ocupa en la línea siguiente y los demás procesos.

- La línea 14 procesa una resta de dos matrices y la inversa del resultado de la resta de esas

matrices. El tiempo de ejecución es $\frac{4}{3}n^3t_2$ (ver información en Sección 2.3.6).

- La línea 16 requiere un tiempo $(n^2+n)t_2$. El tiempo n^2t_2 se obtiene de la línea 5 del Algoritmo 15 y el tiempo nt_2 se obtiene de la línea 6 del mismo algoritmo.
- La línea 17 integra un periodo de tiempo y requiere un tiempo $tp = mn(4t_1 + 3t_2 + t_3)$.
- Las líneas 16, 17 y 18 se ejecutan de forma repetida mientras el *error* sea mayor a la *tolerancia* establecida. Si se compara el tiempo $(n^2+n)t_2$ con el tiempo requerido por la inversa, entonces el tiempo $(n^2 + n)t_2$ se puede despreciar; similarmente se desprecia el tiempo nt_2 de la línea 18. El tiempo para ejecutar los d ciclos finales es $t_d = dt_p = dmn(4t_1 + 3t_2 + t_3)$.

El tiempo para realizar el Algoritmo 16 se obtiene de acuerdo a la Ecuación (3.24). Agrupando adecuadamente términos se obtiene la Ecuación (3.25).

$$t_{dnj} = bt_p + t_p + nt_p + \frac{4}{3}n^3t_2 + dt_p = t_p(b + 1 + n + d) + \frac{4}{3}n^3t_2 \tag{3.24}$$

$$t_{dnj} = (b + 1)t_p + nt_p + \frac{4}{3}n^3t_2 + dt_p \tag{3.25}$$

Donde,

- t_p es el tiempo para integrar un periodo.
- $\frac{4}{3}n^3t_2$ es el tiempo necesario para obtener la inversa de una matriz de orden $n \times n$; es el tiempo de ejecución del bloque 3 de la Figura 3.4.
- b es el número de periodos iniciales de integración.
- n es el número de variables de estado del sistema.
- d es el número de periodos de integración que se requieren después de calcular $\mathbf{C}$.
- $(b + 1)t_p$ es el tiempo de proceso del bloque 1 de la Figura 3.4.
- nt_p es el tiempo necesario para obtener $\mathbf{\Phi}$ (bloque 2 de la Figura 3.4).
- dt_p es el tiempo de proceso de los ciclos finales del Algoritmo 16; es el tiempo para procesar el bloque 4 de la Figura 3.4.

Sustituyendo el valor de $t_p = mn\,(4t_1 + 3t_2 + t_3)$ en la Ecuación (3.25) se obtiene,

$$t_{dnj} = (b + 1 + d)mn\,(4t_1 + 3t_2 + t_3) + n^2 m\,(4t_1 + 3t_2 + t_3) + \frac{4}{3}n^3 t_2 \tag{3.26}$$

Donde,

- $(b + 1 + d)mn\,(4t_1 + 3t_2 + t_3)$ es el tiempo con el que se calcula los ciclos iniciales, el ciclo base, y los ciclos finales del Algoritmo 16. La complejidad computacional es lineal en esta parte del Algoritmo.

- $n^2 m\,(4t_1 + 3t_2 + t_3)$ es el tiempo en el que se obtiene $\mathbf{\Phi}$. La complejidad computacional es cuadrática.

- $\frac{4}{3}n^3 t_2$ es el tiempo para calcular la inversa. La complejidad es cúbica.

La Ecuación (3.26) (que describe el tiempo para obtener el estado estacionario periódico usando $\mathbf{\Phi}$ constante) se puede comparar con la Ecuación (2.27) (que describe el tiempo para obtener el estado estacionario periódico usando el método DN). La Ecuación (2.27) se escribe a continuación para facilitar la lectura.

$$t_{dn} = (b + 1 + p)\,mn\,(4t_1 + 3t_2 + t_3) + p\left(n^2 m\,(4t_1 + 3t_2 + t_3) + \frac{4n^3}{3}t_2\right)$$

La variable p desaparece del Algoritmo 2.27; la variable p indica las veces que se tiene que calcular la matriz $\mathbf{\Phi}$ y la inversa de una matriz de orden $n \times n$. La matriz $\mathbf{\Phi}$ tiene una complejidad computacional cuadrática, la inversa tiene una complejidad computacional cúbica y el resto del proceso una complejidad lineal. Por lo tanto, si p afecta principalmente los elementos de mayor complejidad, entonces se puede asegurar que al eliminar p se tiene una reducción cercana a p veces al efecuar el algoritmo de DNJ.

La variable d aparece en la Ecuación (3.26) y es el número de periodos finales de integración, necesarios para llegar a la solución.

3.5.2. Diferenciación Numérica con Matriz Φ Constante y Procesamiento Paralelo en el Cálculo de Φ

El cálculo de la matriz **Φ** es el proceso más importante en el método de DN y sus variantes hasta un número (n) finito de variables de estado. Además el proceso para el cálculo de las columnas de la matriz **Φ** tienen la característica de que se pueden calcular de forma independiente unas de otras, por lo tanto, se puede ejecutar usando procesamiento en paralelo para reducir aún más el tiempo de solución que ya se redujo al evitar calcular la matriz **Φ**.

En la Sección 3.2 se obtuvo la Ecuación (3.4) para el cálculo de **Φ** usando n elementos de proceso (un elemento de proceso calcula una columna de la matriz **Φ**). Al cambiar el tiempo de procesamiento de **Φ** de secuencial a paralelo en el proceso para obtener la solución en estado estacionario periódico, se obtiene:

$$t_{dnpj} = (b+1+d)mn\,(4t_1+3t_2+t_3) + nm\,(4t_1+3t_2+t_3) + \frac{4}{3}n^3t_2 \tag{3.27}$$

En la Ecuación (3.27) se observa que el término cuadrático (generado por el cálculo de **Φ**) desaparece del tiempo total para determinar el estado estacionario periódico. Cuando se tienen n elementos de proceso para el cálculo de la matriz de identificación, éste tiempo estará dado por la Ecuación (3.28). El tiempo necesario para calcular la inversa tomará mayor importancia para sistemas que hasta antes del procesamiento en paralelo parecía despreciable.

$$t_{dnpj} = (b+2+d)mn\,(4t_1+3t_2+t_3) + \frac{4}{3}n^3t_2 \tag{3.28}$$

En la Sección 3.2 se obtuvo también la Ecuación (3.8) para calcular **Φ** con q elementos de proceso, donde $q < n$. Al sustituir el tiempo de la Ecuación (3.8) en la Ecuación (3.26) de manera repetitiva durante el proceso iterativo se obtiene:

$$t_{dnpj} = (b+1+d)mn\,(4t_1+3t_2+t_3) + \left(\frac{n}{q}+1\right)nm\,(4t_1+3t_2+t_3) + \frac{4}{3}n^3t_2 \tag{3.29}$$

La Ecuación (3.29) se puede escribir en función del tiempo t_p que se requiere para integrar un periodo de tiempo como:

$$t_{dnpj} = (b + 1 + d)\, t_p + \left(\frac{n}{q} + 1\right) t_p + \frac{4n^3}{3} t_2 \tag{3.30}$$

3.6. Proceso en Paralelo del Método de Runge Kutta de Cuarto Orden

El método de DN selectivo redujo de forma importante los cálculos de las columnas de las matriz $\mathbf{\Phi}$. Cuando se usa procesamiento en paralelo en las diferentes versiones del método de DN se reduce de forma significativa el tiempo para obtener $\mathbf{\Phi}$ y con ello el tiempo para la solución del estado estacionario periódico (no se reducen los cálculos; se distribuyen entre los diferentes elementos de proceso). En el método de DN con la matriz $\mathbf{\Phi}$ constante, se reduce el esfuerzo computacional al evitar el cálculo de los dos elementos más importantes (la matriz $\mathbf{\Phi}$ y la inversa de una matriz) que contribuyen en el tiempo de solución del estado estacionario periódico.

El resumen de las ecuaciones obtenidas del método de DN y sus variantes se muestra en la Tabla 3.1.

Tabla 3.1: Ecuación del método de DN y sus variantes

Nombre del Método	Ecuación
DN	$t_{dn} = (b + 1 + p)\, t_p + p\left(n t_p + \frac{4n^3}{3} t_2\right)$
DN ($\mathbf{\Phi}$ con n elementos de proceso)	$t_{dnp} = (b + 1 + p)\, t_p + p\left(t_p + \frac{4n^3}{3} t_2\right)$
DN ($\mathbf{\Phi}$ con q elementos de proceso)	$t_{dnp} = (b + 1 + p)\, t_p + p\left(\left(\frac{n}{q} + 1\right) t_p + \frac{4n^3}{3} t_2\right)$
DN selectiva	$t_{dns} = (b + 1 + p)\, t_p + \sum_{j=1}^{p} (n - k_j)\, t_p + p\frac{4n^3}{3} t_2$
DN selectiva ($\mathbf{\Phi}$ con q elem. proceso)	$t_{dns} = (b + 1 + p)\, t_p + \sum_{j=1}^{p} \left(\frac{n-k_j}{q} + 1\right) t_p + p\frac{4n^3}{3} t_2$
DN ($\mathbf{\Phi}$ constante)	$t_{dnj} = t_p(b + 1 + n + d) + \frac{4}{3} n^3 t_2$
DN ($\mathbf{\Phi}$ constante y n elem. proceso)	$t_{dnpj} = (b + 2 + d) t_p + \frac{4}{3} n^3 t_2$
DN ($\mathbf{\Phi}$ constante y q elem. proceso)	$t_{dnpj} = (b + 1 + d)\, t_p + \left(\frac{n}{q} + 1\right) t_p + \frac{4n^3}{3} t_2$

En todos los casos, el esfuerzo para reducir el tiempo de solución se concentra en la reducción del tiempo para obtener $\mathbf{\Phi}$. En esta sección se propone reducir el tiempo de solución del método de integración (en esta tesis se utilizó el método de Runge Kutta de cuarto orden).

Todas las ecuaciones están escritas en función del tiempo del periodo de integración ($t_p = mn(4t_1+3t_2+t_3)$). Para reducir el tiempo de solución de cualquier versión del método DN se puede reducir el tiempo de solución del periodo de integración. Para esto es necesario reducir una de las variables de la Ecuación $t_p = mn(4t_1 + 3t_2 + t_3)$. Las observaciones son las siguientes:

- m: si se reduce el valor de la variable m, se incrementa el paso de integración y al aumentar el paso de integración, el método de RK4 puede no converger a una solución.

- n: n es el número de variables de estado y de EDO's en el modelo del sistema. Las evaluaciones (necesarias en el método de RK4) de las EDO's se pueden procesar en paralelo, así como algunas operaciones parciales. Con el procesamiento en paralelo del método de RK4 se puede reducir el término n dentro de la ecuación para calcular un periodo de integración.

- $4t_1 + 3t_2 + t_3$: los términos que aparecen en la ecuación $4t_1 + 3t_2 + t_3$ son propios del método de RK4, donde t_1 es el tiempo necesario para evaluar una EDO; t_2 es el tiempo para evaluar un término de dos operaciones y t_3 es el tiempo para evaluar un término de 7 operaciones. El coeficiente 4 de la variable t_1 indica que cada ecuación del conjunto de EDO's se va a evaluar 4 veces; el coeficiente 3 de la variable t_2 indica que será necesario evaluar 3 veces una expresión de dos operaciones y el coeficiente 1 de la variable t_3 indica que será necesario evaluar una vez el término que ocupa 7 operaciones. Los coeficientes de los términos anteriores no se pueden modificar, ya que son propios del método de RK4 (la única forma de modificar los coeficientes es cambiando el método de integración). La otra forma de reducir el tiempo de solución del método RK4 es reducir el tiempo t_1, t_2 y t_3, para ello será necesario que las evaluaciones de las operaciones se realicen de forma eficiente (esta parte depende del dispositivo y de las estructuras que se utilicen en las evaluaciones).

En la Figura 3.5 se muestra el algoritmo de RK4 ejecutado de forma paralela, en ella se observa el proceso que sigue cada hilo (elemento de proceso). Los hilos tendrán que sincronizarse en cada uno de los puntos indicados de sincronización. El primer punto de sincronización está después de calcular el vector $\mathbf{y}_1$; este vector es necesario para el cálculo del $\mathbf{k}_2$. Los siguientes puntos de sincronización son necesarios debido a que después de la sincronización se requerirá la información calculada previamente.

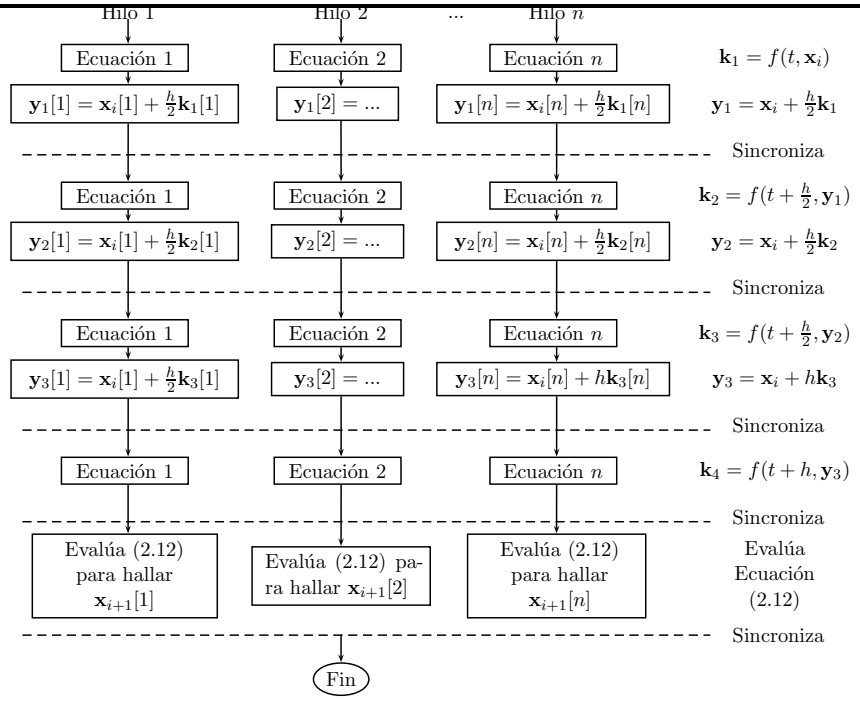


Figura 3.5: Método de Runge Kutta de 4to. orden procesado en paralelo.

3.6.1. Propuesta de Complejidad del Algoritmo de Runge Kutta de Cuarto Orden en Paralelo

El tiempo de solución del método RK4 procesado secuencialmente se describió en la Sección 2.3.2.

Para determinar la complejidad del método de RK4 en paralelo se analizará el Algoritmo 17.

- En la línea 2 se indica el número j elemento de proceso que evaluará las siguientes líneas del algoritmo.
- Las líneas 3, 6, 9 y 12 evalúan una ecuación del conjunto de EDO's y la evaluación de una ecuación requiere de un tiempo t_1. La evaluación de 4 ecuaciones requiere de un tiempo $4t_1$.
- Las líneas 4, 7 y 10 requieren evaluar un término de dos operaciones (una multiplicación y una suma). El tiempo de ejecución de cada línea es t_2.
- La línea 14 requiere de un tiempo t_3 y evalúa un término con siete operaciones (4 sumas y 3

multiplicaciones).

- En las líneas 5, 8, 11, 13 y 15 se realiza una sincronización de todos los elementos de proceso que se utilizan en el método RK4.

Para el análisis del tiempo de solución del método RK4 en paralelo, el tiempo de sincronización se puede despreciar, ya que cada elemento de proceso necesita el mismo tiempo para realizar su tarea. No sería necesario esperar al elemento de proceso con mayor carga computacional debido a que los n elementos de proceso ocuparían el mismo tiempo. La realidad es que no todos los elementos de proceso realizan el mismo trabajo debido a que el tiempo para ejecutar las ecuaciones no es el mismo. Si no se toma el tiempo de la sincronización, el tiempo para ejecutar el método RK4 en paralelo es,

$$t_{rkp} = 4t_1 + 3t_2 + t_3 \tag{3.31}$$

Considerando el tiempo t_4 de sincronización, el tiempo para ejecutar el método de RK4 es,

$$t_{rkp} = 4t_1 + 3t_2 + t_3 + 5t_4 \tag{3.32}$$

Algoritmo 17 Método de Runge Kutta de cuarto orden en paralelo

1: **function** RUNGEK4-PARALELO($\mathbf{f}, \mathbf{x}, n, h, t$)
2: j=elemento de proceso (hilo)
3: $\mathbf{k}_1[j] = \text{EVALÚA}(\mathbf{f}[j], t, \mathbf{x})$
4: $\mathbf{y}_1[j] = \mathbf{x}[j] + \frac{h}{2}\mathbf{k}_1[j]$
5: Sincroniza
6: $\mathbf{k}_2[j] = \text{EVALÚA}(\mathbf{f}[j], t + \frac{h}{2}, \mathbf{y}_1))$
7: $\mathbf{y}_2[j] = \mathbf{x}[j] + \frac{h}{2}\mathbf{k}_2[j]$
8: Sincroniza
9: $\mathbf{k}_3[j] = \text{EVALÚA}(\mathbf{f}[j], t + \frac{h}{2}, \mathbf{y}_2))$
10: $\mathbf{y}_3[j] = \mathbf{x}[j] + h\mathbf{k}_3[j]$
11: Sincroniza
12: $\mathbf{k}_4[j] = \text{EVALÚA}(\mathbf{f}[j], t + h, \mathbf{y}_3))$
13: Sincroniza
14: $\mathbf{y}[j] = \mathbf{x}[j] + \frac{h}{6}\left(\mathbf{k}_1[j] + 2\mathbf{k}_2[j] + 2\mathbf{k}_3[j] + \mathbf{k}_4[j]\right)$
15: Sincroniza
16: **return y**

Para integrar un periodo de tiempo se requiere ejecutar el m veces el método RK4, por lo tanto, el tiempo t_{pp} para integrar un periodo es el que se muestra en la Ecuación (3.33). Si se compara el tiempo de la Ecuación (3.33) con el tiempo de la Ecuación (2.16) que es $t_p = mn(4t_1 + 3t_2 + t_3)$ se observa que desaparece la variable n de la ecuación. Por lo tanto, se puede considerar que el tiempo de ejecución para la integración de un periodo es constante; es decir,

$$t_{pp} = m\,(4t_1 + 3t_2 + t_3) \tag{3.33}$$

Para que el tiempo de ejecución del método de RK4 en paralelo sea como se indica en la Ecuación (3.33) se requieren n elementos de proceso. Cuando solamente se tienen r ($r < n$) elementos de proceso disponibles, entonces el tiempo para ejecutar el método RK4 será el dado por la Ecuación (3.34) y el tiempo para ejecutar un periodo de integración será el dado por la Ecuación (3.35).

$$trkp = \left(\frac{n}{r} + 1\right)(4t_1 + 3t_2 + t_3) \tag{3.34}$$

$$tpp = m\left(\frac{n}{r} + 1\right)(4t_1 + 3t_2 + t_3) \tag{3.35}$$

Para integrar un periodo de tiempo de manera secuencial se requiere un tiempo t_p y para integrar un periodo de tiempo de en paralelo se requiere de un tiempo t_{pp}. Se puede cambiar el tiempo t_p de todas las ecuaciones de la Tabla 3.1 para obtener las ecuaciones mostradas en la Tabla 3.2

Cuando se utiliza procesamiento en paralelo para integrar los periodos de tiempo no solamente se reduce el tiempo en $\mathbf{\Phi}$, también se reduce el tiempo en los ciclos iniciales y el ciclo base (en los métodos de DN, DN con matriz $\mathbf{\Phi}$ selectiva y DN con matriz $\mathbf{\Phi}$ constante) y en los ciclos finales (del método de DN con matriz $\mathbf{\Phi}$ constante).

Como el tiempo para integrar un periodo (T) con n elementos de proceso es t_{pp} (ver Ecuación (3.35)); el tiempo para ejecutar los ciclos iniciales con b periodos de integración será el que se muestra en la Ecuación (3.36). La complejidad de los ciclos iniciales es constante, ya que desaparece de la ecuación la variable n.

Tabla 3.2: Ecuación del método de DN y sus variantes

Nombre del Método	Ecuación
DN	$t_{dn} = (b+1+p)\,t_{pp} + p\left(nt_{pp} + \frac{4n^3}{3}t_2\right)$
DN ($\mathbf{\Phi}$ con n elementos de proceso)	$t_{dnp} = (b+1+p)\,t_{pp} + p\left(t_{pp} + \frac{4n^3}{3}t_2\right)$
DN ($\mathbf{\Phi}$ con q elementos de proceso)	$t_{dnp} = (b+1+p)\,t_{pp} + p\left(\left(\frac{n}{q}+1\right)t_{pp} + \frac{4n^3}{3}t_2\right)$
DN selectiva	$t_{dns} = (b+1+p)\,t_{pp} + \sum_{j=1}^{p}(n-k_j)\,t_{pp} + p\frac{4n^3}{3}t_2$
DN selectiva ($\mathbf{\Phi}$ con q elem. proceso)	$t_{dns} = (b+1+p)\,t_{pp} + \sum_{j=1}^{p}\left(\frac{n-k_j}{q}+1\right)t_{pp} + p\frac{4n^3}{3}t_2$
DN ($\mathbf{\Phi}$ constante)	$t_{dnj} = t_{pp}(b+1+n+d) + \frac{4}{3}n^3t_2$
DN ($\mathbf{\Phi}$ constante y n elem. proceso)	$t_{dnpj} = (b+2+d)t_{pp} + \frac{4}{3}n^3t_2$
DN ($\mathbf{\Phi}$ constante y q elem. proceso)	$t_{dnpj} = (b+1+d)\,t_{pp} + \left(\frac{n}{q}+1\right)t_{pp} + \frac{4n^3}{3}t_2$

$$t_{bp} = bm\,(4t_1 + 3t_2 + t_3) \tag{3.36}$$

El tiempo para obtener $\mathbf{\Phi}$ se beneficia del cálculo en paralelo del método de RK4, ya que en el cálculo se puede llevar doble procesamiento en paralelo (uno para el cálculo de las columnas en paralelo y otro en cada columna, donde se procesa en paralelo el método de RK4). En la Ecuación (3.4) el tiempo para calcular $\mathbf{\Phi}$ en paralelo es $t_{mp} = mn(4t_1+3t_2+t_3)$, donde $mn(4t_1+3t_2+t_3)$ es el tiempo para integrar un periodo (T), siendo este tiempo t_p, por lo tanto, el tiempo para obtener $\mathbf{\Phi}$ en paralelo es $t_{mp} = t_p$. Si se utiliza el método de RK4 para integrar un periodo de tiempo, entonces el tiempo para calcular $\mathbf{\Phi}$ es $t_{mp} = t_{pp}$; es decir,

$$t_{mp} = m\,(4t_1 + 3t_2 + yt_3) \tag{3.37}$$

La complejidad computacional del cálculo $\mathbf{\Phi}$ de ser CUADRÁTICA se convierte en LINEAL con el procesamiento en paralelo de sus columnas y CONSTANTE con el procesamiento en paralelo de las columnas y procesamiento en paralelo del método de integración. La dificultad técnica que se tendrá para obtener resultados como lo indica la Ecuación (3.37) es que se requieren n^2 elementos de proceso para el cálculo de $\mathbf{\Phi}$.

Al utilizar procesamiento paralelo en el cálculo de RK4 para la integración de los periodos

de tiempo se obtiene,

$$t_{dnpj} = (b + 2 + d)m\,(4t_1 + 3t_2 + t_3) + \frac{4}{3}n^3 t_2 \tag{3.38}$$

Donde,

- $m\,(4t_1 + 3t_2 + t_3)$ es el tiempo para integrar un periodo y también es el tiempo para obtener $\mathbf{\Phi}$,
- $\frac{4}{3}n^3 t_2$ es el tiempo para obtener la inversa de la matriz

Despreciando el tiempo de sincronización en el metodo RK4, la reducción en el tiempo al ejecutar el método RK4 en paralelo es n veces el tiempo para obtener $\mathbf{\Phi}$ y el los ciclos iniciales de integración. Cuando se procesan en paralelo las columnas de $\mathbf{\Phi}$, se reduce n veces el tiempo para obtener la matriz $\mathbf{\Phi}$. Por lo tanto, el efecto de aplicar procesamiento en paralelo al método RK4 y al cálculo de las columnas de $\mathbf{\Phi}$ es una reducción de n^2 veces el tiempo para determinar $\mathbf{\Phi}$ y n veces el tiempo para calcular los otros periodos de integración (en los ciclos iniciales, ciclo base y ciclos finales).

3.7. Propuesta para el Cálculo de la Inversa de la Matriz Usando Factorización LU y Procesamiento en Paralelo

En la Sección 2.3.6 se analizó la inversa de la matriz usando la factorización LU y un proceso para la sustitución hacia adelante y hacia atrás. La inversa se requiere en el método de DN y sus variantes.

Para sistemas con pocas variables de estado (menos de 500), el tiempo para calcular la inversa de una matriz no es tan importante comparado con el tiempo para procesar la matriz $\mathbf{\Phi}$ de forma secuencial y también el resto del proceso. El problema se presenta cuando se trabaja con sistemas de más variables de estado o se utiliza procesamiento en paralelo en el resto del proceso.

En la Ecuación (3.38) se observa que la variable n se presenta en el término involucrado en el cálculo de la inversa y además n es cúbica. En sistemas de mediana y gran escala, el tiempo asociado con el cálculo de la inversa será el más importante dentro del tiempo total para obtener la solución del estado estacionario periódico.

En esta sección se presenta un procedimiento para reducir el tiempo de solución de la inversa de la matriz y va a contribuir a la reducción del tiempo del método DN en sus diferentes versiones.

En la Sección 2.3.6 se analizó la complejidad computacional de la factorización LU; está dada por la Ecuación (2.19) ($\frac{1}{3}n^3$) y la complejidad computacional del procedimiento de sustitución hacia adelante y hacia atrás para obtener la inversa de la matriz. La complejidad del procedimiento de sustitución hacia adelante y hacia atrás está determinada por la Ecuación (2.20) (n^3).

En el Algoritmo 18 se muestra el procedimiento para la sustitución hacia adelante y hacia atrás. La sustitución hacia adelante y hacia atrás se debe de repetir n veces (una vez para obtener una columna). Las columnas de la inversa se pueden calcular en forma independiente; es decir, si se usan n elementos de proceso, entonces se pueden calcular las n columnas de forma simultánea. El Algoritmo 18 lo ejecuta un elemento de proceso, donde en la línea 2 se indica el número del elemento de proceso que ejecutará el código.

Algoritmo 18 Sustitución hacia adelante y hacia atrás en paralelo

```
1:  function SUSTITUCIÓN-PARALELO(A, I, n)
2:      c=número de elemento de proceso
3:      for k = 1 : n do
4:          b[k] = I[k][c]
5:      for i = 2 : n do
6:          suma = b[i]
7:          for j = 1 : i − 1 do
8:              suma = suma − A[i][j] ∗ b[j]
9:          b[i] = suma
10:     X[n][c] = b[n]/A[n][n]
11:     for i = n − 1 : 1 do
12:         suma = 0
13:         for j = i + 1 : n do
14:             suma = suma + A[i][j] ∗ X[j][c]
15:         X[i][c] = (b[i] − suma)/A[i][i]
16:     sincroniza
17:     return X
```

A continuación se indica el número de veces que se ejecutan las operaciones de punto flotante en el Algoritmo 18:

- La línea 8 se ejecuta $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$ veces

- La línea 10 se ejecuta 1 vez
- La línea 14 se ejecuta $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$ veces
- La línea 15 se ejecuta $n-1$ veces

Sumando todas las operaciones se tiene $(n-1) + 2\frac{n(n-1)}{2} + 1 = n^2$.

La complejidad del Algoritmo para el proceso de sustitución hacia adelante y hacia atrás usando procesamiento paralelo es,

$$t_{sa} = n^2 \tag{3.39}$$

Algoritmo 19 Inversa de una matriz usando la factorización LU y el procedimiento de sustitución hacia adelante y hacia atrás en paralelo

```
1:  function UNITARIA(n)
2:      for r = 1 : n do
3:          for c = 1 : n do
4:              I[r][c] = 0
5:          I[r][r] = 1
6:      return I
7:  function INVERSA(A, n)
8:      I=UNITARIA(n)
9:      A=LU(A, n)
10:     X =<<< n >>>SUSTITUCION-PARALELO(A, I, n)
11:     return X
```

En el Algoritmo 19 se presenta el procedimiento para el cálculo de la inversa de una matriz de orden $n \times n$. La línea 9 se ejecuta en un tiempo $\frac{1}{3}n^3$ y la línea 10 en un tiempo n^2, por lo que el tiempo para ejecutar el Algoritmo 19 es,

$$t_{inv} = \frac{1}{3}n^3 + n^2 \tag{3.40}$$

Con la propuesta que se presenta en esta tesis, la inversa de la matriz sigue teniendo una complejidad cúbica pero, el coeficiente de la parte cúbica es $\frac{1}{3}$ en lugar de $\frac{4}{3}$. Con ese cambio se

puede reducir hasta en 4 veces el tiempo de solución de la inversa, lo que dará como resultado un mejor tiempo al proceso de solución.

3.8. Plataforma para el Procesamiento en Paralelo Basado en Unidades de Procesamiento Gráfico

En esta sección se analizan algunos elementos del modelo de programación CUDA introducido en [CUDA,2015] por la compañía Nvidia para la programación de sus GPUs.

En la Figura 1.5 se presenta el modelo de programación de CUDA. Un programa puede tener procesos que se pueden calcular de forma secuencial y procesos que se ejecutan en paralelo. El proceso secuencial se puede ejecutar en una CPU debido a que la frecuencia a la que trabaja la CPU es mayor a la frecuencia a la trabajan los núcleos CUDA en una GPU. Al proceso en la CPU se le puede llamar Host y al proceso paralelo de puede ejecutar en una GPU se le puede llamar Device.

En CUDA se definen funciones que serán ejecutadas en la CPU y en la GPU. La funciones que son ejecutadas en la GPU toman el nombre del kernel. Cada kernel se ejecuta idealmente de forma simultánea por los diferentes hilos de la GPU. En la Figura 1.5 se observa que la ejecución de un programa puede iniciar con un proceso secuencial, luego sigue un proceso paralelo en la GPU al ejecutar el Kernel 1 en diferentes bloques de hilos. Una vez que termina la ejecución del kernel 1 en la GPU se procede a ejecutar un proceso secuencial en la CPU. El proceso sigue alternando una parte secuencial y una parte paralela hasta que termina la aplicación.

Los bloques de hilos en la GPU se pueden dividir en una rejilla (Grid) de bloques; que puede ser de una, dos o tres dimensiones. En la rejilla 2 de la Figura 1.5 hay 6 bloques; distribuidos en dos dimensiones (dos renglones de 3 bloques cada uno).

Los hilos en un bloque también pueden estar distribuidos en una, dos o tres dimensiones. En la Figura 1.5 se muestra que el bloque 2 ejecuta 6 hilos distribuidos en dos dimensiones (tres renglones y dos columnas).

Jerarquía de Memoria

En la GPU existen tres clases de memoria que son:

- Memoria local: este tipo de memoria solamente la puede usar un hilo. Los otros hilos no pueden hacer cambios en la memoria local al hilo.
- Memoria compartida (shared): este tipo de memoria la pueden usar todos los hilos del bloque. Los hilos de otros bloques no tienen acceso a esta memoria.
- Memoria global: este tipo de memoria la pueden usar todos los hilos de todos los bloques.

En la Figura 1.5 se observa de forma clara que el hilo puede ocupar una memoria local. En un bloque de hilos es necesario usar la memoria compartida entre los núcleos CUDA del mismo bloque y la memoria global se utiliza para compartir la información entre los diferentes hilos de los diferentes bloques.

3.8.1. Propuesta de Arquitectura de Procesamiento en Paralelo Basado GPUs

En la última década la tecnología de GPUs se ha desarrollado de manera importante; Nvidia ha introducido 5 arquitecturas. Ello reduce la posibilidad de mantener un equipo configurado con las mismas GPUs, sin embargo, se puede mantener la configuración del equipo cambiando las GPUs por unas más actuales. La configuración dependerá del soporte que pueda tener el sistema operativo con las nuevas GPUs y de las disposiciones físicas (conectores, espacio, potencia suficiente en la fuente, etc.).

Uno de los factores para tomar la decisión de qué equipo funciona para una aplicación particular es el recurso económico. La parte técnica y a veces científica es sacrificada por el recurso limitado que se tiene para tomar las mejores decisiones. En esta tesis se configuró un equipo que hoy puede ser cuestionado por la tecnología que ha surgido en los últimos 4 años. En este tiempo la compañía ha lanzado al mercado 2 nuevas arquitecturas (Kepler y Mawell). Hace 4 años, la configuración que se muestra como propuesta fue de lo mejor en el mercado de GPUs.

De la misma forma en que las GPUs las han introducido al mercado, la plataforma CUDA se ha ido desarrollando para soportar las nuevas arquitecturas. En la parte de las aplicaciones, el programador no debería de notar los cambios de la parte física (arquitectura de la GPU) y tampoco de la plataforma de programación.

En la parte de la programación en CUDA, un kernel (función que se ejecuta en una GPU) en una arquitectura Fermi debe llamarse desde la CPU para que se ejecute con varios hilos en una

GPU. En la arquitectura Kepler se puede ejecutar lo que se ejecuta en las GPUs con la arquitectura Fermi y además una GPU puede llamar a un kernel para ser ejecutado por la cantidad de hilos que se indica en la GPU. En una GPU con arquitectura Fermi un kernel-1 no puede llamar a otro kernel-2 con una cantidad de hilos que se indique en el kernel-1. La programación de las GPUs en donde un kernel llama a otro kernel con distinta cantidad de bloques e hilos se le llama programación dinámica y fue introducida en la architectura Kepler de la segunda generación GK110.

En las propuestas de mejora al método DN descritas en el Capítulo 3 se hace notar que para procesar la matriz de identificación se pueden utilizar n^2 elementos de proceso, trabajando de forma simultánea. Uno de los sistemas de prueba más pequeños que se analizaron fue el sistema de prueba modificado del IEEE de 14 nodos con 54 variables de estado; el número de elementos de proceso que se pueden utilizar para resolver este sistema es de $54 \times 54 = 2916$ elementos de proceso. Para acercarse a esa cantidad de elementos de proceso basado en CPUs se requiere de una infraestructura muy costosa.

En esta aplicación, las GPUs son los dispositivos que pueden acercarse al número de elementos de proceso que pueden usarse de forma simultánea para ejecutar el método DN.

Se identificó que las GPUs representan la mejor opción, en cuanto a dispositivos para determinar la solución del estado estacionario periódico usando el método DN. Sin embargo, se tienen algunas dificultades técnicas y económicas. Las dificultades técnicas se presentan debido a que una GPU no tiene los suficientes elementos de proceso que se pueden utilizar; por lo tanto, se puede configurar un equipo con varias GPUs para acercarse al número de elementos de proceso que se pueden utilizar. En el momento de evaluar la gama de GPUs (año 2011); la GPU tesla 2075 tenía las mejores características del mercado con 448 núcleos cuda distribuidos en 14 SM. A la fecha de la terminación de la tesis, Nvidia tiene una GPU tesla K80 con 4992 núcleos CUDA distribuidos en 26 SMX y una tarjeta gráfica GTX titan Z con 5760 núcleos CUDA.

En las GPUs con arquitectura Fermi y las GPUs con arquitectura Kepler de la primera generación es necesario un núcleo de la CPU por cada GPU instalada.

En el año 2011 había pocas computadoras para conectar varias tarjetas gráficas. Una de las computadoras en las que se podía conectar hasta 3 GPUs es la estación de trabajo DELL (Precision R5500). La computadora [Precision-R5500] se configuró con 3 GPUs.

En la Figura 3.6 se muestra la propuesta de arquitectura del equipo utilizado para eje-

cutar en paralelo el programa del método DN y sus variantes propuestas en el Capítulo 3. Las características principales de los elementos de la arquitectura son:

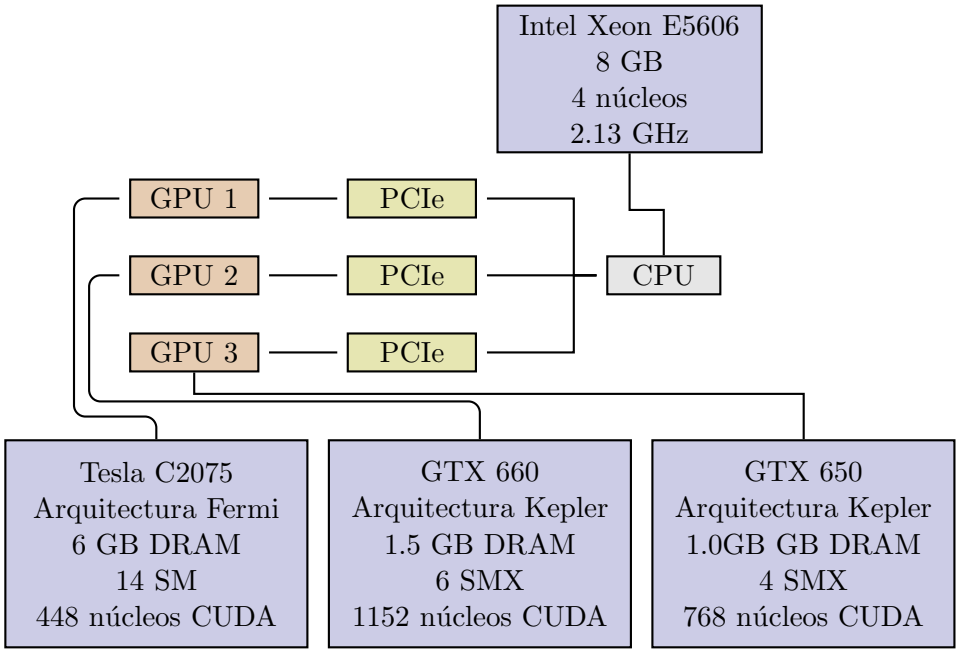


Figura 3.6: Arquitectura Propuesta

La Unidad Central de Proceso (CPU): es el procesador Intel Xeon E5606 [Intel-E5606] que cuenta con 4 núcleos y trabaja a una frecuencia de 2.13 GHz. La CPU que se eligió es suficiente para la cantidad de GPUs que se pueden colocar en la estación de trabajo Dell Precision R5500. La memoria RAM de la estación de trabajo es de 8 GB.

La GPU más importante es la Nvidia Tesla C2075 [Nvidia-tesla-c2075], debido a que tiene 14 multiprocesadores de flujo (SM); esta GPU es de la arquitectura Fermi. Por lo tanto, cada SM tiene 32 núcleos CUDA para tener un total de 448 núcleos CUDA; la frecuencia de los núcleos CUDA es de 1.15 GHz; la memoria DRAM es de 6 GB.

La GPU GTX 660 [Nvidia-GTX-660] tiene 6 multiprocesadores de flujo (SMX). Esta GPU es de arquitectura Kepler, por lo tanto, cada SMX tiene 192 núcleos CUDA para tener un total de 1152 núcleos CUDA; la frecuencia de los núcleos CUDA es de 823 MHz. La memoria DRAM de la GPU es de 1.5 GB.

La GPU GTX 650ti [Nvidia-GTX-650] tiene 4 multiprocesadores de flujo (SMX); esta GPU es de arquitectura Kepler, con 192 núcleos CUDA por SMX y un total de 768 núcleos CUDA en la GPU. Los núcleos CUDA trabajan a una frecuencia de 928 MHz. La memoria DRAM de la GPU es de 1.0 GB.

Las GPUs y la CPU se comunican a través del bus PCIe 3.0×16.

En la aplicación que se propone en esta tesis, se pueden usar n^2 elementos de proceso (el sistema eléctrico de mayor escala que se analizó fue el de 118 nodos con 479 variables de estado; donde $n = 478$. Se pueden analizar sistemas de mayor escala). No hay ningún dispositivo en este momento que tenga esa cantidad de elementos de proceso, por lo tanto, es necesario configurar un equipo con varias GPUs para tener mayores elementos de proceso disponibles.

3.8.2. Propuesta de la Plataforma de Programación

Nvidia desarrolló la plataforma de programación de CUDA para la programación de aplicaciones con GPUs. En la arquitectura Fermi y en la arquitectura Kepler de la primera generación, los kernels son llamados desde una función de la CPU para ejecutarse con varios hilos de la GPU. En la arquitectura Kepler de la segunda generación se introdujo la programación dinámica. En esta tesis se utilizaron GPUs que tienen la arquitectura Fermi y Kepler de la primera generación, por lo tanto, no se podrá usar la programación dinámica.

Cuando se tienen múltiples GPUs con arquitectura Fermi y Kepler de la primera generación y sea necesario trabajar con ellas de forma simultánea en una aplicación, es necesario tener por lo menos el mismo número de núcleos en la CPU que GPUs. Cada GPU ejecutará un Kernel que es llamado por un núcleo de la CPU.

Para hacer trabajar a todas las GPUs de forma simultánea, se tendrá que utilizar una plataforma de procesamiento en paralelo en la CPU para hacer funcionar por lo menos 3 de los núcleos de la CPU; cada uno de los tres núcleos puede activar una GPU. La CPU se encargará de procesar la parte secuencial y cada núcleo de la CPU se encargará de: llamar los kernels que se ejecutarán en la GPU que tienen activa, alimentarán a la GPU que va a procesar y recibirá los resultados del proceso que realizó la GPU.

Para el procesamiento paralelo en la parte de la CPU se utilizará la plataforma OpenMP

(ver Apéndice A) que está soportada por el lenguaje C y para el procesamiento paralelo en la parte de la GPU se utilizará la plataforma de CUDA que también esta soportada por C. Ambas plataformas pueden utilizarse en un programa de C al incluir sus librerías.

En la Figura 3.7 se observa que para el proceso en la CPU se utiliza OpenMP y para el proceso en la GPU se utiliza CUDA; ambas plataformas interactúan en un programa de C. También se observa que uno de los núcleos de la CPU no está asociado a una GPU. En algunos sistemas, la suma de los elementos de proceso que tienen las GPUs será menor a la cantidad de elementos de proceso que se pueden utilizar; los núcleos de la CPU disponibles (en este caso solamente 1) se pueden utilizar para ayudar a las GPUs a procesar la información.

El sistema fue configurado en el sistema operativo Lunix Ubuntu, aunque puede ser configurado en otros sistemas operativos.

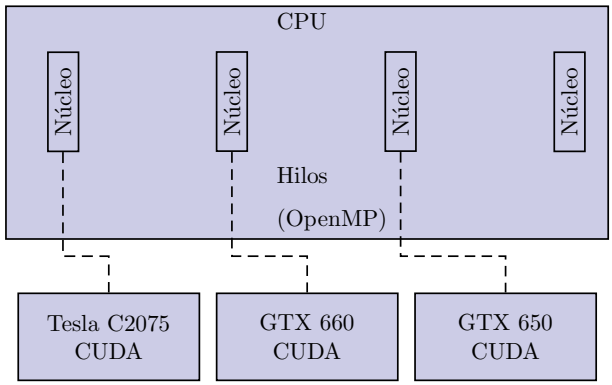


Figura 3.7: Plataforma Propuesta

3.9. Técnicas Computacionales Propuestas Basadas en Unidades de Procesamiento Gráfico

Considerando la cantidad de elementos de proceso que se pueden utilizar durante el procesamiento de $\mathbf{\Phi}$ en el método DN en sus diferentes versiones con procesamiento paralelo, se propone utilizar Unidades de Procesamiento Gráfico (GPU). Las GPU's son dispositivos que tienen muchos elementos de proceso que pueden trabajar de forma simultánea, ayudando a reducir el tiempo de la ejecución del programa (que está programado para ejecutarse de forma paralela).

En esta sección se propone utilizar GPU's al método DN y sus diferentes variantes donde se utilice procesamiento en paralelo. Las secciones donde se utilizó procesamiento paralelo fueron:

- En la Sección 3.2 se utiliza procesamiento paralelo al método DN tradicional.
- En la Sección 3.4 se utiliza procesamiento paralelo al método DN selectivo.
- En la Sección 3.5.2 se utiliza procesamiento paralelo al método DN con la matriz $\mathbf{\Phi}$ constante.
- En la Sección 3.7 se utiliza procesamiento paralelo en el cálculo de la inversa.

3.9.1. Propuesta del Método de DN en Paralelo Basada en GPU's

En la Sección 3.2 se analizó la complejidad computacional del método de DN tradicional usando procesamiento paralelo. En esta sección se usa la técnica de procesamiento en paralelo basada en GPU's para obtener la matriz $\mathbf{\Phi}$ en el método de DN tradicional. En el cálculo de los periodos de integración dentro de la matriz $\mathbf{\Phi}$ se utiliza el método de RK4 ejecutado de forma paralela (ver Sección 3.6).

En la Figura 3.8 se muestra el procedimiento del método DN tradicional procesando $\mathbf{\Phi}$ en paralelo con una GPU. En el procedimiento utilizando, una parte del método DN se ejecuta en la CPU y otra parte se ejecuta en la GPU. En la GPU se calcula la matriz $\mathbf{\Phi}$ y el resto del procedimiento se ejecuta en la CPU.

El proceso del método de DN se ha descrito en la Sección 2.2. Se hacen las siguientes consideraciones al proceso:

1. El proceso inicia calculando $\mathbf{x}_i$, $\mathbf{x}_{i+1}$ y el *error*.
2. El siguiente paso es copiar los vectores $\mathbf{x}_i$ y $\mathbf{x}_{i+1}$ de la CPU a la GPU.
3. En la GPU se procesa la matriz $\mathbf{\Phi}$ y se copia a la CPU.
4. El siguiente paso es en la CPU y es calcular las variables de estado en el Ciclo Límite (ejecutando las ecuaciones $\mathbf{C} = (\mathbf{I} - \mathbf{\Phi})^{-1}$ y $\mathbf{x}_\infty = \mathbf{x}_i + \mathbf{C}\,(\mathbf{x}_{i+1} - \mathbf{x}_i)$.
5. Finalmente, se obtienen los nuevos valores de $\mathbf{x}_i$, $\mathbf{x}_{i+1}$ y el *error*. Repitiendo el proceso desde el paso 2

Algoritmo 20 Procedimiento para calcular la matriz $\mathbf{\Phi}$ usando procesamiento paralelo con una GPU

1: **function** MATRIZ-IDENTIFICACIÓN-GPU($\mathbf{f}, \mathbf{x}, \mathbf{x1}, h, t, m, \mathbf{\Phi}$)
2: r número de bloque
3: c número de hilo del bloque r
4: $\xi = 1x10^{-6}$
5: $\mathbf{x}_j[c] = \mathbf{x}[c]$
6: Sincroniza
7: **if** $c = 1$ **then**
8: $\mathbf{x}_j[r] = \mathbf{x}_j[r] + \xi$
9: Sincroniza
10: **for** $p = 1 : m$ **do**
11: $\mathbf{k}_1[c] =$ EVALÚA($\mathbf{f}[c], t, \mathbf{x}_j$)
12: $\mathbf{y}_1[c] = \mathbf{x}_j[c] + \frac{h}{2}\mathbf{k}_1[c]$
13: Sincroniza
14: $\mathbf{k}_2[c] =$ EVALÚA($\mathbf{f}[c], t + \frac{h}{2}, \mathbf{y}_1$))
15: $\mathbf{y}_2[c] = \mathbf{x}_j[c] + \frac{h}{2}\mathbf{k}_2[c]$
16: Sincroniza
17: $\mathbf{k}_3[c] =$ EVALÚA($\mathbf{f}[c], t + \frac{h}{2}, \mathbf{y}_2$))
18: $\mathbf{y}_3[c] = \mathbf{x}_j[c] + h\mathbf{k}_3(j)$
19: Sincroniza
20: $\mathbf{k}_4[c] =$ EVALÚA($\mathbf{f}[c], t + h, \mathbf{y}_3$))
21: Sincroniza
22: $\mathbf{x}_j[c] = \mathbf{x}_j[c] + \frac{h}{6}\left(\mathbf{k}_1[c] + 2\mathbf{k}_2[c] + 2\mathbf{k}_3[c] + \mathbf{k}_4[c]\right)$
23: **if** $c = 1$ **then**
24: $t = t + h$
25: Sincroniza
26: $\mathbf{\Phi}[c][r] = \frac{\mathbf{x}_j[c] - \mathbf{x1}[c]}{\xi}$

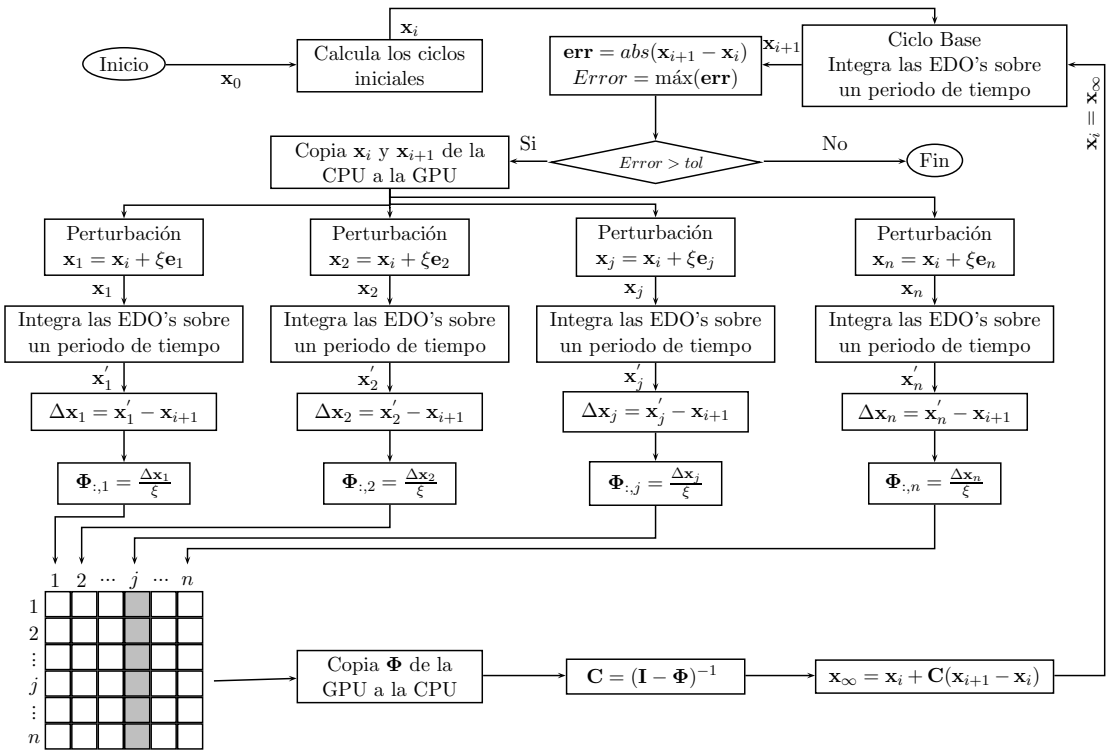


Figura 3.8: Método de DN tradicional usando una GPU para calcula la matriz $\mathbf{\Phi}$.

Los Algoritmos 21 y 10 muestran el método DN tradicional; el Algoritmo 10 muestra el proceso secuencial del método DN y el Algoritmo 21 muestra el proceso DN tradicional calculando la matriz $\mathbf{\Phi}$ con una GPU. El proceso necesario para calcular $\mathbf{\Phi}$ hace diferentes a ambos algoritmos, el resto del proceso es igual.

A continuación se describe el procedimiento para calcular la matriz $\mathbf{\Phi}$ usando una GPU, para ello son necesarios los Algoritmos 21 y 20. En el Algoritmo 21, las líneas que son necesarias para procesar $\mathbf{\Phi}$ son las siguientes:

- La línea 13 copia los vectores **x** y **x1** a la GPU, ambos vectores son necesarios para el cálculo de $\mathbf{\Phi}$.

- La línea 14 llama al procedimiento que se ejecutará en la GPU con n bloques de n hilos; cada bloque peocesará una columna de la matriz $\mathbf{\Phi}$ y los n hilos en cada bloque ayudan a procesar una vez el método de RK4 (que servirá para obtener la integración de un periodo de tiempo).

- Una vez que se ha calculado $\mathbf{\Phi}$ en la GPU, se copia a la CPU (en la línea 15).

El Algoritmo 20 se ejecuta por completo en la GPU y es el cálculo de $\mathbf{\Phi}$ en paralelo. A continuación se describe cada una de sus líneas:

- La línea 2 indica a qué bloque pertenece el hilo que va a procesar el resto de la función.
- La línea 3 indica el número de hilo que procesará las siguientes líneas de la función.
- La línea 4 asigna un valor a la variable ξ; este valor es el que se usará para perturbar las variables de estado en el cálculo de las columnas de la matriz $\mathbf{\Phi}$.
- La línea 5 copia la posición c del arreglo $\mathbf{x}$ a la posición c del vector $\mathbf{x}_j$. Cuando los n hilos del bloque ejecutan la línea, el resultado es la copia completa del vector $\mathbf{x}$ al vector $\mathbf{x}_j$.
- La línea 6 realiza una sincronización; la sincronización es necesaria para asegurar que $\mathbf{x}_j = \mathbf{x}$.
- La línea 8 realiza una perturbación en la posición r del arreglo $\mathbf{x}_j$. El efecto es realizar una perturbación a la variable número r para calcular la columna r de la matriz $\mathbf{\Phi}$.
- La línea 7 es una condición que indica que si el hilo es el número 1 del bloque r entonces se podrá ejecutar la línea siguiente (que es la perturbación en la posición r); los demás hilos del bloque no hacen nada y esperan en la línea que los sincroniza (línea 9). El resultado hasta este punto es $\mathbf{x}_j = \mathbf{x} + \xi \mathbf{e}_j$.
- De la línea 11 a la línea 22 se ejecuta el método RK4 en paralelo en los n bloques de hilos. Idealmente están trabajando n^2 hilos (elementos de proceso). La tarea que realiza cada uno de los n^2 hilos se indica a continuación:
  - En la línea 11 se evalúa la EDO número c; el resultado es el valor de la posición c del vector $\mathbf{k}_1$.
  - En la línea 12 se evalúa la expresión $\mathbf{y}_1[c] = \mathbf{x}_j[c] + \frac{h}{2}\mathbf{k}_1[c]$; el resultado es el valor de la posición c del vector $\mathbf{y}_1$ que se ocupará para el cálculo del vector $\mathbf{k}_2$.
  - En la línea 13 se sincronizan todos los hilos. El resultado hasta este punto son los vectores $\mathbf{k}_1$ y $\mathbf{y}_1$.
  - En la línea 16 se sincronizan todos los hilos. El resultado hasta este punto son los vectores $\mathbf{k}_2$ y $\mathbf{y}_2$.

- En la línea 19 se sincronizan todos los hilos. El resultado hasta este punto son los vectores $\mathbf{k}_3$ y $\mathbf{y}_3$.
- En la línea 21 se sincronizan todos los hilos. El resultado hasta este punto es el vector $\mathbf{k}_4$.
- En la línea 22 se evalúa $\mathbf{x}_j[c] = \mathbf{x}_j[c] + \frac{h}{6}\left(\mathbf{k}_1[c] + 2\mathbf{k}_2[c] + 2\mathbf{k}_3[c] + \mathbf{k}_4[c]\right)$.
- En la línea 25 se sincronizan todos los hilos. El resultado hasta este punto es la integración en el paso h.

- La línea 10 es un ciclo que indica que el método de RK4 se repite m veces.
- La línea 25 sincroniza los hilos; el resultado hasta este punto es la integración de un periodo de tiempo.
- La línea 26 realiza la operación $\mathbf{\Phi}[c][r] = \frac{\mathbf{x}_j[c] - \mathbf{x1}[c]}{\xi}$. El resultado al terminar los n hilos de los n bloques de hilos es la matriz $\mathbf{\Phi}$.

Algoritmo 21 Método de Diferenciación Numérica usando una GPU para procesar $\mathbf{\Phi}$

```
1:  function DIFERENCIACIÓN-NUMÉRICA-GPU(f, x0)
2:      n=número-de-variables-de-estado
3:      T = 1/60
4:      m = 512
5:      h = T/m
6:      t = 0
7:      periodos = 8
8:      x =CICLOS-INICIALES(f, x0, n, h, t, m, periodos, T)
9:      x1 = INTEGRA-PERIODO(f, x, n, h, t, m)
10:     error = ERROR(x, x1, n)
11:     I = UNITARIA(n)
12:     while error > tolerancia do
13:         Copia x y x1 de la CPU a la GPU
14:         <<< n, n >>>MATRIZ-IDENTIFICACIÓN-GPU(f, x, x1, h, t, m, Φ)
15:         Copia Φ de la GPU a la CPU
16:         x = CICLO-LIMITE(Φ, I, n, x, x1)
17:         x1 = INTEGRA-PERIODO(f, x, n, h, t, m)
18:         error = ERROR(x, x1, n)
19:     return x1
```

3.9.2. Propuesta del Método de DN Selectivo en Paralelo Basada en GPU's

Cuando se tienen suficientes elementos de proceso para hallar de forma simultánea todas las columnas de la matriz $\mathbf{\Phi}$, la metodología de DN selectiva deja de ser importante, ya que todas las columnas se pueden calcular de forma simultanea (sin importar la cantidad de columnas, debido a que hay suficientes elementos de proceso). Debido a la cantidad real de elementos de proceso que tienen las GPU's y la cantidad de elementos de proceso que se pueden usar, la matriz $\mathbf{\Phi}$ selectiva sigue siendo importante.

En la Figura 3.9 se muestra el método DN selectivo usando una GPU para el cálculo $\mathbf{\Phi}$. En la Sección 3.3 se explicó el procedimiento para la determinación de la matriz de identificación selectiva; se utiliza en esta sección y se agrega el cálculo paralelo de las columnas de $\mathbf{\Phi}$.

Primeramente, se calculan los vectores $\mathbf{x}_i$, $\mathbf{x}_{i+1}$ y $\mathbf{err}$ en la CPU. Una vez que se tienen todos los errores de las variables de estado se procede a calcular el error máximo ($Error$) que tiene una variable de estado; si el $Error$ es menor que una tolerancia dada, entonces todas las variables de estado han alcanzado su estado estacionario periódico, de lo contrario se procede a calcular $\mathbf{\Phi}$.

Para calcular $\mathbf{\Phi}$ en la GPU es necesario copiar los vectores $\mathbf{x}_i$, $\mathbf{x}_{i+1}$ y $\mathbf{err}$ de la CPU a la GPU y realizar lo siguiente:

1. El elemento de proceso j (que es el bloque de hilos número j) revisa la condición C_j (que es $CDN = 0$ or $\mathbf{err}[j] > tolerancia$). La condición revisa si es la primera vez que se calcula $\mathbf{\Phi}$ (con $CDN = 0$) o si el error asociado a la variable j es mayor a la tolerancia especificada, entonces se procede a calcular la columna j de la matriz $\mathbf{\Phi}$, de lo contrario, la columna j de la matriz $\mathbf{\Phi}$ calculada en la iteración anterior será la columna j de $\mathbf{\Phi}$.

2. Las columnas a obtener, siguen el mismo proceso descrito en la Sección 3.9.1.

Una vez que se tiene la matriz $\mathbf{\Phi}$ en la GPU, se copia a la CPU para realizar las operaciones $\mathbf{C} = (\mathbf{I} - \mathbf{\Phi})^{-1}$ y $\mathbf{x}_\infty = \mathbf{x}_i + \mathbf{C}(\mathbf{x}_{i+1} - \mathbf{x}_i)$.

El siguiente paso del algoritmo es hacer que los nuevos valores de las variables de estado sean los valores de las variables en el ciclo límite, esto se logra haciendo que $\mathbf{x}_i = \mathbf{x}_\infty$, para posteriormente realizar un periodo de integración para obtener un nuevo vector $\mathbf{x}_{i+1}$ que servirá para encontrar los nuevos errores ($\mathbf{err}$) de las variables de estado y proceder con el nuevo cálculo de la matriz de identificación (y repetir el proceso) o finalizar la tarea.

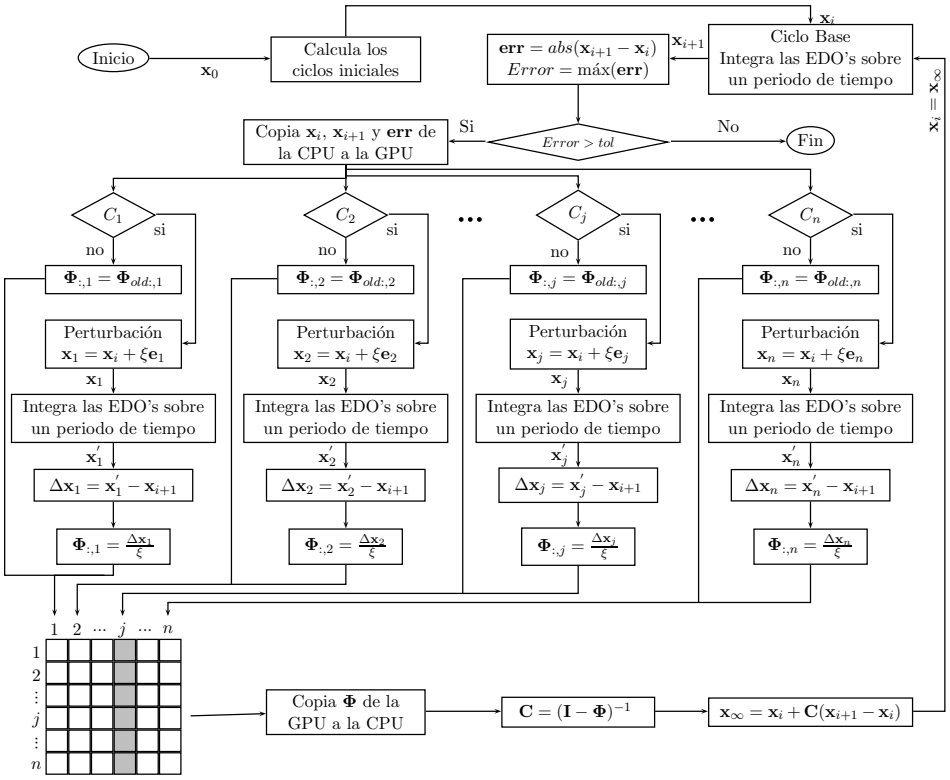


Figura 3.9: Método de DN selectivo usando una GPU para calcular la matriz $\mathbf{\Phi}$.

3.9.3. Diferenciación Numérica usando la Matriz Φ Constante y una GPU para el Cálculo de la Matriz Φ

El método de DN usando $\mathbf{\Phi}$ constante solamente calcula la matriz $\mathbf{\Phi}$. La inversa de la matriz, también se calcula una sola vez. Cuando se evitan los cálculos de la matriz $\mathbf{\Phi}$ y de la inversa de forma repetida, se reducen el número de operaciones para llegar a la solución del estado estacionario periódico y con ello el tiempo de solución también es significativamente menor (ver Sección 3.5).

La propuesta en esta sección es resolver el bloque 2 (la matriz $\mathbf{\Phi}$) con una GPU. En la sección 3.9.1 se explicó el proceso para calcular las columnas de la matriz $\mathbf{\Phi}$ usando n bloques de n hilos cada bloque. El cálculo de la matriz $\mathbf{\Phi}$ en esta propuesta es el mismo; la diferencia está en el proceso que sigue al cálculo de la matriz $\mathbf{\Phi}$, mostrado en la Figura 3.4.

3.9.4. Diferenciación Numérica usando la Matriz Φ Constante, Multi-CPUs, Multi-GPU's y Proceso Paralelo en la Inversa de la Matriz

La Figura 3.4 muestra el algoritmo de solución del método DN usando GPU's con $\mathbf{\Phi}$ constante. El algoritmo tiene cuatro bloques que son importantes para reducir el tiempo de solución. El bloque 2 calcula la matriz $\mathbf{\Phi}$, es el más importante hasta un número finito de variables de estado. Cuando el bloque 2 se resuelve solamente una vez y su proceso es paralelo, los otros bloques toman mayor importancia debido a que el tiempo que consumen ya es muy significativo.

La propuesta en esta sección es reducir el tiempo en todos los bloques del algoritmo de la Figura 3.4. Para ello, es necesario aplicar procesamiento paralelo en todos los periodos de integración (que incluye los periodos de integración de los ciclos iniciales y el ciclo base en el bloque 1, la matriz $\mathbf{\Phi}$ en el bloque 2 y los ciclos finales en el bloque 4). En el bloque 3, que es el cálculo de las variables de estado en el Ciclo Límite es necesario de una inversa de una matriz; parte del procedimiento para calcular la inversa se puede realizar en paralelo (ver Sección 3.7).

En el bloque 2 se calcula la matriz de identificación; en ella se pueden utilizar n^2 elementos de proceso. Una de las GPU's con mayor cantidad (en la fecha de termino de esta tesis) de elementos de proceso es la Nvidia Tesla K80; tiene 4992. En un sistema de con 70 variables de estado se pueden utilizar 4900 elementos de proceso. Para sistemas con más variables de estado, el número de elementos de proceso que se tienen disponible en una GPU es insuficiente. En esta sección se propone el uso de varias GPU's y CPU's para el cálculo de $\mathbf{\Phi}$; con esta propuesta se tendrán más elementos de proceso disponibles para obtener $\mathbf{\Phi}$.

La Figura 3.10 se muestra la propuesta para el cálculo de la matriz $\mathbf{\Phi}$ con varias GPUs y CPUs. El mayor esfuerzo computacional (hasta cierto número finito de variables de estado) en el algoritmo DN con $\mathbf{\Phi}$ constante es el cálculo de la matriz $\mathbf{\Phi}$, por lo que, si este proceso se resuelve con más de un dispositivo con varios elementos de proceso cada uno, entonces se tendrá una mayor reducción en el tiempo de proceso.

En la Figura 3.10 se tienen g GPUs y k CPUs; cada GPU y CPU procesa un número de columnas, dependiendo de su capacidad de procesamiento. Para el cálculo de las columnas, es necesario distribuir el trabajo entre los elementos de proceso disponibles. En la figura se observa que la GPU 1 procesa de la columna 1 a la columna a; la GPU 2 iniciará calculando la columna

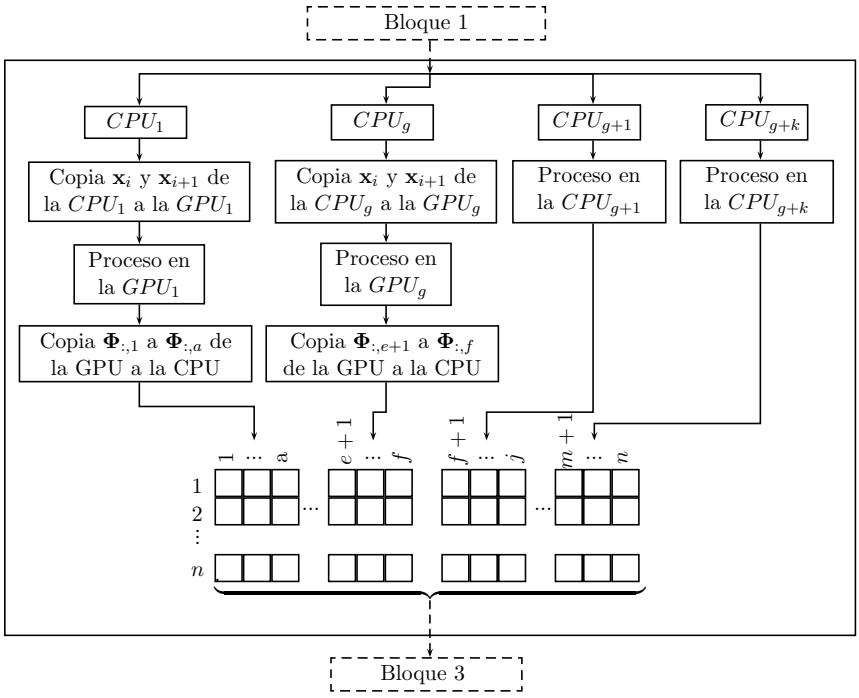


Figura 3.10: Cálculo de la matriz $\mathbf{\Phi}$ usando varias GPU's y varias CPU's.

$a+1$. La GPU $g-1$ (penúltima GPU) deberá procesar hasta la columna e y la GPU g (última GPU) procesará de la columna $e+1$ a la columna f. Las columnas restantes (de la columna $f+1$ a la columna n) serán obtenidas por las CPU's disponibles.

Para trabajar con varias GPU's y CPU's de forma simultánea es necesario de herramientas de procesamiento en paralelo con CPU's (que puede ser openmp o threads) y herramientas para la programación de GPU's (CUDA). En esta tesis se usó openMP para procesamiento paralelo con CPU's y CUDA para la programación de las GPU's.

Cuando se usan g GPUs de forma simultánea, es necesario de g núcleos de la CPU, ya que la comunicación (en la arquitectura Fermi) entre la GPU y la CPU se da de uno a uno (un núcleo de la CPU con una GPU). En un sistema de cómputo, es posible que se tengan mayor número de núcleos en la CPU que GPU's, por lo tanto, los núcleos libres de la CPU pueden ayudar en el cálculo de $\mathbf{\Phi}$.

Para el cálculo de las columnas en las GPU's, es necesario hacer una copia de los vectores $\mathbf{x_i}$ y $\mathbf{x_{i+1}}$ de la CPU a cada GPU. Cada GPU procesará varias columnas de forma simultánea. Una vez que procesan todas las columnas que le fueron asignadas a la GPU, se hace una copia de esas

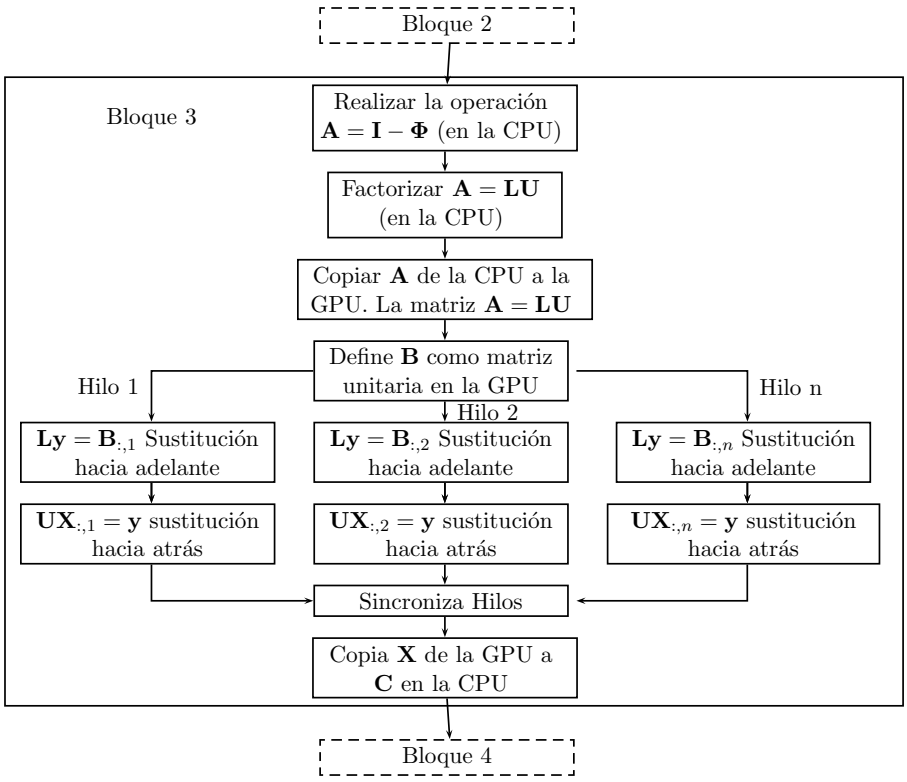


Figura 3.11: Inversa de una matriz usando factorización LU y la sustitución hacia adelante y hacia atrás en paralelo

columnas de la GPU a la CPU y se colocan en la columnas de la matriz $\mathbf{\Phi}$ en la CPU.

Para el proceso en la CPU, no es necesario realizar copias de un dispositivo a otro, dado que los vectores $\mathbf{x_i}$ y $\mathbf{x_{i+1}}$ se ven desde cualquier núcleo de la CPU. Después de calcular cada columna (que le toca procesar a cada núcleo de la CPU o CPU's) se procede a colocarla en la matriz $\mathbf{\Phi}$. Una vez que se calculó $\mathbf{\Phi}$, el proceso sigue en el bloque 3.

Al terminar ejecutar el procedimiento que está en la Figura 3.10 se tiene $\mathbf{\Phi}$ en la CPU y para reducir el tiempo de ejecución del bloque 3, se propone el procedimiento mostrado en la Figura 3.11, que se para reducir el tiempo para calcular la inversa de la matriz. El procedimiento se explica a continuación:

- Una vez que se calculó la matriz $\mathbf{\Phi}$ se realiza la operación $\mathbf{A} = \mathbf{I} - \mathbf{\Phi}$ y se factoriza la matriz $\mathbf{A} = \mathbf{LU}$; este proceso se realiza en la CPU.

- El siguiente paso del procedimiento es copiar la matriz $\mathbf{A}$ de la CPU a la GPU. La tarea de

la GPU es obtener $\mathbf{A}^{-1}$.

- En la GPU es necesario definir la matriz unitaria (en la matriz $\mathbf{B}$).
- El hilo n de la GPU obtendrá el vector $\mathbf{y}$ con un procedimiento de sustitución hacia adelante con la expresión $\mathbf{Ly} = \mathbf{B}_{:,n}$. $\mathbf{B}_{:,n}$ es la columna n de la matriz unitaria.
- Después de obtener el vector $\mathbf{y}$, se obtiene la columna n de la matriz $\mathbf{X}$ por una sustitución hacia atrás de la expresión $\mathbf{UX}_{:,n} = y$. En $\mathbf{X}$ está la inversa de la matriz $\mathbf{A}$.
- El último paso para tener la inversa de $\mathbf{A}$ en la CPU es copiar la matriz $\mathbf{X}$ de la GPU a la matriz $\mathbf{C}$ en la CPU.

La reducción en los tiempos de ejecución en el cálculo de la inversa se analizó en la Sección 3.7.

3.10. Conclusiones

En este capítulo se han propuesto variantes de proceso para incrementar el desempeño computacional del método DN. Dos de ellas disminuyen el esfuerzo computacional y las demás propuestas distribuyen el esfuerzo computacional entre los elementos de proceso.

Las dos propuestas que reducen el esfuerzo computacional fueron el método DN selectivo y el método DN con la matriz de identificación constante. En ambas metodologías se presentó su complejidad y se observó la reducción de cálculos en cada una de ellas. La reducción en los cálculos está directamente relacionada con la reducción en el tiempo de simulación.

Las demás propuestas se basan en el análisis de procesamiento paralelo en algunas partes de los métodos: DN tradicional, DN selectivo y DN con matriz de identificación constante. Se logra una disminución en el tiempo de solución cuando se utiliza procesamiento paralelo. El proceso que más reducción logra es el que obtiene la matriz de identificación.

En este capítulo de introduce el cálculo paralelo del método de Runge Kutta de cuarto orden en el cálculo de los periodos de integración (necesarios en casi todo el proceso del método de DN en sus diferentes versiones). Al introducir el procesamiento paralelo del método, reduce en n la complejidad de los periodos de integración. La matriz de identificación puede ser calculada

aplicando doble procesamiento paralelo; uno para el cálculo de las columnas de la matriz y otro para el proceso interno del cálculo de cada columna.

Se mostró que la complejidad computacional para obtener $\mathbf{\Phi}$ es cuadrática en su procesamiento secuencial; cambia a complejidad lineal cuando se aplica procesamiento paralelo para determinar las columnas de $\mathbf{\Phi}$ y a complejidad constante cuando se aplica procesamiento paralelo en el método RK4.

El cálculo de la inversa se requiere en el método de DN en todas sus versiones. En el capítulo anterior se mostró que tiene una complejidad cúbica y que es el elemento que requiere más esfuerzo computacional después de cierto tamaño del sistema eléctrico. En este capítulo se introduce el procesamiento paralelo en parte del proceso de la inversa (que usa la factorización LU) logrando reducir el coeficiente (de $\frac{4}{3}$ a $\frac{1}{3}$) que multiplica al término cúbico. Se mostró que cuando se utiliza procesamiento paralelo, el cálculo de la inversa toma mayor importancia en el tiempo de la simulación.

Los análisis de procesamiento en paralelo se realizaron independiente de los elementos de proceso que se pueden utilizar. En este capítulo se propone utilizar GPUs junto con la CPU para procesar los métodos de DN en sus diferentes versiones. Se utiliza la configuración multi-GPU y Multi-CPU para calcular la matriz de identificación en las que se pueden usar n^2 elementos de proceso.

Capítulo 4

Casos de Estudio

4.1. Introducción

En este capítulo se presentan estudios en los cuales se aplican los algoritmos de proceso y solución descritos en el Capítulo 3. El tiempo obtenido en cada una de las simulaciones, usando los diferentes algoritmos se comparan con el tiempo obtenido para procesar de forma secuencial el método DN tradicional. Los sistemas de prueba que se analizan en las diferentes propuestas son los sistemas de prueba modificados del IEEE de 14, 30, 57 y 118 nodos.

En este capítulo también se presentan las formas de onda, contenidos armónicos y convergencia al ciclo límite de algunas variables de estado de los sistemas de prueba.

4.2. Caso de Estudio 1. Resultados del Método DN selectivo

La complejidad computacional del método de DN tradicional se demostró en la Sección 2.3.7 y se dedujo la Ecuación (2.27) (que se presenta en la Ecuación (4.1)) para medir el tiempo total para hallar la solución en estado estacionario periódico de un sistema eléctrico.

$$t_{dn} = (b + 1 + p)\, mn\, (4t_1 + 3t_2 + t_3) + p\left(n^2 m\, (4t_1 + 3t_2 + t_3) + \frac{4n^3}{3} t_2\right) \tag{4.1}$$

La Ecuación (4.1) se puede escribir como,

$$t_{dn} = (b+1+p)\,t_p + p\left(nt_p + \frac{4n^3}{3}t_2\right) = (b+1+p(n+1))\,t_p + p\left(\frac{4n^3}{3}t_2\right) \tag{4.2}$$

La solución para los sistemas de prueba del IEEE se obtiene a partir de la propuesta del método DN selectivo descrita en la Sección 3.3. En la Sección 3.3 se encontró una expresión para calcular el tiempo total para ejecutar el método DN selectivo, repetida por conveniencia en la Ecuación (4.3).

$$t_{dns} = (b+1+p)\,t_p + p\left(nt_p + \frac{4n^3}{3}t_2\right) - \sum_{j=2}^{p} c_j t_p \tag{4.3}$$

El término $\sum_{j=2}^{p} c_j t_p$ es la parte que se reduce del tiempo requerido por el método DN tradicional. La variable c_j es el número de variables de estado que llegaron al estado estacionario periódico (y es el número de columnas que no se calculan de la matriz $\mathbf{\Phi}$) después de la iteración $j-1$ del método DN.

Para mostrar la importancia del método de DN selectivo, en la Tabla 4.1 se muestra una comparación del número de periodos de integración que se necesitan en el método DN tradicional y del método DN selectivo. La Tabla 4.1 contiene la información siguiente:

La primera columna muestra el número de nodos del sistema de prueba del IEEE.

La segunda columna muestra el número de variables de estado n que hay en el modelo del sistema. En la Ecuación (4.1), n representa el número de variables de estado.

La columna tres muestra el número de variables de estado (VE) que han convergido al terminar la primera iteración de método DN. En la Ecuación (4.3), c_j (con $j = 2$) representa el número de variables de estado que convergieron después de la primera iteración del método DNS.

La cuarta columna presenta el número de periodos de integración que se usaron en los ciclos iniciales. En algunos sistemas se observa que a mayor número de periodos de integración en los ciclos iniciales se tienen mayor número de variables de estado que llegan al estado estacionario periódico después de la primera iteración del método DN; eso significa que se evita calcular mayor número de columnas de $\mathbf{\Phi}$. En las Ecuaciones 4.2 y 4.3, b es el número de periodos de integración en los ciclos iniciales.

La quinta columna muestra el número de periodos de integración requeridos para llegar a la solución en estado estacionario periódico, usando el método DN tradicional. La solución se obtiene en dos iteraciones; $(b + 1 + p(n + 1))$ es el número de periodos de integración que son necesarios por el método DN tradicional. Por ejemplo, para el sistema eléctrico de 14 nodos con 54 variables de estado se procesan $(13 + 1 + 2(54 + 1)) = 124$ periodos (ciclos).

La columna seis muestra el número de periodos requeridos por el método DN selectivo (DNS). Considerando que c_j es el número de columnas que no se calculan en la segunda iteración del método de DN. Restando la columna 3 a la columna 5 se tiene la columna 6 de la Tabla 4.1.

Tabla 4.1: Reducción de periodos de tiempo con la metodología NDS.

Sistemas de prueba modificados de IEEE	n	VE que convergieron después de la primera iteración	Ciclos Iniciales	DN (periodos)	DNS (periodos)
14	54	42	13	124	82
30	88	59	8	187	128
57	143	140	20	309	169
118	357	351	8	725	374

En la columna 3 se presenta la información más importante que produce el algoritmo DNS y es el número de variables de estado que llegaron a la solución después de la primera iteración del método DN. El número que se presenta en esta columna también es el número de columnas que no se calcularán en la siguiente iteración. Por ejemplo, en el sistema de 14 nodos, 42 variables estado de 54 llegaron al estado estacionario periódico, por lo tanto, solamente 12 columnas se calculan en la segunda evaluación de $\mathbf{\Phi}$. De la misma forma, para los sistemas de 30, 57 y 118 nodos, se calculan solamente 29, 3 y 6 columnas, respectivamente. La diferencia de las columnas 5 y 6 la columna 3, que contiene el número de periodos que integración que no se calcularon.

En la Tabla 4.2 se muestran los tiempos de ejecución de los algoritmos DN (columna 3) y DNS (columna 4). También se muestra un factor o ganancia ($g = \frac{DN(ms)}{DNS(ms)}$) (columna 5) que compara el tiempo de los dos algoritmos, para los sistemas de prueba del IEEE (columna 1).

El factor g indica el número de veces que un algoritmo A se ejecuta cuando el algoritmo B se ejecuta una vez. Su valor se obtiene por medio de: $g = \frac{B}{A}$.

Tabla 4.2: Reducción en tiempos en método DNS

Sistemas de prueba modificados de IEEE	n	DN (ms)	DNS (ms)	$\frac{DN(ms)}{DNS(ms)}$	$\frac{DN(periodos)}{DNS(periodos)}$
14	54	1123	744	1.51	1.51
30	88	1419	969	1.46	1.46
57	143	2347	1316	1.78	1.82
118	357	11546	6834	1.69	1.94

La ganancia g servirá para comparar de forma cuantitativa los dos algoritmos con los datos utilizados. Con el valor g se pueden tener tres casos:

1. Primer caso: si $g = 1$, los algoritmos se ejecutan en el mismo tiempo.

2. Segundo Caso: si $g > 1$, el algoritmo A se ejecuta en menor tiempo. Por ejemplo si $g = 2$, se puede decir que el algoritmo B requiere dos veces más que el algoritmo A.

3. Tercer caso: si $g < 1$, el algoritmo A se ejecuta en mayor tiempo.

En la columna 5 se observa que en todos los casos la ganancia es mayor a uno. La ganancia mayor entre los sistemas de prueba del IEEE se tiene en el sistema 57 nodos; el valor de esta ganancia es de 1.78. El promedio de las ganancias entre los cuatro sistemas de prueba es de 1.61.

El peor escenario para el método de DNS es que calculen todas las columnas de la matriz $\mathbf{\Phi}$ como lo hace el método DN tradicional. En este escenario, los métodos DNS y DN tradicional son equivalentes, por lo tanto, la ganancia será de 1.0. En ningún caso el método DNS tendrá una ganancia menor a 1.0.

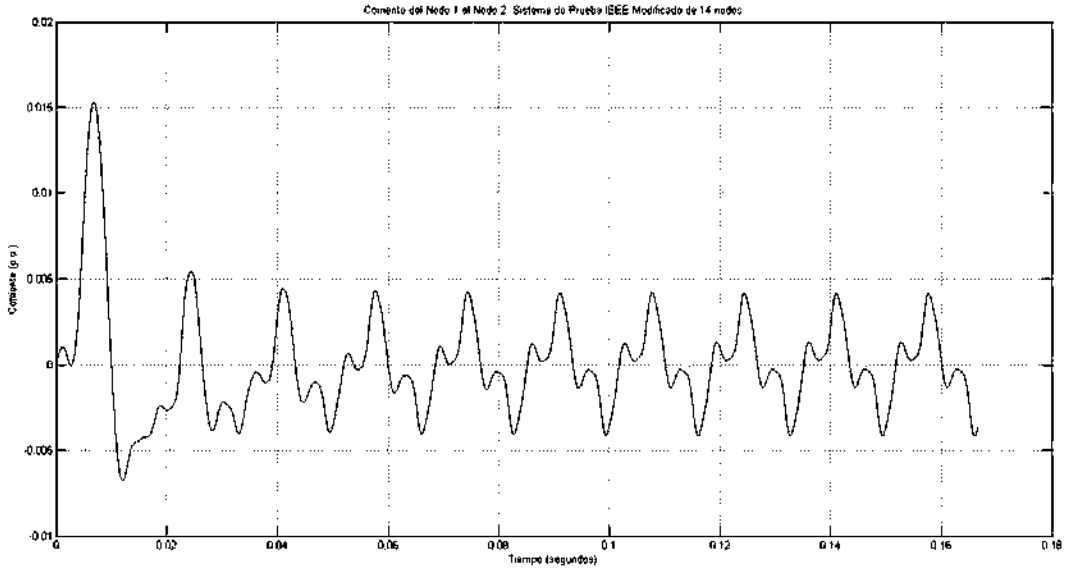
En la columna 6 de la Tabla 4.2 se presenta el resultado de dividir el número de periodos de integración del método DN tradicional entre el número de periodos de tiempo del método DN selectivo $\left(\frac{DN(periodos)}{DNS(periodos)}\right)$. La diferencia entre las columnas 5 y 6 se presenta por el término $\frac{4n^3}{3}t_2$ (que es el tiempo necesario para obtener le inversa de una matriz) en las ecuaciones 2.28 y 3.20. Cuando n se incrementa, la inversa toma mayor importancia, por lo tanto, cuando n incrementa el término $\frac{4n^3}{3}t_2$, afecta negativamente la ganancia del método DNS.

La solución en estado estacionario periódico de los sistemas de prueba se muestran para algunas variables.

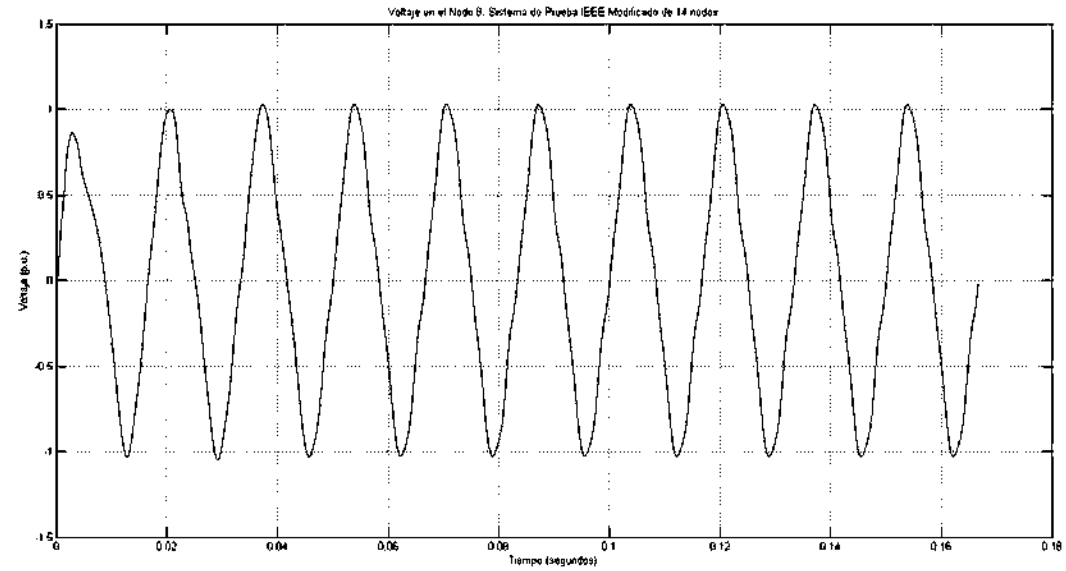
- Sistema de prueba de IEEE de 14 nodos: para este sistema se muestra:
  - En la Figura 4.1 (a) la corriente (en p.u.) del nodo 1 al nodo 2.
  - En la Figura 4.1 (b) el voltaje (en p.u.) en el nodo 8. El voltaje está en su rango de operación.
  - En la Figura 4.2 (a) se muestra el Ciclo Límite del voltaje del nodo 8 con respecto a la corriente del nodo 1 a 2; en la figura se observa de forma clara que en los ciclos iniciales esa solución se encuentra muy alejada del Ciclo Límite, esto se presenta al iniciar el proceso y conforme transcurre la simulación las variables de estado convergen al Ciclo Límite.
  - En la Figura 4.2 (b) se presenta la forma de onda de la corriente en el generador conectado al nodo 5 y su espectro armónico. Se observa que los armónicos 3, 5 y 7 contribuyen en un 62 %, 8 % y 5 % de la fundamental, respectivamente y los demás armónicos no tienen contribución o no es significativa. Estas magnitudes indican en todos los casos que rebasan los valores permisibles para la operación segura de la red eléctrica, de acuerdo a la norma IEEE-519 [IEEE-519 Std. 1992].
- Sistema de prueba de IEEE de 57 nodos: para este sistema se presenta:
  - En la Figura 4.3 (a) la corriente (en p.u.) del generador conectado al nodo 8.
  - En la Figura 4.3 (b) el voltaje (en p.u.) en el nodo 29.
  - En la Figura 4.4 (a) la corriente del nodo 13 al nodo 14.
  - En la Figura 4.4 (b) el Ciclo Límite del voltaje del nodo 29 con respecto a la corriente del nodo 13 al 14.

4.3. Caso de Estudio 2. Resultados del Método DN con una GPU

En este caso de estudio se presenta la solución en Estado Estacionario Periódico de los sistemas de prueba modificados del IEEE de 14, 30, 57 y 118 nodos; en donde para el método

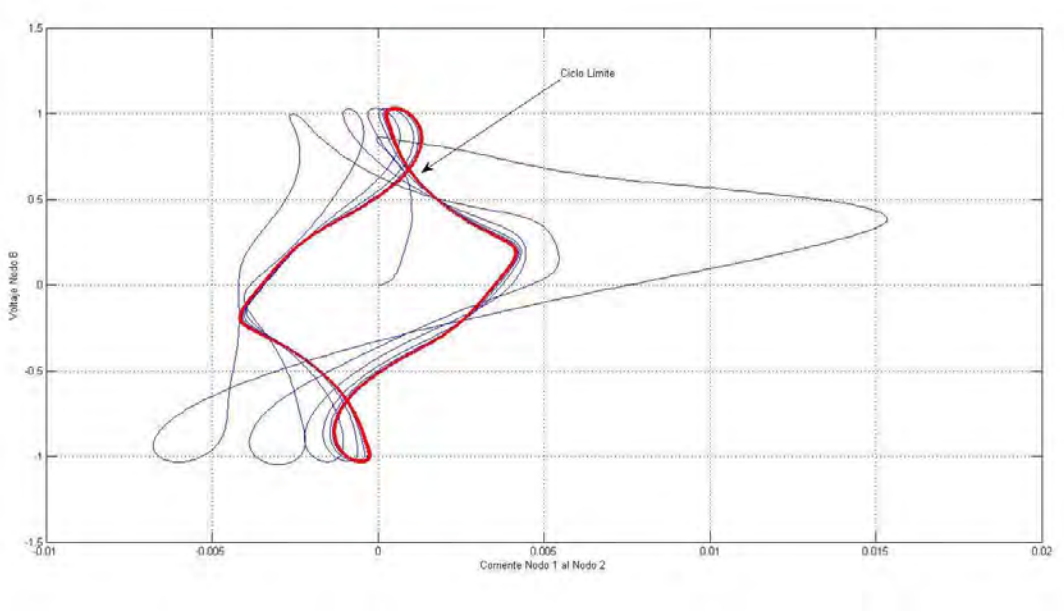


(a) ***Corriente del nodo 1 al nodo 2***

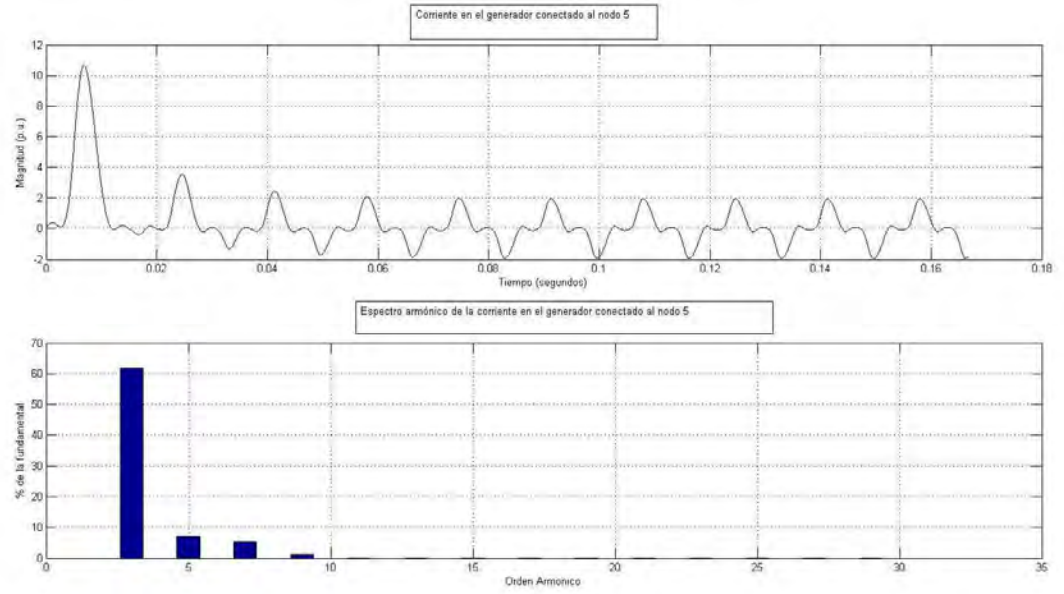


(b) ***Voltaje en el nodo 8***

Figura 4.1: Formas de onda de algunas variables de estado del sistema de prueba del IEEE de 14 nodos modificado.

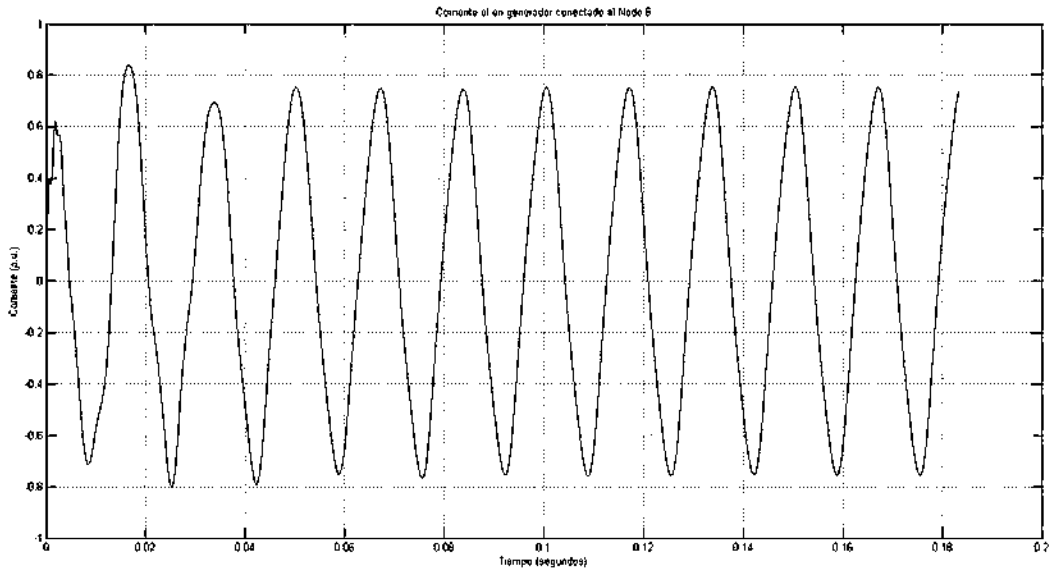


(a) ***Ciclo Límite del voltaje del nodo 8 con respecto a la corriente del nodo 1 al nodo 2***

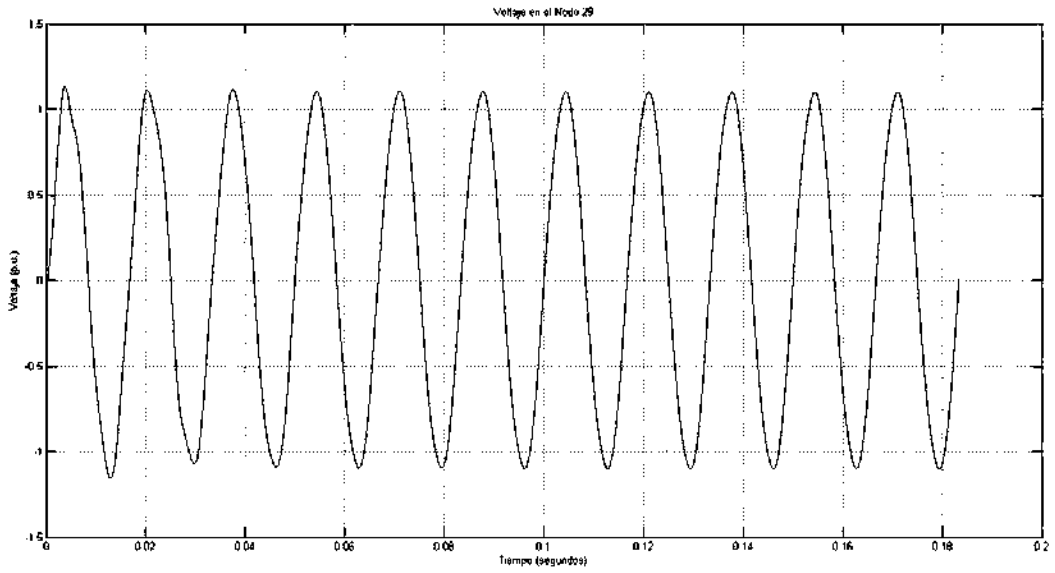


(b) ***Espectro armónico de la corriente del generador conectado al nodo 5***

Figura 4.2: Ciclo Límite y espectro armónico en algunas variables del sistema de prueba del IEEE de 14 nodos modificado.

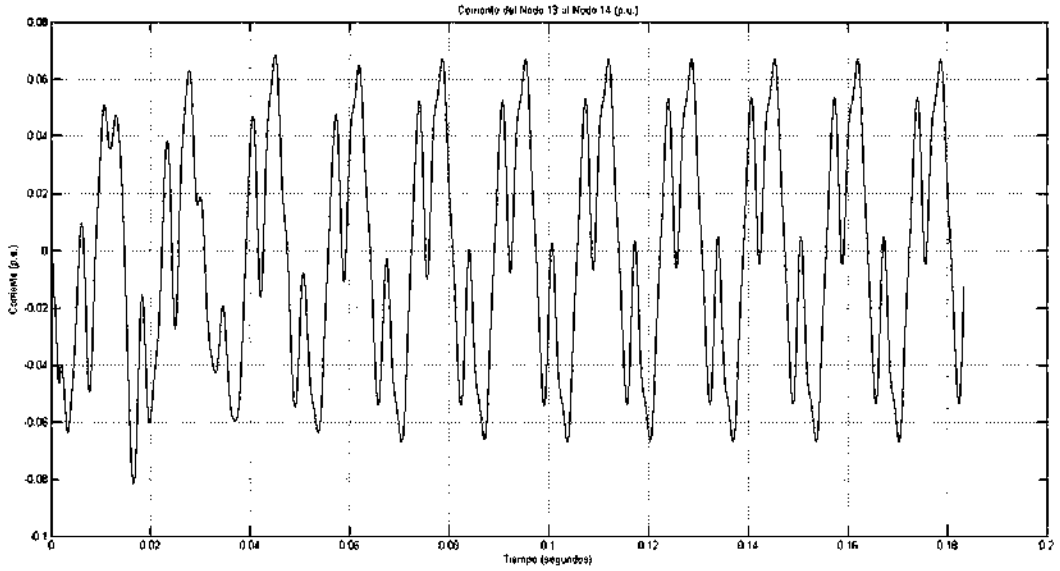


(a) ***Corriente en el generador conectado al nodo 6***

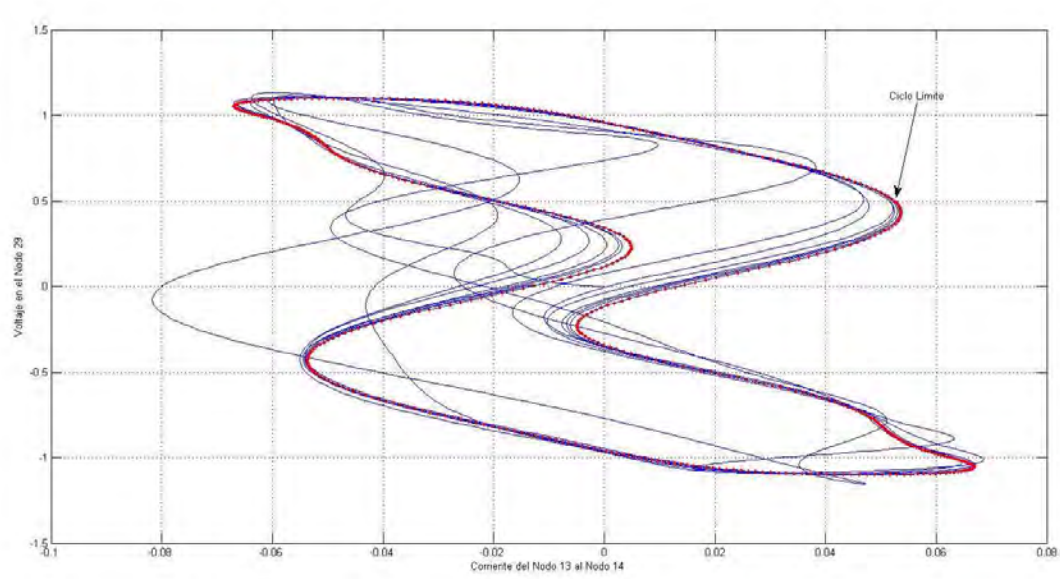


(b) ***Voltaje en el nodo 29***

Figura 4.3: Formas de onda de algunas variables de estado del sistema de prueba del IEEE de 57 nodos modificado.



(a) ***Corriente del nodo 13 al nodo 14 en p.u.***



(b) ***Ciclo Límite del voltaje del nodo 29 con respecto a la corriente del nodo 13 al nodo 14***

Figura 4.4: Corriente del nodo 13 al nodo 14 y Ciclo Límite de un sistema de prueba del IEEE de 57 nodos modificado.

DN se procesa la matriz $\mathbf{\Phi}$ en paralelo, utilizando una GPU (DNC); los tiempos para obtener la solución se comparan con los tiempos para ejecutar el método DN convencional procesando todo el algoritmo de forma secuencial.

Se espera una reducción de tiempo importante, ya que las columnas de $\mathbf{\Phi}$ se obtienen de forma simultánea y el método de RK4 de cada periodo de integración para calcular cada columna de la matriz $\mathbf{\Phi}$ se procesa en paralelo. Idealmente, la reducción en el tiempo para obtener $\mathbf{\Phi}$ es de n^2 veces, considerando n^2 elementos de proceso.

La simulación fue realizada usando un procesador Intel Xeon E5606 que trabaja a una frecuencia de 2.2 GHZ; la GPU utilizada fue Nvidia Tesla C2075 con 448 núcleos CUDA, 6 GB de memoria y una frecuencia en los núcleos CUDA de 1.15GHz. Los 448 núcleos CUDA se distribuyen en 14 multiprocesadores de flujo (Stream Processsors (SM)); cada SM tiene 32 núcleos CUDA en la arquitectura Fermi.

La Tabla 4.3 muestra los tiempos y las ganancias que resultan de procesar en paralelo el método DN tradicional. Las columnas 1 y 2 muestran el número de nodos y variables de estado del sistema, respectivamente; la columna 3 el tiempo de ejecución del método DN tradicional, de forma secuencial; la columna 4 el tiempo de ejecución del método DN procesando $\mathbf{\Phi}$ en paralelo y finalmente, la columna 5 muestra la ganancia del método de DNC con respecto al método de DN.

Tabla 4.3: Reducción en tiempos en el método DNC

Sistemas de prueba modificados de IEEE	n	DN (ms)	DNC (ms)	$\frac{DN}{DNC}$
14	54	1123	205	5.5
30	88	1419	284	5.0
57	143	2347	555	4.2
118	357	11546	4252	2.7

La ganancia que se observa en la columna 5 es importante; se ve afectada negativamente por las siguientes consideraciones:

- Con la GPU que se realizaron las simulaciones se pueden procesar solamente 14 columnas de forma simultánea (una por cada SM) y en cada columna se pueden usar solamente 32 hilos

(cada SM tiene 32 hilos). Por lo tanto, algunos hilos realizarán mas operaciones en el cálculo del periodo de integración de cada columna y algunos SM tendrán que procesar más de una columna.

- En el método DN, la matriz $\mathbf{\Phi}$ se ejecuta de manera secuencial en el procesador a una frecuencia mayor que la frecuencia que manejan los elementos de proceso paralelo en la GPU.

- La inversa de la matriz se procesa secuencialmente y cuando el tiempo de proceso de $\mathbf{\Phi}$ disminuye, el tiempo para obtener la inversa de la matriz toma mayor importancia.

La ganancia del método DNC es más alta que la ganancia del método DNS; el promedio de la ganancia en los cuatro sistemas de prueba es de 4.35.

4.4. Caso de Estudio 3. Resultados del Método DNS con una GPU

En este caso se obtiene la solución en estado estacionario periódico de los sistemas de prueba modificados del IEEE usando el método DN selectivo con procesamiento paralelo en la matriz $\mathbf{\Phi}$ (DNSC); el proceso paralelo está basado en una GPU. Los resultados obtenidos en esta sección toma los elementos de las dos secciones anteriores.

La Tabla 4.4 muestra los tiempos y la ganancia del método DNSC, comparados con los tiempos del método DN. Las columnas 1 y 2 muestran los nodos y las variables de estado respectivamente de los sistemas de prueba; la columna 3 el tiempo de ejecución del método DN; la columna 4 el tiempo de ejecución del método DNSC y la columna 5 la ganancia del método DNSC respecto al método DN tradicional.

La ganancia obtenida en este estudio es en promedio de 5.6. La ganancia es mayor a la obtenida con el método de DNC. La diferencia de las ganancias del método DNC y el método DNSC en promedio es 1.25.

Tabla 4.4: Reducción en tiempos del método DNSC

Sistemas de prueba modificados de IEEE	n	DN (ms)	DNSC (ms)	$\frac{DN}{DNSC}$
14	54	1123	201	5.6
30	88	1419	206	6.9
57	143	2347	415	5.7
118	357	11546	2749	4.2

4.5. Caso de Estudio 4. Resultados del Método DNSC con un Proceso Paralelo en la Inversa de la Matriz

En este caso de estudio, la solución para los sistemas de prueba del IEEE considerados se obtienen del método DNS y procesamiento paralelo de $\mathbf{\Phi}$ con una GPU y aplicando procesamiento en paralelo en una parte de la inversa de la matriz (DNSCI).

La inversa de la matriz en el método de DN en sus diferentes versiones representa el proceso de mayor esfuerzo computacional a partir de un valor finito de variables de estado (ver Sección 2.3.6). Cuando la matriz $\mathbf{\Phi}$ se procesa en paralelo $\mathbf{C} = (\mathbf{I} - \mathbf{\Phi})^{-1}$ toma mayor importancia.

La Tabla 4.5 muestra los resultados obtenidos. Las columnas 1 y 2 muestran el número de nodos y variables de estado respectivamente de los sistemas de prueba; la columna 3 muestra el tiempo de cómputo del método DN; la columna 4 el tiempo de ejecución del método DN procesando $\mathbf{\Phi}$ en paralelo con una GPU y parte del proceso de la inversa de la matriz también con una GPU.La reducción en el tiempo final con respecto los resultados de la Sección 4.4 se debe al procesamiento en paralelo de parte del proceso de la inversa de la matriz.

La diferencia de la columna 5 de la Tabla 4.5 con respecto a la columna 5 de la Tabla 4.4 es el incremento de la ganancia del método de DNSCI debido a parte del procesamiento en paralelo de la inversa de la matriz (ver Sección 3.7), La diferencia es más importante mientras mas grande es el número de variables de estado (n).

En la Ecuación (2.19) se presenta la complejidad para obtener el proceso de sustitución hacia adelante y hacia atrás de la ecuación recursiva para obtener las varaibles de estado en el ciclo límite. Este proceso es necesario para el cálculo de la inversa de la matriz de forma secuencial. En

Tabla 4.5: Reducción en tiempos en el método DNSCI

Sistemas de prueba modificados de IEEE	n	DN (ms)	DNSCI (ms)	$\frac{DN}{DNSCI}$
14	54	1123	197	5.7
30	88	1419	201	7.1
57	143	2347	386	6.1
118	357	11546	2185	5.3

la Ecuación (3.39) se presenta la complejidad el proceso paralelo de la sustitución hacia adelante y hacia atrás en el cálculo de la inversa. La complejidad computacional en proceso secuencial es cúbica (n^3)y en el proceso paralelo el cuadrática (n^2). Dividiendo $\frac{n^3}{n^2}$ se obtiene como resultado n. La ganancia del proceso para obtener la inversa es lineal.

La Figura 4.5 muestra el incremento de la ganancia del método de DNSCI (ver Tabla 4.5) con respecto al método de DNSC (ver Tabla 4.4). El comportamiento del incremento de la ganancia (Δg) es lineal y ello se debe a que la ganancia del proceso paralelo en la inversa es lineal. Cada punto en la gráfica de la Figura 4.5 representa un sistema de prueba. El sistema de 14 nodos con 54 variables de estado observa un incremento en la ganancia 0.1, el de 30 nodos con 88 variables de estado de 0.2, el de 57 nodos y 143 variables de estado 0.4 y el sistema de 118 nodos y 357 variables de estado 1.1. Se observa que conforme crece el sistema, la ganancia debido a la inversa de la matriz es mayor y toma un comportamiento lineal.

4.6. Caso de Estudio 5. Resultados del Método DN con Φ Constante

En el caso del sistema de 118 nodos se analizan 3 sistemas modificados (con 379, 415 y 479 variables de estado). El método para encontrar la solución es el método de DN usando la matriz $\mathbf{\Phi}$ constante (DNJ). En la Sección 3.5 se analizó el método de DN con matriz $\mathbf{\Phi}$ constante y se obtuvo la Ecuación 3.24. Se observó una reducción importante en el tiempo debido a que no es necesario repetir el cálculo de la matriz $\mathbf{\Phi}$ y de la inversa de la matriz; ambos procesos son los que mayor tiempo de procesamiento requieren. La Ecuación (3.24) se muestra a continuación:

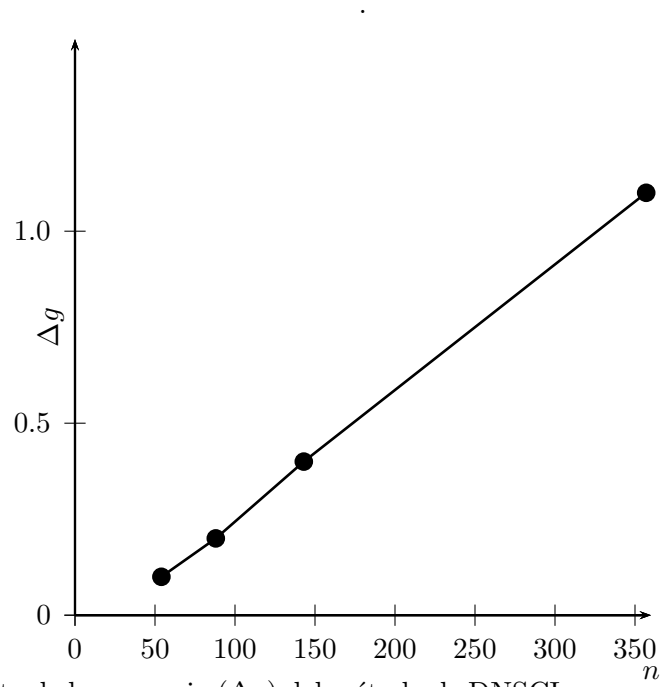


Figura 4.5: Incremento de la ganancia (Δg) del método de DNSCI con respecto al método de DNSC.

$$t_{dnj} = t_p(b + 1 + n + d) + \frac{4}{3}n^3 t_2 \tag{4.4}$$

La Tabla 4.6 muestra la reducción de tiempos del método de DN con **Φ** constante y la ganancia de este método con respecto al método DN tradicional. Las columnas 1 y 2 muestran el número de nodos y variables de estado de los sistemas de prueba del IEEE; la columna 3 el tiempo en que se ejecuta el método DN; la columna 4 los tiempos para procesar el método DNJ y la columna 5 muestra las ganancias del método de DNJ con respecto al método DN.

Los sistemas que se analizaron fueron los de 14 nodos (con 56 variables de estado), 30 nodos (con 94 variables de estado), 57 nodos (con 152 variables de estado) y tres sistemas de 118 nodos (con 379, 415 y 479 variables de estado).

Para llegar a la solución en estado estacionario periódico usando el método DN, se requirió calcular 2 veces la matriz **Φ** en los primeros 5 sistemas de la Tabla 4.6 y 4 veces en el último sistema. Se observa que las ganancias de los primeros 5 sistemas es cercana a 2 y la ganancia del último sistema es superior a 3; esto se debe al número de veces que se calcula la matriz **Φ** en el método DN tradicional.

Tabla 4.6: Reducción en tiempos en el método DNJ

Sistemas de prueba modificados de IEEE	n	DN (ms)	DNJ (ms)	$\frac{DN(ms)}{DNJ(ms)}$
14	56	1117	619	1.80
30	94	2335	1282	1.82
57	152	3124	1637	1.86
118	379	21408	10935	1.96
118	415	39773	20112	1.97
118	479	154113	41855	3.68

4.7. Caso de Estudio 6. Resultados del Método DNJ usando varias GPUs y CPUs

En este caso se analiza la solución del estado Estacionario Periódico de los sistemas de prueba del IEEE modificados que se analizaron en la sección anterior. El método que se utiliza para encontrar la solución es el método DN con $\mathbf{\Phi}$ constante, procesada en varias GPU's y CPU's (DNJC).

En la Sección 3.9.4 se propuso calcular la matriz $\mathbf{\Phi}$ con varias GPU's ya que se pueden usar n^2 elementos para procesarla. Con una GPU difícilmente se tendrá esa cantidad de elementos de proceso que se pueden emplear para procesar la matriz $\mathbf{\Phi}$.

El sistema que se utilizó para hacer las simulaciones es una estación de trabajo Dell precision R5500 con un procesador Intel Xeon E5606, que trabaja a una frecuencia de 2.2 GHz. y una memoria de 6 GB. El procesador Intel Xeon E5606 tiene 4 núcleos. La estación de trabajo tiene tres GPU's con las siguientes características:

- La GPU Nvidia Tesla C2075; tiene 448 núcleos CUDA, 6 GB de memoria y una frecuencia en el procesador de 1.1 GHz. Los 448 núcleos de distribuyen en 14 SM y 32 núcleos CUDA por SM. La GPU Nvidia Tesla C2075 tiene una arquitectura Fermi.

- La GPU Nvidia GeForce GTX 650 Ti; tiene 768 núcleos CUDA, un GB de memoria y una frecuencia de 928 MHz. Los 768 núcleos CUDA se distribuyen en 4 multiprocesadores de flujo

de la arquitectura Kepler (SMX) de 192 núcleos por SMX. La GPU Nvidia geForce GTX 650 Ti tiene una arquitectura Kepler.

- La GPU Nvidia GeForce GTX 660 (OEM) tiene 1152 núcleos CUDA, una memoria de 1.5 GB y una frecuencia de 823 MHz. Los 1152 núcleos CUDA se distribuyen en 6 SMX de 192 núcleos por SMX. La GPU Nvidia geForce GTX 660 (OEM) tiene una arquitectura Kepler.

La estación de trabajo tiene 3 GPU's y el procesador Intel Xeon E5606 tiene 4 núcleos, por lo tanto, un núcleo de la CPU queda libre; los otros 3 núcleos de la CPU están ocupados para atender a las 3 GPU's.

En la Tabla 4.7 se muestra la reducción del tiempo del método DN usando $\mathbf{\Phi}$ constante, 3 GPU's y un núcleo de la CPU para procesarla. En las columnas 1 y 2 se muestra el número de nodos y variables de estado de cada sistema de prueba; en la columna 3 el tiempo requerido por el método DN tradicional; en la columna 4 se presenta el tiempo de proceso del método DNJC y en la columna 5 la ganancia del método DNJC comparado con el método DN.

La ganancia que se presenta en la columna 5 es muy importante; esta ganancia se debe a dos elementos importantes que son:

- Método DN con $\mathbf{\Phi}$ constante; el método obtuvo como resultado ganancias muy cercanas a 2 en los primeros 5 sistemas de prueba y una ganancia de 3.7 en el último sistema de prueba (ver Tabla 4.6).

- Las ganancias de los mismos sistemas de la tabla 4.6 se incrementaron de manera muy importante. Por ejemplo, en el sistema de 118 nodos con 479 variables de estado, la ganancia se incrementó en 30.92 debido al procesamiento paralelo de $\mathbf{\Phi}$ con 3 GPU's y un núcleo de la CPU.

4.8. Caso de Estudio 7. Resultados del Método DN con un Proceso Paralelo en Todas las Partes del Método

En este caso de estudio se consideran los sistemas de prueba modificados del IEEE que se analizaron en la sección anterior. El método que se utiliza para encontrar la solución es el método

Tabla 4.7: Reducción en tiempos en el método DNJC

Sistemas de prueba modificados de IEEE	n	DN (ms)	DNJC (ms)	$\frac{DN(ms)}{DNJC(ms)}$
14	56	1117	181	6.2
30	94	2335	249	9.4
57	152	3124	271	11.5
118	379	21408	1025	20.9
118	415	39773	1436	27.7
118	479	154113	4456	34.6

DN con $\mathbf{\Phi}$ constante y proceso paralelo con tres GPU's y un núcleo para procesar $\mathbf{\Phi}$, procesamiento en paralelo para el método de RK4 para integrar los periodos de tiempo en: los ciclos iniciales, el ciclo base y los ciclos finales y durante el cálculo de la inversa de la matriz (DNJCI).

La Ecuación (3.24) (repetida por conveniencia) muestra el tiempo para ejecutar proceso completo usando procesamiento secuencial.

$$t_{dnj} = bt_p + t_p + nt_p + \frac{4}{3}n^3t_2 + dt_p = t_p(b + 1 + n + d) + \frac{4}{3}n^3t_2$$

La Figura 3.4 muestra el proceso completo del método DN con $\mathbf{\Phi}$ constante.

Asociando la Figura 3.4 con la Ecuación (3.24) y el procesamiento paralelo en cada una de las partes del método de DN con $\mathbf{\Phi}$ constante, se tiene:

- El bloque 1 de la Figura 3.4 procesa los ciclos iniciales y el ciclo base. El tiempo asociado a este bloque en la Ecuación (3.24) es $(b+1)t_p$. En este bloque se procesa en paralelo el método RK4.

- En el bloque 2 de la Figura 3.4 se procesa la matriz $\mathbf{\Phi}$. El tiempo asociado a este bloque en la Ecuación (3.24) es nt_p. En este bloque se procesa en paralelo las columnas de $\mathbf{\Phi}$ y el método de RK4. Se utilizan 3 GPU's y un núcleo de la CPU.

- El bloque 3 de la Figura 3.4 procesa $\mathbf{C} = (\mathbf{I} - \mathbf{\Phi})^{-1}$. El tiempo asociado a este bloque en la Ecuación (3.24) es $\frac{4}{3}n^3t_2$. En este bloque se procesa la sustitución hacia adelante y hacia atrás de forma paralela usando una GPU.

- En el bloque 4 se procesan d periodos de integración y algunas operaciones con tiempo despreciable comparado con el resto del proceso. El tiempo asociado de este bloque con la Ecuación (3.24) es dt_p. En este bloque se procesa el método de RK4 usando una GPU.

En la Tabla 4.8 se muestra la reducción del método de DNJCI, con respecto al método DN. Las columnas 1 y 2 muestran el número de nodos y variables de estado de cada sistema de prueba; la columna 3 muestra el tiempo requerido por el método DN; la columna 4 el tiempo del método DNJCI y la columna 5 la ganancia del método de DNJCI, comparado con el método de DN tradicional. Esta ganancia es muy importante y se observa que incrementa conforme el sistema tenga más variables de estado.

Tabla 4.8: Reducción en tiempos del método DNJCI

Sistemas de prueba modificados de IEEE	n	DN (ms)	DNJCI (ms)	$\frac{DN(ms)}{DNJCI(ms)}$
14	56	1117	192	5.8
30	94	2335	257	9.1
57	152	3124	286	10.9
118	379	21408	956	22.4
118	415	39773	1185	33.6
118	479	154113	2128	72.4

Se observa una diferencia de la columna 5 de la Tabla 4.8 y la columna 5 de la Tabla 4.7, esta diferencia se debe al procesamiento paralelo del método de RK4 en los periodos de integración de los ciclos iniciales, el ciclo base y ciclos finales y al procesamiento paralelo en la inversa de la matriz. Para observar el comportamiento del algoritmo de la Figura 3.4 es necesario considerar el tiempo secuencial y paralelo en cada uno de los bloques; esos tiempos se presentan en la Tabla 4.9.

La Tabla 4.9 presenta en la columna 1 el número de nodos de cada sistema de prueba; la columna 2 el número de variables de estado (n); la columna 3 el número de ciclos finales d; las columnas 4 y 5 el tiempo de los ciclos iniciales y ciclo base en forma secuencial y paralelo, respectivamente; las columnas 6 y 7 el tiempo de proceso para obtener $\mathbf{\Phi}$ en forma secuencial y paralela, respectivamente; las columnas 8 y 9 el tiempo de procesamiento de la inversa en forma

secuencial y paralela, respectivamente y finalmente, las columnas 10 y 11 el tiempo para procesar los ciclos finales de forma secuencial y paralela, respectivamente.

Tabla 4.9: Tiempo secuencial y paralelo de los cuatro bloques del algoritmo de la Figura 3.4

Nodos	n	d	Ciclos Iniciales		Matriz $\mathbf{\Phi}$		Inversa		Ciclos finales	
			Sec (ms)	Par (ms)	Sec (ms)	Par (ms)	Sec (ms)	Par (ms)	Sec (ms)	Par (ms)
14	56	3	75	124	514	27	2	2	28	39
30	94	7	98	136	1093	37	9	6	82	78
57	152	4	83	123	1477	89	37	19	40	55
118	379	3	220	211	10057	467	575	227	83	51
118	415	3	369	251	18825	583	780	298	138	53
118	479	32	630	275	37274	777	1202	447	2716	629

Las diferencias en los tiempos en secuencial y paralelo se explican a continuación:

El bloque que presenta mayor reducción en el tiempos es el que calcula la matriz $\mathbf{\Phi}$. Este bloque aún puede ver mayor reducción si se tienen más elementos de proceso, ya que pueden utilizarse n^2. Cabe señalar que la reducción lograda es muy importante (de 47.97 veces en el sistema de 118 nodos con 479 variables de estado) a pesar que el proceso secuencial se realiza con un elemento de proceso que trabaja a mayor frecuencia que los elementos de proceso que tiene la GPU.

El siguiente bloque muestra una reducción en el tiempo es el que procesa la inversa de la matriz (bloque 3). En la Figura 2.4 se observó que el tiempo para obtener la inversa de hasta 500 variables de estado es muy pequeño (de aproximadamente 1 segundo) comparado con el tiempo para procesar la matriz $\mathbf{\Phi}$ (de aproximadamente 28 segundos). Sin embargo, cuando la matriz $\mathbf{\Phi}$ se procesa en paralelo, el tiempo para obtener la inversa es mayor que el tiempo para procesar la matriz $\mathbf{\Phi}$ (en promedio de 1.37 veces mayor en los sistemas de 118 nodos, al comparar las columnas 7 y 8 de la Tabla 4.9).

En el proceso para calcular los ciclos iniciales y el ciclo base, hay una reducción en el tiempo de

ejecución en los sistemas de 118 nodos y un incremento en el resto de los sistemas; el incremento en el tiempo se debe a que las GPU's necesitan un tiempo cercano a 100 milisegundos para inicializarse. Cuando el procesamiento secuencial de los ciclos iniciales y ciclo base es inferior a 100 milisegundos, no tiene sentido procesarlos con con una GPU.

En los ciclos finales la mayor reducción en el tiempo de proceso se observa en los sistemas con mayor número de variables de estado. En particular el sistema de 118 nodos y 479 variables de estado. Esta reducción se debe al mayor número de ciclos finales, que es de 32. La reducción en tiempo en los ciclos finales se debe exclusivamente al método de RK4 procesado en paralelo.

La Tabla 4.10 compara los tiempos obtenidos de los métodos DNJCI y fuerza bruta. Las columnas 1 y 2 muestran los sistemas de prueba y las variables de estado de cada sistema, respectivamente. La columna 3 tiene los tiempos del método de fuerza bruta; la columna 4 los tiempos del método DNJCI y la columna 5 la ganancia del método DNJCI comparado con el método fuerza bruta.

El sistema que llama la atención es el de 30 nodos con 94 variables de estado; la ganancia en este sistema es de 1355.60 comparado con el método fuerza bruta. El otro sistema que también ve una ganancia importante es el de 118 nodos con 415 variables de estado y es de 139.48. Cuando un sistema llega al estado estacionario periódico de forma rápida usando el método de fuerza bruta significa que el sistema está muy amortiguado y cuando el sistema no llega de forma rápida a la solución significa que tiene poco amortiguamiento. De acuerdo a lo anterior, el sistema de 57 nodos con 152 variables de estado es un sistema muy amortiguado, ya que la solución con el método de fuerza bruta es mejor que el método DN. El sistema de 30 nodos con 94 variables de estado debe ser un sistema poco amortiguado debido a la cantidad de tiempo que requiere para llegar a la solución.

4.9. Conclusiones

En este capítulo se presentaron, mediante siete casos de estudio, los resultados de los tiempos de ejecución obtenidos con las propuestas variantes de solución al método DN, reportadas en el Capítulo 3, usando la estación de trabajo con la arquitectura que se propuso en la Sección 3.8. En todas las propuestas hay reducciones en los tiempos de ejecución. En la propuesta del método

Tabla 4.10: Reducción en tiempos del método DNJCI comparado con fuerza bruta

Sistemas de prueba modificados de IEEE	n	Fuerza bruta (ms)	DNJCI (ms)	$\frac{DN(ms)}{DNJCI(ms)}$
14	56	930	192	4.84
30	94	348388	257	1355.60
57	152	776	286	2.71
118	379	52058	956	54.45
118	415	165282	1185	139.48
118	479	291505	2128	90.00

DNS, el promedio de reducción es 1.61 veces; en el método DNC es de 4.35 veces; en el método DNSC es de 5.6; en el método DNSCI es de 6.05. En el método DNJ la reducción en los tiempos depende de la evaluación repetitiva de $\mathbf{\Phi}$, su reducción se aproxima al número de veces que se calcula. En el método DNJC, el promedio de reducción en los tiempos en los sistemas de 118 nodos es 27.7 veces y en los otros sistemas de 9.0 veces. En el método DNSCI, el promedio de reducción en los tiempos de ejecución es de 42.8 veces en los sistemas de 118 nodos y 8.6 veces los otros sistemas.

Los mejores resultados se obtuvieron con el método de DN usando la matriz de identificación constante y procesamiento paralelo en la mayor parte del proceso. Para utilizar este método se requiere un equipo con una configuración con GPUs. Con respecto a la evaluación repetitiva de $\mathbf{\Phi}$ y procesamiento secuencial, en particular el sistema de 118 nodos con 479 variables de estado requiere 72.4 veces menos tiempo de proceso.

Se comprobó que se pueden reducir los tiempos de ejecución de cada una de las partes de los algoritmos. La parte que mayor reducción observa es el proceso de cálculo para obtener la matriz de identificación. La reducción tan importante en el tiempo de ejecución de la matriz de identificación (en promedio del orden 33.9 veces en los sistemas de 118 nodos y 21.7 veces en los otros sistemas de prueba) se debe al doble procesamiento en paralelo utilizado, que es el cálculo de la columnas de forma simultánea y proceso interno (usando el método de RK4 en paralelo) en el cálculo de cada columna usando procesamiento paralelo. Otra de las partes del método DN en donde se reduce el tiempo de ejecución es la propuesta de procesamiento en paralelo de la inversa de

la matriz; la reducción de tiempo en el proceso de la inversa tiene un comportamiento lineal, siendo el sistema de 118 nodos con 479 variables de estado el que mayor reducción de tiempo observa y es de 2.69 veces con respecto al proceso secuencial. En los ciclos finales del método de DN con la matriz de identificación constante se observa una reducción importante en tiempo (especialmente en el sistema de 118 nodos con 479 variables de estado, en donde es aproximadamente 4.32), esta reducción en el tiempo se debe exclusivamente a la propuesta del método RK4 procesado en paralelo.

Se mostró la solución en estado estacionario para uno de los casos de estudio, en términos de variables de estado distorsionadas, su contenido armónico y evolución de su convergencia al ciclo límite. Esta información es importante para determinar el grado de penetración de armónicos en la red eléctrica, así como el cumplimiento de normas establecidas para la operación segura del sistema eléctrico, tal como lo indica [IEEE-519 Std. 1992]. Las gráficas que se presentan son representativas de dos sistemas únicamente, aunque pueden obtenerse para los demás considerados.

Capítulo 5

Conclusiones

5.1. Conclusiones Generales

Esta tesis ha reportado la experiencia y resultados obtenidos para la determinación eficiente de la solución en estado estacionario de sistemas eléctricos en el dominio del tiempo. Se ha tomado como referencia el método Newton basado en Diferenciación Numérica [Semlyen y Medina, 1995], consolidado actualmente en la literatura abierta para este propósito.

Se trabajó también a nivel de detalle en los campos de análisis de complejidad del método DN y procesamiento en paralelo. Es este último considerando el diseño y arquitectura de distintas configuraciones y plataformas.

Se ha contribuido al estado del arte presente sobre solución eficiente de redes eléctricas en estado estacionario periódico, análisis de complejidad y aplicación de tecnología de procesamiento en paralelo.

A continuación se detalla de manera concisa la experiencia recabada en cada uno de los aspectos mencionados:

En esta tesis se realizó un análisis de complejidad computacional al método DN que se introdujo en [Semlyen y Medina, 1995] y en base en ella se realizaron propuestas para mejorar el tiempo de solución del método DN.

En las metodologías propuestas DN selectiva y DN usando la matriz de identificación constante se logró reducir el esfuerzo computacional. Las metodologías en las que se logró distribuir

el esfuerzo computacional entre los diferentes elementos de proceso fueron en el procesamiento paralelo basado en CUDA para hallar la matriz de identificación, en el proceso para obtener la inversa de una matriz y en el proceso para obtener los periodos de integración que se requieren durante el proceso iterativo del método DN en sus distintas versiones.

La ganancia (que mide de forma cuantitativa la reducción de los tiempos) en el método DN selectivo fue importante a analizar. Se observó que al aplicar el método DN selectivo casi siempre hay una reducción de proceso, con respecto al método DN tradicional. En el único caso en el que no se obtuvo ganancia fue cuando el método DN selectivo calcula todas las columnas de $\mathbf{\Phi}$ en cada evaluación repetitiva. En ningún caso el método DN selectivo es peor que el método de DN.

Se observó que el método DN con matriz de identificación constante permite reducir de forma significativa el esfuerzo computacional y con ello el tiempo de solución. El método reduce el tiempo de ejecución aproximadamente el número de veces que se tiene que calcular la matriz de identificación en el método de DN; es decir, si la matriz de identificación se calcula 2 veces, se reduce el tiempo del proceso aproximadamente 2 veces. La reducción en el tiempo no es igual al número de veces que se calcula la matriz de identificación debido a que requiere un proceso que implica el cálculo de los ciclos iniciales y el ciclo base antes del cálculo de la matriz de identificación y de ciclos adicionales, posterior a su cálculo.

Los métodos DN selectivo y DN con matriz de identificación constante reducen el esfuerzo computacional, pero la mayor reducción en el tiempo de solución se observa cuando a estas dos metodologías se les aplica procesamiento paralelo especialmente en el cálculo de la matriz de identificación.

El método de Runge Kutta de cuarto orden procesado en paralelo, idealmente puede reducir en n el tiempo para calcular un periodo de integración. Esa reducción no se observa en esa proporción en los sistemas de prueba que se reportaron debido a que no se tienen suficientes elementos de proceso y a que la comparación se realiza con elementos de proceso trabajando a diferentes frecuencias; aún así se logró reducir el tiempo de ejecución en los ciclos iniciales y ciclo base (en el orden de 1.60 en los sistemas de 118 nodos) y en los ciclos finales (en el orden de 2.85) en el método DNJCI.

La inversa de la matriz es un elemento que en otros estudios no se había considerado tan importante debido a que el tiempo en su ejecución es demasiado pequeño, comparado con el

tiempo para encontrar la matriz de identificación de forma secuencial. El análisis de complejidad computacional del algoritmo DN permitió observar que cuando los sistemas son muy pequeños, el tiempo requerido por el cálculo de la inversa de la matriz es muy pequeño, pero a medida que el orden del sistema se incrementa, el tiempo para calcular la inversa de la matriz toma mayor importancia llegando a ser el más importante, ya que tiene complejidad computacional cúbica.

En esta tesis se logró reducir el tiempo para encontrar la inversa al utilizar procesamiento paralelo basado en GPUs en parte de su proceso. La complejidad sigue siendo cúbica, pero ahora el coeficiente de la variable que está al cubo fue $\frac{1}{3}$ en lugar de $\frac{4}{3}$, lo cual significa que la ganancia es de 4 en esta parte del proceso.

La reducción más considerable en el tiempo de proceso para obtener la solución en estado estacionario periódico se logró al aplicar a la matriz de identificación constante procesamiento paralelo para calcular sus columnas, procesamiento paralelo al método de Runge Kutta de 4to. orden y al cálculo de la la inversa de la matriz. En los resultados se reporta una ganancia de 72.4 en el sistema de prueba de IEEE de 118 nodos modificado con 479 variables de estado. Es importante mencionar que esta importante ganancia se logra a pesar de que los núcleos CUDA trabajan a menor frecuencia en el reloj que la frecuencia del reloj de la CPU.

La configuración realizada del equipo de cómputo con GPUs y CPUs, permitió lograr las reducciones en tiempo de proceso reportadas. El equipo de cómputo configurado fue de 2368 núcleos CUDA (en 3 GPUs) y 4 núcleos de una CPU. Este número de núcleos (elementos de proceso) es menor al número de elementos de proceso que se pueden usar en la parte del proceso que calcula la matriz de identificación. El equipo fue suficiente para mostrar las metodologías. Sin embargo, es posible reducir aún más los tiempos de proceso debido a que se pueden usar n^2 elementos de proceso.

5.2. Trabajos Futuros

Tomando como referencia lo reportado en esta investigación, se propone continuar con el trabajo de investigación en las siguientes direcciones:

1. Para mejorar las técnicas que propusieron en esta tesis, es importante incorporar técnicas de matrices dispersas para obtener la inversa de la matriz en sistemas con gran cantidad

de variables de estado, debido que este proceso puede requerir de un considerable esfuerzo computacional.

2. En cada una de las propuestas se indicó su complejidad computacional; esta complejidad computacional está en función de algunas variables que dependen del tiempo. Las ecuaciones de la complejidad quedan en función del tiempo debido a que algunas ecuaciones diferenciales ordinarias (que modelan el sistema eléctrico) dependen de la conectividad del sistema y de la complejidad del modelo. Es importante hacer un estudio que permita determinar el grado de conectividad del sistema o del grado de complejidad de cada ecuación y con ello evitar el uso del tiempo en la ecuación de complejidad.

3. Los resultados que se presentaron en los casos de estudio se realizaron con una programación de GPUs usando la arquitectura de GPUs Fermi. Al terminar la tesis, la arquitectura de las GPUs mas reciente es la arquitectura Maxwell. Es importante hacer pruebas en esta arquitectura para mejorar el tiempo de solución.

4. En las ecuaciones de complejidad de las propuestas del método de diferenciación numérica hay un término que está asociado a la método de Runge Kutta de cuarto orden; este es $4_t1 + 3t_2 + t_3$. Al usar otro método de integración la expresión de complejidad computacional estará asociada al método utilizado. Un estudio interesante sería probar diferentes métodos de integración que podrán incluirse métodos de Runge Kutta de orden superior.

5. La cantidad n^2 elementos de proceso que se pueden usar de forma simultánea en las diferentes propuestas del método DN es mayor a los elementos de proceso que hay en las mejores super-computadoras actuales y en las mejores GPUs actuales. Un estudio que puede ser interesante para efectuar el procesamiento computacional es el uso de FPGA (Field Programmable Gate Array).

6. En las aplicaciones de sistemas eléctricos, es importante agregar otros modelos, por ejemplo agregar estudios con fuentes renovables de energía.

7. Con la herramienta computacional utilizada puede analizarse su comportamiento dinámico y realizarse estudios en tiempo real, mediante su interacción con simuladores desarrollados para tal fin, tal como Opal RT, entre otros.

8. Incorporación de modelos más detallados de cargas y componentes eléctricos, tales como líneas de transmisión con parámetros distribuidos, modelos de máquinas de inducción doblemente alimentada, entre otros.

9. Tomando como referencia la solución periódica en estado estacionario periódico, aplicar una técnica selectiva de mitigación de armónicos en nodos críticos del sistema que rebasen los límites permisibles por normas establecidas, tal como [IEEE-519 Std. 1992].

Apéndice A

Plataforma de Programación OpenMP

A.1. Introducción

OpenMP (Open Multi-Processing) es una interfaz de programación de aplicaciones (Application Programming Interface (API)) para la programación de varios procesos usando una memoria compartida. Permite la programación en paralelo usando diferentes Lenguajes de programación. En [OpenMP] se muestra la información de los lenguajes de programación y los sistemas operativos que son soportados por OpenMP. Entre los lenguajes en donde se puede usar OpenMp están los lenguajes C, C++ y fortran.

En este apéndice se tratarán las herramientas de OpenMP que se utilizaron en la tesis. El resto de la información de OpenMP la puede encontrar en su sitio oficial (ver [OpenMP]). En esta tesis OpenMP sirvió para activar 3 hilos de forma paralela; cada hilo activó una GPU y le envió información, después recibió la información de las GPUs y la juntó. El trabajo que se realizó en OpenMP fue muy importante, pero no se ocuparon todas las herramientas que ofrece (no se requrían). En el sitio ÏEEE Xplore"hay 1182 trabajos cientificos relacionados con OpenMP, por lo que es de suponer que OpenMP ofrece elementos importantes de procesamiento paralelo para resolver aplicaciones de diferentes áreas.

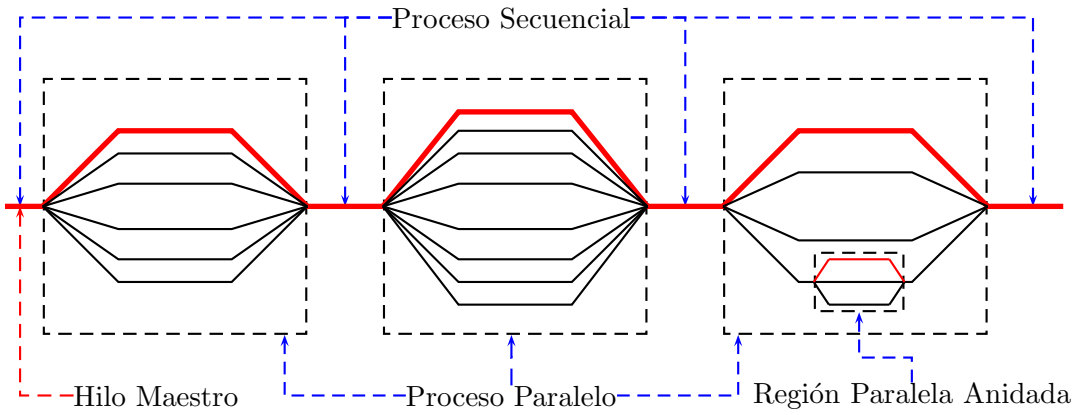


Figura A.1: Modelo Fork-Join de la programación de OpenMP.

A.2. Módelo de Programación de OpenMP

En esta sección se presenta el modelo de OpenMP y las rutinas para generar regiones paralelas en OpemMP. También se presenta la conexión de OpenMP con CUDA.

A.2.1. Modelo Fork-Join

OpenMP se basa en el modelo Fork-join, donde una tarea que requiere un esfuerzo computacional importante puede dividirse en k hilos (fork) con menor carga de procesamiento, para luego unir (join) los resultados de cada hilo en un resultado.

En la Figura A.1 se muestra el modelo Fork-join, en él se muestra que el hilo maestro comienza el proceso secuencial, luego se activan los hilos para resolver una tarea en un proceso paralelo. Al finalizar el proceso paralelo se juntan los resultados para seguir con un proceso secuencial (con el hilo maestro). El proceso sigue con bloques que se procesan de forma secuencial seguido de bloques que se ejecutan de forma paralela hasta terminar la aplicación. En muchas aplicaciones es posible encontrarse con regiones paralelas anidadas; las regiones paralelas anidadas son regiones paralelas dentro de una región que se puede procesar de forma paralela.

A.2.2. Construcciones de Regiones Paralelas en OpenMP

Para construir regiones paralelas en OpenMP es muy simple. Se puede usar la el código «#pragma omp parallel». En el código siguiente se muestra un ejemplo para generar una región paralela usando 4 hilos.

```
1.    #include "omp.h"
2.    int main(){
3.        variables globales
4.        #pragma omp parallel num_threads(4)
5.        {
6.           int identificador=omp_get_thread_num();
7.           funcion-paralela(identificador, otros-parametros);
8.        }
9.    }
```

- En la línea 1 se carga la librería «omp.h» y es en donde están todos los elementos de OpenMP.
- En la línea 2 inicia el programa de C
- En la línea 3 se definen las variables del programa. Las variables que se definen antes de la región paralela pueden ser accesados por todos los hilos del programa.
- En la línea 4 se define la región paralela para ejecutarse con 4 hilos. La región paralela se define de la línea 5 a la 8.
- En la linea 6 se identifica el hilo que ejecutará el resto de la región paralela que le corresponde. La variable $identificador$ tomará los valores 0, 1, 2 o 3.
- En la línea 7, cada hilo ejecutará la función «función-paralela». La función «función-paralela» recibirá como argumento el identificador del hilo que la ejecutará y la información que procesará la función.
- En la línea 8 termina la región paralela y ningún hilo sigue a la siguiente línea hasta que todos terminen hasta este punto.

A.2.3. OpenMP con Cuda

En esta sección se presenta una forma de unir OpenMP y CUDA. La unión que se explicará fue la que se usó en esta tesis debido a que las arquitecturas de las tarjetas gráficas que se utilizaron son la Fermi y Kepler de la primera generación; estas tarjetas gráficas no permiten la programación dinámica de las GPUs.

Por lo anterior, un hilo generado por OpenMP (que es un núcleo de la CPU) puede activar a una GPU y hacerla trabajar de forma paralela con otras GPUs. A continuación se presenta un ejemplo de la conexión de OpenMP con CUDA.

```
1.      #include <omp.h>
2.      #include <cuda.h>
3.      __global__ void función-cuda(parametros){
4.           cuerpo de cuda
5.      }
6.      función-paralela(int identificador, otros parametros){
7.           cudaSetDevice(identificador);
8.           Define variables de cuda
9.           copia informacion de la CPU a la GPU
10.          función-cuda<<<4,7>>>(parametros);
11.          copia información de la GPU a la CPU
12.          resto del proceso
13.     }
14.     int main(){
15.          define variables
16.          #pragma omp parallel num_threads(4)
17.          {
18.               int identificador=omp_get_thread_num();
19.               funcion-paralela(identificador, otros-parametros);
20.          }
21.     }
```

- La línea 1 carga la librería de OpenMP al programa de C
- La línea 2 carga la librería de CUDA al programa de C
- En la linea 14 comienza la ejecución del programa de C
- En la línea 15 se definen las variables que usarán todos los hilos generados por OpenMP (todos los núcleos de la CPU).
- En la línea 16 comienza el proceso paralelo en la CPU con los hilos indicados (en este caso 4).
- En la línea 18 se identifica el número de hilo que procesará la función que está en la línea 19.
- La línea 19 ejecuta la función «función-paralela» por cada hilo (núcleo de la CPU) generado por OpenMP.
- En la línea 6 se define la función «función-paralela»; esta función se ejecutará por cada hilo generado por OpenMP, la función recibe como parámetro el identificador del hilo y otros parámetros que ocupará la función. En la función se realiza lo siguiente:
  - La línea 7 activará la GPU con el número hilo generado por OpenMP (el hilo cero activa la GPU cero). Para que el programa no tenga problemas se deben activar únicamente las GPUs que se tienen disponibles. Cuando hay mas núcleos de la CPU que GPUs

conectadas se pueden hacer trabajar las GPUs para hacer un proceso y los núcleos sobrantes de la CPU para hacer otro.

  - La linea 8 definirá las variables globales (de cada GPU) que ocuparán todos los hilos de todos los bloques de CUDA. Las variables que usan memoria compartida entre los hilos de un bloque se definen en el cuerpo del kernel.
  - En la línea 9 se realiza la copia de la información que requiere la GPU para trabajar.
  - En la 10 se ejecuta el kernel en cada una de las GPUs. El kernel se ejecuta con 4 bloques de 7 hilos (ver [CUDA,2015] para ver el manejo de bloques de hilos en CUDA).
  - En la línea 11 se hace una copia de la información que procesó la GPU a la memoria de la CPU.
  - En la línea 12 continúa el proceso paralelo con los hilos de OpenMP.

- En la línea 3 se define una función de cuda (kernel) que se ejecutará en la GPU. En el cuerpo de la función (línea 4) de CUDA van todos los elementos que se pueden usar (memoria global, compartida, local, hilos, bloques, sincronización, etc.) para realizar una tarea. El kernel termina en la línea 6.

Referencias

[Acha, 1988] , *E. Acha, Modelling of Power System Transformers in the Complex Conjugate Harmonic Space, PhD Thesis, University of Canterbury, New Zealand, 1988.*

[Acha, 1989] , *E. Acha, J. Arrillaga, A. Medina, and A. Semlyen, General frame of reference for analysis of harmonic distortion in systems with multiple transformer nonlinearities, Proceedings IEE, Pt. C, vol. 136, no. 5, Sept. 1989, pp. 271-278.*

[Aprille y Trick, 1972] , *Aprille, T.J. and Trick, Timothy N. Steady-state analysis of nonlinear circuits with periodic inputs Proceedings of the IEEE, pp. 108 - 114, 1972.*

[Arrillaga, 1995] , *J. Arrillaga, A. Medina, M. L. V. Lisboa, M. A. Cavia, and P. Sánchez, The harmonic domain. A frame of reference for power system harmonic analysis, IEEE Trans. on Power Systems, vol. 10, no. 1, Feb. 1995, pp. 433-440.*

[Arrillaga, 2004] , *J. Arrillaga, N. R. Watson, and G. N. Bathurst, A multifrequency power flow of general applicability, IEEE Trans. on Power Delivery, vol. 19, no. 1, Jan. 2004, pp. 342-349.*

[Bathurst, 2000] , *G. N. Bathurst, B. C. Smith, N. R. Watson, and J. Arrillaga, A modular approach to the solution of the three-phase harmonic flow, IEEE Trans. on Power Delivery, vol. 15, no. 3, July 2000, pp. 984-989.*

[Chapra, 2006] , *Steven C. Chapra and Raymond P. Canale, Métodos Numéricos para Ingenieros, Quinta Edición, Editorial McGraw-Hill Interamericana, 2016.*

[Chua y Ushida, 1981] . *Chua L.O. y Ushida A. .ªlgorithms for Computing Almost Periodic Steady-State Response of Nonlinear Systems to Multiple Input Frequencies", IEEE Transactions on Circuits and Systems, Vol. 28, No. 10, pp. 953-971, Octubre 1981.*

[Callaghan, 1990] , *C. D. Callaghan and J. Arrillaga, Convergence criteria for iterative harmonic analysis and its application to static convertors, Proc. 1990 IEEE/ICHPS IV International Conference on Harmonics in Power Systems, Budapest, Hungary, October 4-6, pp. 38-43.*

[Callaghan, 1989] , *C. D. Callaghan and J. Arrillaga, A double iterative algorithm for iterative harmonic analysis and harmonic flows at ac-dc terminals, Proc. of the IEE, vol. 136, no. 6, 1989, pp. 319-324.*

[CUDA,2015] , *www.nvidia.com,CUDA C PROGRAMMING GUIDE, Design Guide,2015.*

[Debnath, 2011] , *Debnath, J.K.; Wai-Keung Fung; Gole, A.M.; Filizadeh, S., Electromagnetic transient simulation of large-scale electrical power networks using graphics processing units, Electrical & Computer Engineering (CCECE) ,25th. IEEE Canadian Conference on. 2012.*

[Demsem, 1984] , *T. J. Demsem, P. S. Bodger, and J. Arrillaga, Three phase transmission system modelling for harmonic penetration studies, IEEE Trans. on Power Apparatus and Systems, vol. PAS-103, no. 2, Feb. 1984, pp. 310-317.*

[Dommel, 1986] , *H. W. Dommel, A Yan, and S. Wei, ´Harmonics from transformer saturation,IEEE Trans. on Power Systems, vol. PWRD-1, no. 2, Apr. 1986, pp. 209-214.*

[Fermi,2009] , *NVIDIA, http://www.nvidia.co.uk/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Arc*

[Foster,1994] , *Foster I. Designing and Building Parallel Programs. Addison Wesley 1994.*

[García-Olmos, 2013] , *García N., Olmos R.C., GPU-Accelerated Poincaré Map Method for Harmonic-Oriented Analyses of Power Systems Power and Energy Society General Meeting (PES), 2013 IEEE, 2013.*

[García, 2001] , *N. García, E. Acha, A. Medina, "Swift Time Domain Solutions of Electric Systems Using Parallel Processing", Proceedings of the Sixth IASTED International Conference, Rhodes, Greece, pp. 172-177. 2001.*

[García y Medina, 2003] , *Garcia, N. and Medina, A.Swift time domain solution of electric systems including SVSs, Power Delivery, IEEE Transactions on, Vol. 18, pp. 921 - 927, 2003.*

[García, 2005] , *N. García, "SParallel Harmonic-Oriented Method for Large-Scale Electric Systems Based on MPI Programming and the Limit Cycle Method", Power Tech, 2005 IEEE Russia.*

[He-K.,2015] , *GPU-Accelerated Parallel Sparse LU Factorization Method for Fast Circuit Analysis, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 2015.*

[IEEE-754 Std.1985] , *754-1985 - IEEE Standard for Binary Floating-Point Arit hmetic.*

[IEEE-754 Std. 2008] , *https://standards.ieee.org/findstds/standard/754-2008.html.*

[IEEE-519 Std. 1992] , *Recommended Practices and Requirements for Harmonic Control in Electric Power Systems. IEEE Press. 1993 H. Akagi, New trends in active filter for power conditioning, IEEE Trans. Ind. Applicat., vol 32, pp. 1312-1322, 1996.*

[Intel-E5606] , *http://ark.intel.com/es/products/52583/Intel-Xeon-Processor-E5606-8M-Cache-2_13-GHz-4_80-GTs-Intel-QPI.*

[Jalili, 2012] , *Jalili-Marandi, V.; Zhiyin Zhou; Dinavahi, V. Large-Scale Transient Stability Simulation of Electrical Power Systems on Parallel GPUs Parallel and Distributed Systems, IEEE Transactions on, pp. 1255-1266. 2012.*

[Mahmoud, 1982] , *A. A. Mahmoud and R. D. Schultz, A method for analyzing harmonic distribution in a.c. power systems, IEEE Trans. on Power Apparatus and Systems, vol. PAS-101, no. 6, 1982, pp. 1815-1824.*

[Maggioni, 2013] , *Maggioni, M.; Berger-Wolf, T.; Jie Liang, GPU-Based Steady-State Solution of the Chemical Master Equation Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International, pp. 579 - 588, 2013.*

[Magaña y Medina,2015] , *Ernesto Magaña, Aurelio Medina and Antonio Ramos, Periodic Steady State Solution of Power Systems by Selective Transition Matrix Identification, LU Decomposition and Graphic Processing Units, Power and Energy Society General Meeting (PES), 2015 IEEE, 2015.*

[Medina et. al 2013] , *A. Medina, A. Medina, J. Segundo, P. Ribeiro, W. Xu, K. L. Lian, G.*

W. Chang, V. Dinavahi, N. R.Watson, Harmonic Analysis in Frequency and Time-Domain, Power Delivery, IEEE Transactions on, Vol. 28, p.p. 1813-1821.

[Medina y Ramos, 2003] , *A. Medina, A. Ramos-Paz, and C. R. Fuerte-Esquivel, Periodic Steady State Solution of Electric Systems With Nonlinear Components Using Parallel Processing, IEEE TRANSACTIONS ON POWER SYSTEMS, VOL. 18, NO. 2, MAY 2003.*

[Medina y García, 2004] , *Medina A. y García N. "Fast time domain computation of the periodic steady-state of systems with nonlinear and time-varying components", Electrical Power and Energy Systems, Vol. 26, Págs. 637-643, 2004.*

[Neic,2012] , *Neic, A.; Karl Franzens; Liebmann, M. ; Hoetzl, E. ; Mitchell, L. Accelerating Cardiac Bidomain Simulations Using Graphics Processing Units, Biomedical Engineering, IEEE Transactions on, p.p. 2281 - 2290, 2012*

[Nvidia, 2009] , *NVIDIA, Cuda Compute Arquitecture Fermi. 2009*

[Nvidia, 2012] , *NVIDIA, Cuda Compute Arquitecture Kepler. 2014*

[Nvidia, 2015] , *NVIDIA, Tuning cuda applications for maxwell. 2015*

[Nvidia] , *http://la.nvidia.com/page/corporate_timeline.html*

[Nvidia-93-2016] , *https://nvidia.com*

[Nvidia-tesla-c2075] , *http://www.nvidia.com.mx/docs/IO/43395/NV-DS-Tesla-C2075.pdf*

[Nvidia-GTX-660] , *http://www.nvidia.com.mx/object/geforce-gtx-660-oem-la.html.*

[Nvidia-GTX-650] , *http://www.nvidia.com.mx/object/geforce-gtx-650ti-la.html.*

[OpenMP-2002] , *http://www.openmp.org/mp-documents/cspec20.pdf.*

[OpenMP] , *http://openmp.org/wp/openmp-compilers/*

[Openmp] , *Referencia en línea, http://openmp.org*

[Parker y Chua et. al 1989] , *Parker T.S., Chua L.O. "Practical Numerical Algorithms for Chaotic Systems", Springer-Verlag, 1989.*

[Peña y Medina,2010] , *Rafael Peña, and Aurelio Medina, Fast Steady-State Parallel Solution of the Induction Machine, XIX International Conference on Electrical Machines - ICEM 2010, Rome.*

[Precision-R5500] , *http://topics-cdn.dell.com/pdf/precision-r5500_Service %20Manual_en-us.pdf*

[Rodríguez y Medina, 2004] , *Rodriguez, O. and Medina, A. Efficient methodology for the transient and periodic steady-State analysis of the synchronous Machine using a phase coordinates model Energy Conversion, IEEE Transactions on, Vol. 19 , p.p. 464-466, 2004.*

[Sharma, 1991] , *V. Sharma, R. J. Fleming, and L. Niekamp, An iterative approach for analysis of harmonic penetration in power transmission networks, IEEE Trans. on Power Delivery, vol. 6, no. 4, Oct. 1991, pp. 1698-1706.*

[Segundo y Medina, 2008] , *Segundo, J y Medina, A, Periodic Steady-State Solution of Electric Systems Including UPFCs by Extrapolation to the Limit Cycle Power Delivery, IEEE Transactions on, Vol. 23 , P.P. 1506 - 1512, 2008.*

[Segundo y Medina, 2009] , *Juan Segundo, Aurelio Medina , "Modeling of Electric Power Systems including FACTS and Custom Power Devices and Their Stability Analysis Through Application of Bifurcation Theory, Continuation Methods, and Newton Methods to Compute the Periodic Steady State", Tesis de Doctorado de Juan Segundo, 2009.*

[Segundo y Medina, 2010] , *J. Segundo-Ramírez and A. Medina, Computation of the Steady-State Solution of Nonlinear Power Systems by Extrapolation to the Limit Cycle Using a Discrete Exponential Expansion Method, International Journal of Nonlinear Sciences and Numerical Simulation., vol. 11, no. 8, pp. 655-660, 2010.*

[Segundo y Medina, 2010a] , *J. Segundo-Ramírez and A. Medina, An Enhanced Process for the Fast Periodic Steady State Solution of Nonlinear Systems by Poincaré Map and Extrapolation to Limit Cycle, International Journal of Nonlinear Sciences and Numerical Simulation., vol. 11, no. 8, pp. 655-660, 2010.*

[Semlyen y Medina, 1995] , *A. Semlyen and A. Medina, "Computation of the periodic steady state*

in systems with nonlinear components using a hybrid time and frequency domain methodology , IEEE Transactions on Power Systems, vol. 10, No. 3, pp. 1498-1504, Aug. 1995.

[Top-500,2015] , *http://www.top500.org/*

[Vahid-Zhiyin y Venkata, 2012] , *Vahid Jalili-Marandi, Zhiyin Zhou and and Venkata Dinavahi, "Large-Scale Transient Stability Simulation of Electrical Power Systems on Parallel GPUs", Parallel and Distributed Systems, IEEE Transactions on, pp1255 - 1266. 2012.*

[Xiaoming, 2015] , *Xiaoming Chen; Ling Ren; Yu Wang; Huazhong Yang, GPU-Accelerated Sparse LU Factorization for Circuit Simulation with Performance Modeling Parallel and Distributed Systems, IEEE Transactions on, p.p. 786-795, 2015.*

[Xia, 1982] , *D. Xia and G. T. Heydt, ´Harmonic power flow studies, part I formulation and solution, part II implementation and practical application, IEEE Trans. on Power Apparatus and Systems, vol. PAS-101, June 1982, pp. 1257-1270.*

[Xu, 1991] , *W. Xu, J. R. Marti, and H. W. Dommel ´A multiphase harmonic load flow solution technique,IEEE Trans. on Power Systems, vol. PS-6, Feb. 1991, pp. 174-182.*