UNIVERSIDAD MICHOACANA DE
SAN NICOLÁS DE HIDALGO

DIVISIÓN DE ESTUDIOS DE POSGRADO
FACULTAD DE INGENIERÍA ELÉCTRICA

"APLICACIÓN DE UNIDADES DE PROCESAMIENTO GRÁFICO EN LA DETERMINACIÓN DEL ESTADO ESTACIONARIO PERIÓDICO DE REDES ELÉCTRICAS"

TESIS

PARA OBTENER EL GRADO DE
MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA

PRESENTA
ING. VÍCTOR HUGO MONTESINOS GONZÁLEZ

DIRECTOR DE TESIS
DR. EN ING. ELÉCTRICA
ANTONIO RAMOS PAZ

MORELIA, MICHOACÁN AGOSTO DEL 2013

UNIVERSIDAD MICHOACANA DE
SAN NICOLÁS DE HIDALGO

DIVISIÓN DE ESTUDIOS DE POSGRADO
FACULTAD DE INGENIERÍA ELÉCTRICA

"APLICACIÓN DE UNIDADES DE PROCESAMIENTO GRÁFICO EN LA DETERMINACIÓN DEL ESTADO ESTACIONARIO PERIÓDICO DE REDES ELÉCTRICAS"

TESIS

PARA OBTENER EL GRADO DE
MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA

PRESENTA
ING. VÍCTOR HUGO MONTESINOS GONZÁLEZ

DIRECTOR DE TESIS
DR. ANTONIO RAMOS PAZ

MORELIA, MICHOACÁN AGOSTO DEL 2013

DIVISIÓN POSGRADO

Dedicatoria

A **Dios**, por haberme dado la sabiduría y la fortaleza para que fuera posible alcanzar este triunfo, a mis padres: **Víctor Hugo Montesinos Horiuchi** y **Patricia González Liy**, por sus consejos, valores, motivación, sacrificios que hicieron a lo largo de mi carrera que me han permitido ser una persona de bien, por ser unos excelentes padres, pero más que nada, por su amor, a mi esposa: **Diana Judith Mondragón Sánchez**, por todo su amor, apoyo incondicional, comprensión, por darme la dicha de ser parte de su vida y por darme el mejor regalo de mi vida, mi hija: **Danna Sofía Montesinos Mondragón**.

Agradecimientos

- A la Universidad Michoacana de San Nicolás de Hidalgo (UMSNH), por abrirme las puertas y brindarme la oportunidad de cursar la maestría en sus aulas y con ello superarme.
- Al Consejo Nacional de Ciencia y Tecnología (CONACYT), por el apoyo económico durante el transcurso de la maestría.
- A los profesores de este posgrado por compartir su vasto conocimiento el cual me ha hecho crecer como persona y profesionista.
- Mi más sincero agradecimiento a mi Director de tesis: Dr. Antonio Ramos Paz, por su amistad, por compartir sus grandes conocimientos del tema, por tener la paciencia y escuchar atentamente los problemas y dudas que surgieron a lo largo de la tesis y ayudarme a resolverlos.
- A los profesores que participaron como revisores de tesis, por sus apreciables comentarios y correcciones que enriquecieron el presente trabajo.
- A mis abuelos: Rafael, Yaena, Rosalinda y Humberto, por su cariño tan especial y su confianza de siempre.
- A mis hermanos: Alejandra, Jackeline y Francisco, por todo el apoyo brindado, comprensión y cariño.
- A toda mi familia y amigos: por todo el apoyo moral ofrecido a lo largo de mi carrera y su amistad.
- A mis compañeros de maestría: Eduardo, Rafael, Celso, Sanzon, Pavel, Víctor, Gerson, Héctor y demás, por todo el apoyo mutuo en nuestra formación profesional y su amistad.

Resumen

En la actualidad las plataformas *multicore* lideran la industria de las computadoras, obligando a los desarrolladores de *software* a adaptarse a nuevos paradigmas de programación para poder explotar su capacidad de cómputo. Al día de hoy uno de los principales exponentes de las plataformas *multicore* son las Unidades de Procesamiento Gráfico (GPUs).

En concreto, NVIDIA y sus dispositivos, en los cuales nos centraremos, se han establecido como líderes en el sector, y su modelo de programación CUDA, basado en la utilización de GPUs para el cómputo de propósito general, es ampliamente usado.

La investigación llevada a cabo en la presente tesis describe la aplicación de técnicas de procesamiento paralelo multinúcleo basado en GPUs a la técnica Newton de Diferenciación Numérica, la cual permite acelerar la convergencia de las variables de estado al Ciclo Límite para obtener la solución en Estado Estacionario Periódico (EEP) de redes eléctricas con componentes no lineales y variantes en el tiempo. Dos variantes de programación del método de Diferenciación Numérica (DN) son propuestos e implementados en una GPU.

De igual forma, se aborda una estrategia alternativa de iteración durante la aplicación del método de DN, esto con la finalidad de reducir el número de ciclos completos de integración y por ende el tiempo de cómputo.

El potencial de dichas metodologías ejecutadas de manera paralela, se demuestra mediante la obtención del EEP de distintos casos de estudio del IEEE modificados con la inclusión de elementos no lineales. Las soluciones son comparadas en términos de eficiencia respecto a la aplicación de un método de integración convencional como es el Runge-Kutta de 4° orden y las contrapartes secuenciales de los métodos propuestos.

Abstract

At present, multicore platforms lead the computers industry, forcing software developers to adapt to new programming paradigms to be able to exploit their computation capacity. Today one of the main exponents of the multicore platforms is the Graphic Processing Units (GPUs).

In particular, NVIDIA and their devices, have settled as leaders in the sector, and their programming model CUDA, based on the use of GPUs for the general purpose computing, is broadly used.

The research conducted in this thesis describes the application of multicore parallel processing techniques based in GPUs to the Newton technique based on Numerical Differentiation (ND) approach, which can accelerate the convergence of the state variables to Limit Cycle to obtain the Periodic Steady State (PSS) solution of electric networks with nonlinear and time-varying components. Two Programming models of the ND technique are proposed and implemented on a GPU.

Also discusses an alternative iteration strategy during the application of the ND method, this in order to reduce the number of complete cycles of integration and therefore of the computation time.

The potential of these methodologies executed in parallel, is demonstrated obtaining the PSS solution of different IEEE test systems modified with the inclusion of nonlinear elements. The solutions are compared in terms of efficiency respect to the application of a conventional numerical integration process such as the fourth order Runge-Kutta method and their sequential counterparts of the proposed methods.

Índice general

Dedicatoria III

Agradecimientos V

Resumen VII

Abstract IX

Índice de Figuras XVI

Índice de Tablas XVII

Índice de Algoritmos XIX

Lista de Símbolos y Abreviaturas XX

Lista de Publicaciones XXIII

1. Introducción 1

1.1. Antecedentes . . . 2

1.1.1. Procesamiento paralelo aplicado a Sistemas Eléctricos de Potencia . . . 2

1.1.2. Técnicas para la determinación del Estado Estacionario Periódico . . . 3

1.1.2.1. Métodos en el dominio del tiempo . . . 4

1.1.2.2. Métodos en el dominio de la frecuencia . . . 5

1.1.2.3. Métodos en el dominio híbrido (Frecuencia-Tiempo) . . . 6

1.2. Objetivos . . . 6

1.2.1. Objetivo General . . . 6

1.2.2. Objetivos Específicos . . . 6

1.3. Justificación . . . 7

1.4. Metodología . . . 7

1.5. Contribuciones . . . 7

1.6. Estructura de la Tesis . . . 8

2. Procesamiento en paralelo: arquitecturas, plataformas y métricas **9**
2.1. Introducción . . . 9
2.2. Clasificación de computadoras paralelas . . . 10
2.2.1. Clasificación de Hwang-Briggs . . . 10
2.2.2. Clasificación de Flynn . . . 11
2.3. Multicomputadoras . . . 12
2.4. Multiprocesadores y Procesadores Multinúcleo . . . 13
2.4.1. Multiprocesadores . . . 13
2.4.2. Procesadores Multinúcleo . . . 13
2.5. Plataformas operativas para el procesamiento en paralelo . . . 14
2.5.1. PVM . . . 14
2.5.2. MPI . . . 15
2.5.3. Multithreading . . . 15
2.5.3.1. Concurrencia y paralelismo . . . 15
2.5.3.2. Estructura de MT . . . 16
2.5.4. OpenMP . . . 16
2.5.5. Unidades de Procesamiento Gráfico . . . 18
2.5.5.1. Arquitectura de una GPU moderna . . . 18
2.5.5.2. Modelo de programación CUDA . . . 21
2.5.5.3. Tipos de funciones . . . 25
2.5.5.4. Tipos de Variables . . . 25
2.5.5.5. Administración de la memoria . . . 25
2.6. Diseño de algoritmos paralelos . . . 26
2.7. Evaluación de algoritmos paralelos . . . 27
2.7.1. Ganancia de velocidad (*Speed-Up*) . . . 27
2.7.2. Eficiencia . . . 27
2.7.3. Escalamiento fuerte y la ley de Amdahl . . . 27
2.8. Conclusiones . . . 28

3. Técnicas numéricas para determinar el EEP en el dominio del tiempo **29**
3.1. Introducción . . . 29
3.2. Determinación de la solución de estado estable en el dominio del tiempo de redes eléctricas . . . 30
3.2.1. Métodos Newton para la obtención de una rápida solución al Estado Estacionario Periódico . 30
3.2.1.1. Método Newton para el cálculo del EEP basado en el Mapa de Poincaré . . . 31
3.2.1.2. Método Newton de Diferenciación Numérica . . . 32
3.2.1.3. Estrategia Alternativa de iteración: Método de Diferenciación con Jacobiano Constante 34
3.3. Conclusiones . . . 36

4. Paralelización de la técnica Newton de Diferenciación Numérica utilizando la plataforma de procesamiento en paralelo CUDA **37**
4.1. Introducción . . . 37
4.2. Propuestas de paralelización del cálculo de la matriz de transición de estados . . . 38

4.2.1. Primera Propuesta de Paralelización del cálculo de la matriz de transición de estados empleando CUDA . . . 38
4.2.1.1. Descripción del primer Algoritmo paralelo para el cálculo de la matriz de identificación propuesto con CUDA . . . 40
4.2.2. Segunda Propuesta de Paralelización del cálculo de la matriz de transición de estados empleando CUDA . . . 41
4.2.2.1. Descripción del segundo Algoritmo paralelo para el cálculo de la matriz de identificación propuesto con CUDA . . . 45
4.3. Método paralelo de Diferenciación Numérica con Jacobiano Constante . . . 46
4.4. Conclusiones . . . 46

5. Casos de estudio y su solución en el dominio del tiempo **47**
5.1. Introducción . . . 47
5.2. Sistema de 3 nodos . . . 49
5.3. Sistema de IEEE de 14 nodos modificado . . . 52
5.4. Sistema de IEEE de 30 nodos modificado . . . 54
5.5. Sistema de IEEE de 57 nodos modificado . . . 57
5.6. Sistema de IEEE de 118 nodos modificado . . . 59
5.7. Análisis de la eficiencia relativa (*Speed up*) . . . 62
5.8. Conclusiones . . . 63

6. Conclusiones Generales y Sugerencias para Trabajos Futuros **65**
6.1. Conclusiones Generales . . . 65
6.2. Recomendaciones para trabajos futuros . . . 66

Apendices **67**

Bibliografia **73**

Índice de figuras

2.1. Clasificacion de Hwang-Brigss . . . 10
2.2. Clasificacion de Flynn . . . 11
2.3. Arquitectura MIMD . . . 12
2.4. Plataformas de procesamiento en paralelo . . . 14
2.5. a)Ejecución concurrente y b) Ejecución paralela de p hilos . . . 16
2.6. Modelo de Programación de OpenMP(*Fork-Join*) . . . 17
2.7. Arquitectura GPU modelo GTX 280 [Marandi & Dinavahi, 2009] . . . 18
2.8. Modelo de memoria de una GPU. . . . 19
2.9. Asignación de bloques a los SMs (Streaming Multiprocessors) de un TPC. . . . 21
2.10. Modelo de programamción CUDA NVIDIA. . . . 22
2.11. Grid de Bloques de hilos . . . 24
2.12. Escalabilidad de un Kernel CUDA en distintos dispositivos . . . 25
2.13. diseño de algoritmos paralelos . . . 26
2.14. Ley de Amdahl . . . 28

3.1. Plano de Poincaré . . . 31
3.2. Procedimiento de Diferenciación Numérica . . . 33
3.3. Método simple de diferenciación numérica . . . 35

4.1. Primer propuesta de cálculo paralelo de la matriz de transición . . . 39
4.2. Ciclo para obtener k_2 en el metodo RK4 convencionalmente . . . 43
4.3. Ciclo para obtener k_2 de manera paralela con el uso de *threads* [*Magaña* et al*., 2013*] . . . 43
4.4. Segunda propuesta paralela de calculo de la matriz de transición [Magaña *et al.*, 2013] . . . 45

5.1. a) CPU modelo E5606 b) NVIDIA GPU modelo Tesla C2075 . . . 48
5.2. Caso de estudio 1: Sistema eléctrico de 3 nodos monofásico [Semlyen & Medina, 1995]. . . . 49
5.3. Forma de onda de la corriente en la LT de 1-2. . . . 51
5.4. Comportamiento del Voltaje nodo 2 . . . 51
5.5. Sistema de 14 nodos modificado del IEEE . . . 52
5.6. Comportamiento del Voltaje en el nodo 10 . . . 54
5.7. Flujo en la rama magnetizante nodo 10 . . . 54
5.8. Voltaje en el nodo 22 . . . 56

5.9. Corriente en la línea que une los nodos 10 y 22 56
5.10. Voltaje en el nodo 38 58
5.11. Forma de onda de del voltaje en el nodo 15. Caso de estudio 5.5 59
5.12. Voltaje en el nodo 64 61
5.13. Flujo en la Rama Magnetizánte del nodo 64 61

6.1. Modelo de la línea de transmisión monofásica 67
6.2. Modelo del generador monofásico 68
6.3. Modelo de Banco de capacitores monofásico 68
6.4. Modelo de rama magnetizante monofásica 69
6.5. Modelo del horno de arco eléctrico monofásico 70

Índice de Tablas

Índice de Tablas

2.1. Memorias en una GPU. 20

5.1. Sistema de 3 nodos, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas en una computadora convencional. 50

5.2. Sistema de 3 nodos, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas mediante CPU-GPU. 50

5.3. Sistema de 14 nodos, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas en una computadora convencional. 53

5.4. Sistema de 14 nodos, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas mediante CPU-GPU. 53

5.5. Sistema de 30 nodos, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas en una computadora convencional. 55

5.6. Sistema de 30 nodos, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas mediante CPU-GPU. 55

5.7. Sistema de 57 nodos, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas en una computadora convencional. 58

5.8. Sistema de 57 nodos, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas mediante CPU-GPU. 58

5.9. Sistema de 118 nodos, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas en una computadora convencional. 61

5.10. Sistema de 118 nodos, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas mediante CPU-GPU. 61

5.11. Concentrado de tiempos de simulación de los casos de estudio. 63

5.12. Eficiencia relativa obtenida en los casos de estudio. 63

Índice de algoritmos

2.1. Ejemplo de Construcción de un Kernel 23
4.1. Primera implementación del cálculo paralelo de la matriz de transición de estados 41
4.2. Cálculo convencional de las pendientes del método de Runge Kutta de 4° orden 42
4.3. Cálculo paralelo de las pendientes del método de Runge Kutta de 4° orden 44
4.4. Segunda implementación del cálculo paralelo de la matriz de transición de estados 46

Lista de Símbolos y Abreviaturas

AD	Aproximación Directa
ANSI	*American National Standards Institute*
API	*Application Programming Interface*
ARB	*Architectural Review Board*
ASCII	*American Standard Code for Information Interchange*
BC	Banco de Capacitores
CA	Corriente Alterna
CD	Corriente Directa
CL	Ciclo Limite
CUDA	*Compute United Device Architecture*
DN	Diferenciación Numérica
DN-$\mathbf{\Phi}$	Diferenciación Numérica y empleando la primer propuesta de paralelización
DN-$\mathbf{\Phi_{RK}}$	Diferenciación Numérica y empleando la segunda propuesta de paralelización
DNJC	Diferenciación Numérica con Jacobiano Constante
DNJC-$\mathbf{\Phi}$	Diferenciación Numérica con Jacobiano Constante y empleando la primer propuesta de paralelización
DNJC-$\mathbf{\Phi_{RK}}$	Diferenciación Numérica con Jacobiano Constante y empleando la segunda propuesta de paralelización
DNM	Diferenciación Numérica Mejorado
e_i	columna i-ésima de la matriz identidad
E	Eficiencia
EDOs	Ecuaciones Diferenciales Ordinarias

EEP	Estado Estacionario Periódico
FACTS	*Flexible AC Transmission Systems*
FB	Fuerza Bruta
GPGPU	*General Purpose Computing on the GPU*
GPUs	*Graphical Processing Units*
ME	Matriz Exponencial
MED	Matriz Exponencial Discreta
MISD	*Multiple Instruction stream, Single Data stream*
MIMD	*Multiple Instruction stream, Multiple Data stream*
MPI	*Message Passing Interface*
MT	*Multithreading*
MUL	*Multiply*
NCCI	Número de ciclos completos de integración
N_P	Número de Procesadores
OpenMP	*Open Multi Processing*
P	Elementos de proceso
p.u.	Por Unidad
PVM	*Parallel Virtual Machine*
RM	Rama Magnetizante
RK4	Runge-Kutta de 4° Orden
SEP	Sistema Eléctrico de Potencia
SIMD	*Single Instruction stream, Multiple Data stream*
SISD	*Single Instruction stream, Single Data stream*
SFUs	*Special Function Units*
SM	*Stream Multiprocessor*
S_P	*Speed up*

SP	*Stream Processors*
SM	*Stream Multiprocessor*
S_P	*Speed up*
SP	*Stream Processors*
T_P	Tiempo del programa con p procesadores
T_S	Tiempo del programa secuencial
T_{sp}	Tiempo de la sección paralela del programa
TPC	*Thread Processing Cluster*
UPFC	*Unified Power Flow Controller*
$\dot{\boldsymbol{x}}$	Vector de derivadas de variables de estado $\boldsymbol{x}$
$\boldsymbol{x}$	Vector de variables de estado
$\boldsymbol{x}_\infty$	Vector de variables de estado al ciclo limite
$\boldsymbol{P}$	Plano de Poincaré
$\boldsymbol{\Phi}$	Matriz de transición de estados
$\boldsymbol{\Lambda}$	Flujo magnético
$\boldsymbol{J}(t)$	Jacobiano
$\boldsymbol{\xi}$	perturbación de las variables de estado
Δ	Incremento
T	Periodo
$\frac{d}{dt}$	Derivada con respecto al tiempo
THD	*Total Harmonic Distortion*

Lista de Publicaciones

- Rápida Solución en Estado Estacionario Periódico de Sistemas Eléctricos No-Lineales Usando Unidades de Procesamiento Gráfico, Víctor O. Montesinos Glez., Ernesto Magaña L., Antonio Ramos P. y Aurelio Medina R., Artículo presentado en la XIV Reunión de Otoño de Potencia, Electrónica y Computación, ROPEC'2012 INTERNACIONAL, celebrada en la Ciudad de Colima, Colima, México, del 7 al 9 de Nov. del 2012. ISBN: 978-607-95476-6-0.
- *Application of Graphics Processing Units in the Fast Steady State Solution of Electric Power Systems*, Víctor H. Montesinos Glez., Ernesto Magaña L., Antonio Ramos P. y Aurelio Medina R., Artículo presentado en la *4th. INTERNATIONAL SUPERCOMPUTING CONFERENCE IN MEXICO*, ISUM 2013 INTERNACIONAL, celebrada en la Ciudad de Manzanillo, Colima, México, del 5 al 8 de Marzo del 2013.
- *Periodic Steady State Determination of Power Systems Using Graphics Processing Units*, Ernesto Magaña L., Víctor H. Montesinos Glez., Antonio Ramos P. y Aurelio Medina R., Artículo enviado a la *10th. International Conference on Electrical Engineering, Computing Science and Automatic Control*, CCE 2013, a realizarse en la Ciudad de México D.F., México, del 30 de septiembre al 4 de Octubre del 2013.

Capítulo 1

Introducción

En la operación de un Sistema Eléctrico de Potencia (SEP) resulta indispensable una cuidadosa planeación para mantener un suministro eléctrico sin interrupciones, conservando la calidad del servicio requerido por los usuarios al menor costo posible y dentro de márgenes de seguridad. Es por ello que en el ámbito del análisis y planeación de los SEP una de las herramientas más ampliamente utilizadas es la simulación digital. Por medio de ella se puede reproducir y predecir el comportamiento del sistema, además de brindar la posibilidad de estudiar los efectos de la red de potencia ante una gran diversidad de escenarios de operación.

Una observación detallada del sistema que se está simulando puede conducir a un mejor entendimiento y por consiguiente a sugerir o tomar las decisiones más pertinentes que mejoren la operación y su eficiencia. El uso de la simulación facilita y mejora de manera considerable las tareas de planeación, asimismo contribuye a la disminución en los costos de producción.

En las últimas décadas el análisis de grandes redes eléctricas mediante simulación se ha tornado una tarea sumamente compleja, debido a que la red se ha incrementado de manera considerable y se han incorporado nuevos dispositivos y modelos. Esto ha ocasionado que los tiempos de simulación se incrementen al igual que la necesidad de tener mayor capacidad de cómputo.

La disponibilidad de hardware y software de procesamiento en paralelo presenta una oportunidad y un desafío para ser aplicado al análisis de los SEP. Además tiene el potencial de ser rentable y la capacidad de utilizarse para computaciones intensas [Alvarado *et al.*, 1992]. Asimismo el enfoque paralelo promete ser la mejor alternativa a largo plazo ya que es la única manera de superar las limitaciones de la arquitectura tradicional [Patel *et al.*, 1990], tales como la frecuencia de reloj que es limitada por la velocidad de la luz y la reducción del tamaño de los elementos la cual se restringe por el tamaño de las moléculas [Jin, 94].

En particular, esta tesis presenta la aplicación de procesamiento en paralelo basado en Unidades de Procesamiento Grafico (*GPUs*, por sus siglas en inglés) a la técnica Newton de Diferenciación Numérica (DN) [Semlyen & Medina, 1995], esto con la finalidad de disminuir el tiempo requerido para la obtención de la solución en Estado Estacionario Periódico (EEP) en el dominio del tiempo de los SEP con componentes no lineales y variantes en el tiempo. Dos modelos de programación del método de DN son propuestos e implementados en una *GPU* mediante la plataforma *CUDA*. En el primer modelo se tiene la particularidad de ser ejecutada de manera paralela el cálculo de la matriz de transición de estados o matriz de identificación $\mathbf{\Phi}$, mientras que en el segundo modelo, además de obtenerse la matriz de transición de estados de manera paralela, implícitamente el método de integración utilizado para el cálculo de las columnas de la matriz de identificación es ejecutado de manera paralela.

De igual manera, en esta tesis se analiza una variante del método de DN que es propuesta en [Semlyen & Medina, 1995], la cual consiste en calcular el Jacobiano en una sola ocasión y mantenerlo constante, esto con la finalidad de disminuir el número de ciclos completos de integración necesarios para alcanzar el EEP y por ende el tiempo de cómputo.

1.1. Antecedentes

1.1.1. Procesamiento paralelo aplicado a Sistemas Eléctricos de Potencia

El cómputo paralelo y distribuido se encuentra entre las nuevas tecnologías que presentan un gran potencial para su aplicación en las metodologías utilizadas en el análisis y planeación de los SEP , así como el control en tiempo real [Falcao, 1994].

En la literatura existen diversos textos que incorporan técnicas de procesamiento en paralelo para el análisis de redes eléctricas. A continuación se presentan una serie de trabajos en donde se involucran dichas técnicas.

En [Happ, 1969] y [Happ & Undrill, 1969] se hace el uso de una configuración multicomputadora y la teoría de división de redes (Diakoptics) propuesta por Gabriel Kron, para el problema de despacho de potencia real y para el problema de flujos de potencia respectivamente, ambos trabajos aplicados con la finalidad de coordinar las operaciones de grupos de sistemas de potencia interconectados. Por su parte [Hatcher *et al.*, 1977] implementó un sistema multiprocesador para la solución de problemas de estabilidad transitoria. En este documento Hatcher obtuvo un resultado equivalente al 3 % del tiempo del algoritmo implementado de manera secuencial.

En [Oyama *et al.*, 1990] se propone la aplicación de procesamiento paralelo basado en sistemas multiprocesador a un algoritmo para la solución de ecuaciones lineales. Este algoritmo fue probado en el análisis de sistemas de potencia resultando hasta 2.7 veces más rápido que su contraparte probada en un solo procesador.

Por otro lado en [Stavrakakis *et al.*, 1990] se expone un algoritmo paralelo enfocado a la detección de fallas en manejadores de motores de corriente directa. En [Alvarado *et al.*, 1992] se define el estado del arte del procesamiento en paralelo y se identifica los posibles campos de aplicación para la solución de los sistemas de potencia. Más adelante [Werler & Glavitsch, 1993] realizan un análisis de transitorios electromagnéticos en donde consiguen obtener un *speedup* aproximado de 7. En [Marinos *et al.*, 1994] se explora la aplicación de procesamiento en paralelo al análisis de armónicos en sistemas eléctricos de potencia. Posteriormente [Borges & Falcao, 1999] publican una metodología para la paralelización del análisis de confiabilidad de sistemas de potencia utilizando simulación Monte Carlo. La metodología propuesta por Borges Y Falcao se implementó en un sistema multicomputadora y la plataforma de programación paralela utilizada fue en base a la Interface de Paso de Mensajes (MPI, por sus siglas en inglés). En este documento obtienen un speed up que oscila entre 1.8 y 4.0, dependiendo del número de procesadores empleados.

Subsecuentemente en la contribución hecha por [Garcia *et al.*, 2001] se examina la aplicación de técnicas de procesamiento en paralelo a las técnicas Newton de acercamiento rápido al Ciclo Límite (CL) . También en [Garcia & Acha, 2004] se presentan las técnicas de acercamiento rápido al CL para la evaluación de un sistema de 118 nodos trifásico modificado utilizando la plataforma Multithreading. Durante este trabajo los autores reportan un porcentaje equivalente al 5 % del tiempo de ejecución secuencial. Una aportación importante fue hecha por [Garcia, 2005], en donde utiliza la plataforma MPI para el análisis de un sistema eléctrico de 118 nodos que incluye dispositivos FACTS.

El trabajo realizado por [Ramos, 2002] exhibe un análisis preciso de 2 plataformas operativas: Multithreading [Kleiman *et al.*, 1992] y PVM [Geist *et al.*, 1994]; la primera consiste en un estándar para programación de sistemas multiprocesadores, la cual separa un proceso en muchos hilos de ejecución, cada uno se ejecuta por separado, pero

comparten una memoria general. La segunda se basa en un esquema para la programación multi-computadora en donde se convierte una red ya sea homogénea o heterogénea de computadoras en una gran computadora multi-procesadores.

En el trabajo de [Medina & Ramos, 2005] se aplican las técnicas de procesamiento en paralelo basadas en dos plataformas distintas, al análisis de la respuesta a la frecuencia de redes eléctricas.

Otra contribución en la cual se utilizan las plataformas PVM y MT combinadas con un método Newton para obtener una rápida solución del EEP en el dominio del tiempo de sistemas eléctricos se encuentra en [Medina *et al.*, 2006]. En este artículo los autores reportan eficiencias relativas del orden de 2.9 a 9.7, lo cual demuestra la potencialidad de aplicar procesamiento en paralelo a sistemas de gran escala.

Por su parte [Mora, 2008] aplica procesamiento paralelo basado en MPI al algoritmo de diferenciación numérica para obtener la solución en EEP.

El articulo presentado por [Peña & Medina, 2010], recurre a un método Newton basado en DN para conseguir la solución en estado estacionario periódico de una máquina de inducción de una manera eficiente y con el fin de aumentar la eficacia del proceso de DN, aplican técnicas de procesamiento paralelo basadas en threads. Esta metodología es aplicada a parques eólicos consiguiendo eficiencias relativas que oscilan entre 1.55 y 4.0.

El uso de técnicas de procesamiento paralelo de granularidad fina basadas en la plataforma OpenMP para el análisis de redes eléctricas es expuesto en [Rico & Paz, 2011] y [Rico, 2012]. Al igual en [Ortega, 2012] se utiliza la plataforma OpenMP para paralelizar el método Newton de la matriz exponencial discreta (MED) y obtener con ello el EEP.

Por otro lado recientemente se han incorporado el uso de Unidades de Procesamiento Grafico (GPUs) a la solución de problemas enfocados a sistemas de potencia. Por ejemplo en [Gopal *et al.*, 2007] y [Gopal (2010)] se aborda la implementación de GPUs al problema de flujos de potencia de CD para el análisis de contingencias. La eficiencia relativa reportada en estos trabajos de investigación se encuentra entre 2.3 y 4.8.

En [Marandi & Dinavahi, 2010] se resuelve problemas de estabilidad transitoria con ayuda de estos coprocesadores, obteniéndose una importante ganancia de velocidad de aproximadamente 480. Mientras que en [Kamiabad, 2011] se presenta el uso de GPUs, para la solución lineal iterativa pre condicionada basada en métodos de Krylov, para la solución de sistemas de gran escala.

El articulo presentado por [Debnath *et al.*, 2012] muestra la implementación del análisis de transitorios electromagnéticos de sistemas de gran escala usando GPUs. En este trabajo los autores reportan un *speed up* de 70.

En [Jing *et al.*, 2013] se presenta un método novedoso de análisis de subestaciones inteligentes mediante computo paralelo basado en CUDA. Los resultados mostrados en este artículo son comparados con la plataforma MPI y una computadora convencional. En él se muestra un *speed up* aproximado a 168.

1.1.2. Técnicas para la determinación del Estado Estacionario Periódico

Aunque muchos programas de simulación se centran en el análisis de transitorios, resulta de interés para los diseñadores el comportamiento en EEP de circuitos eléctricos [Kundert *et al.*, 1990]. Esto se debe a que diferentes tipos de análisis requieren el cálculo de esta etapa para poder ser realizados [Segundo & Medina, 2010aa]. Por ejemplo, en circuitos electrónicos las especificaciones de diseño son en base a su operación en estado estacionario. En lo que respecta a calidad de energía, la solución en EEP es necesaria para determinar el contenido armónico de una red eléctrica [Segundo & Medina, 2010aa].

El detectar y predecir armónicos permite realizar una evaluación y diagnóstico de la calidad de la energía. La detección se logra mediante el procesamiento en tiempo real de la información monitoreada en la red mientras que la predicción se obtiene mediante modelos analíticos implementados en simulaciones digitales. A este último enfoque

corresponden las técnicas descritas en este apartado de la tesis.

Se han implementado y desarrollado diversas técnicas para la determinación de la solución en EEP para distintos marcos de referencia y así obtener estudios precisos de análisis armónicos. En [Medina et al., 2013] se analizan las características de los métodos y en base a ello los clasifica en las siguientes categorías:

- Métodos en el dominio del tiempo
- Métodos en el dominio de la frecuencia
- Métodos híbridos (Tiempo-Frecuencia)

1.1.2.1. Métodos en el dominio del tiempo

El comportamiento de un SEP puede representarse por un conjunto de Ecuaciones Diferenciales Ordinarias (EDOs), así que la solución en EEP del sistema eléctrico puede obtenerse directamente integrando las EDOs que modelan su dinámica durante un lapso de tiempo suficientemente grande como para que el transitorio pueda ser despreciable [Parker & Chua, 1989]. Este procedimiento, conocido como método de Fuerza Bruta (FB), es el más simple [Parker & Chua, 1989].

La ventaja de realizar el estudio de un sistema eléctrico en el dominio del tiempo es que se puede determinar de manera implícita las componentes armónicas asociadas con las formas de onda de las variables que representan el comportamiento de los elementos de la red [Medina et al., 2013]. Sin embargo, tiene algunos inconvenientes que lo hacen en muchas ocasiones inapropiado, tal como en el caso de sistemas que contienen elementos como transformadores de potencia, generadores y dispositivos FACTS los cuales poseen pobre amortiguamiento [Chua & Ushida, 1981] o sistemas que sean difíciles de integrar a una alta precisión, como es el caso de los sistemas rígidos (Stiff).

Con el propósito de eliminar la severa limitante de aplicación e ineficiencia computacional de los métodos convencionales de Fuerza Bruta para obtener las soluciones periódicas de sistemas eléctricos de potencia, se han desarrollado otros métodos basados en métodos Newton, que son conocidos como métodos de disparo [Parker & Chua, 1989].

La primera contribución orientada a la determinación rápida del estado estacionario periódico de redes no-lineales en el dominio del tiempo es presentada por Aprille y Trick [Aprille & T. N. Trick, n.d.]. En otras contribuciones [Bedrosian & Vlach, 1992], [Donde & Hiskens, 2006], aplican los métodos de disparo a sistemas eléctricos con dispositivos de conmutación.

Posterior, a los métodos de disparo, se han propuesto métodos Newton basados en el mapa de Poincaré [Parker y Chua, 1989] para el cálculo de la solución en EEP. En el trabajo presentado en [Semlyen y Medina, 1995] se describen tres técnicas Newton que permiten la aceleración de las variables de estado al Ciclo Límite. Los métodos propuestos fueron: Diferenciación Numérica (DN), Aproximación Directa (AD) y Matriz Exponencial (ME).

Al igual en [Semlyen y Medina, 1995] se presentan dos estrategias alternativas de iteración para el cálculo en EEP en el dominio del tiempo. Ambas estrategias son planteadas con la finalidad de reducir el número de ciclos completos de integración calculados y por ende el tiempo de cómputo.

En [Garcia *et al.*, 2001]se examina la aplicación de técnicas de procesamiento en paralelo basada en la plataforma MPI, a las técnicas Newton de acercamiento rápido al CL. En el trabajo de [Garcia y Acha, 2004] se presenta la aplicación de estas técnicas en el análisis de redes eléctricas trifásicas para estudios de armónicos. En [Chang *et al.*, 2006] se determina la solución en EEP de redes con convertidores de CA/CD, utilizando el mapa de Poincaré. En [Segundo & Medina, 2008] se detalla una técnica en el dominio del tiempo para la solución en EEP de sistemas eléctricos con controladores de flujo de potencia unificado (UPFC, por sus siglas en inglés) incorporando tres estrategias para la

representación del proceso de conmutación del UPFC, la solución en EEP es obtenida con el método Newton basado en el proceso de DN. Recientemente en las contribuciones hechas por [Segundo & Medina, 2010aa] y [Segundo & Medina, 2010bb] se proponen las metodologías de Diferenciación Numérica Mejorada (DNM) y Matriz Exponencial Discreta (MED) respectivamente, las cuales son técnicas Newton eficientes para la obtención de la solución de EEP de sistemas eléctricos no lineales.

En [Lian & Noda, 2010] se presenta un método de flujos de potencia armónicos, el cual se implementa completamente en el dominio del tiempo, el método propuesto esencialmente es una extensión del método de disparo para incluir las restricciones de flujos de potencia y así poder agregar cargas en los cálculos de flujos de potencia

1.1.2.2. Métodos en el dominio de la frecuencia

Estos métodos son muy utilizados debido a la facilidad para construir el conjunto de ecuaciones que describen la dinámica del sistema a ser analizado. En [Medina et al., 2013] se realiza una clasificación de los métodos en el dominio de la frecuencia en las siguientes categorías:

- Método directo
- Análisis armónico iterativo
- Flujos de potencia armónicos

En el método directo la respuesta a la frecuencia de una red eléctrica, se obtiene inyectando una corriente de uno en por unidad o un voltaje en el nodo de interés con pasos de frecuencia para un rango en particular, el método más simple de análisis utiliza el marco de referencia de las componentes de secuencia para determinar la propagación de corrientes armónicas características mediante inyección de fuentes de corriente ideales en la red eléctrica [Mahmoud & Schultz, 1982]. En una contribución posterior, la solución de sistemas de potencia se obtiene directamente en el marco de referencia de fases para sistemas trifásicos [Howroyd, 1982]. Un sistema que contenga excitación híbrida, es decir, voltajes armónicos en algunos nodos e inyecciones de corriente en otros; se resuelve dividiendo la matriz de admitancia y realizando una inversión parcial. Este procedimiento de solución híbrida permite que las tensiones y corrientes armónicas nodales desconocidas sean encontradas. En [Medina & Arrillaga, 1990] se presenta una técnica eficiente para la solución de redes no lineales que contengan excitaciones de voltaje y corriente.

El análisis armónico iterativo (IHA, por sus siglas en inglés) se encuentra basado en sustituciones sucesivas del tipo Gauss-Seidel. Los dispositivos que producen armónicos son modelados como una fuente de corriente dependiente de voltaje; representada por una fuente de corriente de armónicos fija en cada iteración. Usando una fuente de voltaje estimada se obtienen las corrientes armónicas, entonces las corrientes armónicas son usadas para obtener los voltajes, éstos voltajes a su vez son utilizados para el cálculo de corrientes armónicas más precisas. El proceso de solución se detiene una vez que los cambios en las corrientes armónicas son suficientemente pequeños [Callaghan & Arrillaga, 1990].

Los métodos de flujos de potencia armónicos (HPF por sus siglas en inglés), se basan en un proceso de linealización alrededor de un punto particular de operación de componentes no lineales y variantes en el tiempo, se obtiene así una relación lineal entre voltajes y corrientes armónicas. Esta condición es válida únicamente en una proximidad muy cercana al punto de operación, estos métodos toman en cuenta la naturaleza dependiente de la tensión de los componentes de potencia. En general, las ecuaciones de voltajes y corrientes armónicas se resuelven simultáneamente usando algoritmos del tipo Newton [Xia & Heydt, 1982]. Los armónicos producidos por elementos no lineales y

variantes en el tiempo son incorporados por medio de sus equivalentes armónicos Norton en un marco de referencia unificado denominado Dominio Armónico[Arrillaga *et al.*, 1995]. El cálculo de dicho equivalente puede no ser fácil y para obtener resultados precisos debe ser actualizado iterativamente. Este esfuerzo computacional se incrementa en proporción directa al tamaño del sistema analizado y al número de armónicos representados explícitamente. Esta metodología ha sido reportada como robusta numéricamente y con buenas características de convergencia en [Medina, 1992]. Al igual en [Arrillaga *et al.*, 2000] se presentan ventajas del modelado en el dominio de la frecuencia.

1.1.2.3. Métodos en el dominio híbrido (Frecuencia-Tiempo)

Son métodos que aprovechan las ventajas tanto del dominio del tiempo como de la frecuencia., en donde los componentes del sistema de potencia se representan directamente en sus marcos de referencia naturales. Es decir, los lineales en el dominio de la frecuencia y los no lineales y variantes en el tiempo en el dominio del tiempo, aplicando en estos últimos las técnicas de aceleración al Ciclo Limite para obtener de manera eficiente el EEP.

En [Usaola, 1990] se presenta un análisis híbrido en el cual se incorpora una modificación de la técnica de acercamiento rápido al EEP propuesta en [Aprille & T. N. Trick, n.d.] para el acercamiento rápido al EEP de la parte no lineal. Tambien en [Semlyen & Medina, 1995] se propone una técnica de análisis hibrida; en este trabajo se proponen tres métodos Newton para alcanzar la rápida convergencia hacia el EEP de la parte no lineal del sistema. Las técnicas propuestas tanto en [Usaola, 1990] como en [Semlyen & Medina, 1995] son aplicadas en el análisis de sistemas eléctricos monofásicos con componentes no lineales. La metodología propuesta por [Semlyen & Medina, 1995] se aplica a una red trifásica simple en [Semlyen & Shlash, 2000].

1.2. Objetivos

1.2.1. Objetivo General

- Aplicar técnicas de procesamiento en paralelo basado en Unidades de Procesamiento Gráfico a la solución en estado estacionario periódico de redes eléctricas con componentes no lineales y variantes en el tiempo.

1.2.2. Objetivos Específicos

- Implementar de manera paralela en la plataforma CUDA, el cálculo de la matriz de transición de estados, la cual es necesaria para la obtención de las variables de estado al CL en los métodos Newton de Diferenciación Numérica y el método de Diferenciación Numérica con Jacobiano constante. Posteriormente realizar un análisis cuantitativo de la solución obtenida de manera secuencial y paralela en EEP de sistemas de potencia mediante la aplicación de los métodos de DN y DN con Jacobiano Constante.
- Obtener el espectro armónico presente en los casos de estudio debido a la incorporación de elementos no lineales.

1.3. Justificación

En la actualidad, una correcta planeación de los sistemas eléctricos es fundamental para poder garantizar la seguridad, calidad y economía del suministro, pero con la constante expansión del sistema y la introducción de nuevas tecnologías (dispositivos de electrónica de potencia y fuentes alternas de energía), la demanda computacional aumentó considerablemente en las últimas décadas y con ella los tiempos de simulación.

Por dichas razones se justifica la búsqueda de herramientas computacionales, tanto de hardware como de software, que proporcionen un menor esfuerzo computacional, optimicen los tiempos de cómputo y permitan hacer análisis a sistemas de potencia de mayor escala.

El procesamiento en paralelo se implementa como una alternativa para poder afrontar dichos inconvenientes además de superar las limitantes que tiene una arquitectura uniprocesador y aprovechar los avances tecnológicos.

En esta tesis se optó por implementar en una GPU mediante la plataforma paralela CUDA, el cálculo de la matriz de transición de estados necesaria para los métodos Newton de DN y DN con Jacobiano Constante, ya que en la revisión del estado del arte, no se encontró ningún documento que emplee dicha metodología para obtener de manera eficiente el EEP en el dominio del tiempo.

1.4. Metodología

La investigación reportada en esta tesis, está basada en la realización del siguiente procedimiento metodológico:

1. Revisión del estado del arte sobre procesamiento en paralelo aplicado a SEP, técnicas numéricas y computacionales para la determinación del estado estacionario periódico enfocándose principalmente en el método Newton basado en Diferenciación Numérica.
2. Representar mediante modelos monofásicos cada uno de los elementos de la red de potencia en el dominio del tiempo, así como su implementación
3. Programación del método Newton basado en Diferenciación Numérica y del método de Diferenciación Numérica con Jacobiano Constante, ambos escritos en lenguaje C.
4. Incorporación de técnicas de procesamiento en paralelo multinúcleo basada en la plataforma CUDA a los métodos de DN y SDN.
5. Evaluación de programas implementados mediante casos de estudio.
6. Comparar los resultados obtenidos y los tiempos de cómputo.
7. Análisis de conclusiones y recomendaciones para trabajos futuros.

1.5. Contribuciones

- Utilización de Unidades de Procesamiento Grafico (GPUs) en el análisis de Sistemas de Potencia, en este caso en particular con la finalidad de obtener el EEP de manera eficiente.
- Paralelización de la matriz de transición de estados necesaria para los métodos de DN y DN con Jacobiano Constante mediante CUDA.

- Implementación paralela en CUDA del método de integración de Runge Kutta de 4° orden necesario para el cálculo de la matriz de transición de estados de los métodos Newton de DN y DN con Jacobiano Constante.

1.6. Estructura de la Tesis

Esta tesis se encuentra estructurada en 6 capítulos, de los cuales se proporciona una breve descripción a continuación.

En el capítulo 1 se presenta una introducción de la tesis que contempla todo lo relacionado con la revisión del estado del arte sobre técnicas de procesamiento en paralelo implementadas al análisis de SEP, al igual se redactan los antecedentes de las técnicas de acercamiento al Estado Estacionario Periódico. Asimismo se plantean los objetivos, justificación, contribuciones y Metodología implementada en este trabajo.

En el Capítulo 2 se abordan las técnicas de procesamiento en paralelo, tanto los tipos de arquitecturas, plataformas existentes teniendo énfasis en CUDA. De igual manera se analizan las métricas utilizadas en este trabajo de investigación para poder calcular la eficiencia del algoritmo implementado.

En el Capítulo 3 se describe las técnicas numéricas empleadas en la determinación del EEP, centrándose en el método Newton de diferenciación numérica, de igual forma se explica la alternativa de iteración del método de DN.

En el Capítulo 4 se detalla la incorporación de técnicas de procesamiento en paralelo basadas en CUDA a los método Newton de DN y DN con Jacobiano Constante.

En el Capítulo 5 se exponen y analizan distintos casos de estudio de redes eléctricas monofásicas de pequeña y mediana escala con la finalidad de comparar los métodos de DN y DN con Jacobiano Constante implementados en CUDA, además se da una breve descripción de los resultados obtenidos en términos de la eficiencia relativa, así como su contenido armónico.

En el Capítulo 6 se presentan las conclusiones generales obtenidas como resultado del trabajo realizado y se dan recomendaciones para trabajos de investigación posteriores.

Capítulo 2

Procesamiento en paralelo: arquitecturas, plataformas y métricas

2.1. Introducción

Un desafío importante en la actualidad es el uso óptimo de las computadoras para conseguir objetivos que de otra forma no serian alcanzables. La respuesta a este desafío desde el punto de vista de hardware viene marcada por la evolución de la tecnología al conseguir computadoras más veloces, y por la organización de su arquitectura, utilizando el paralelismo en la resolución de problemas previamente descompuestos en sub-problemas independientes. Esta tendencia a la alza en los últimos años puede verse reflejada claramente en todos los tipos de computadoras presentes en el mercado, desde las computadoras de escritorio y personales hasta las supercomputadoras. La tecnología multinúcleo, la utilización de computadoras interconectadas mediante redes locales o de área amplia, el uso de procesadores gráficos, son algunos ejemplos claros de cómo la computación paralela va imponiéndose en la organización de las computadoras.

El procesamiento en paralelo puede ser definido como la habilidad de un sistema de llevar a cabo múltiples operaciones o procesos, de tal manera que un problema complejo puede ser dividido en fracciones más pequeñas y ser resuelto simultáneamente [Grama *et al.*, 2003]. En otro punto de vista [Alvarado *et al.*, 1992], especifica que el cómputo paralelo se basa en una forma de procesamiento de información en la cual dos o más procesadores juntos o con algún tipo de esquema de comunicación entre ellos, pueden cooperar para obtener la solución de un problema.

Gracias a las computadoras paralelas ha sido posible acercarse paulatinamente a la solución de grandes problemas, que anteriormente debido a las limitaciones tecnológicas parecían intratables, tales como el cifrado del genoma humano, la predicción meteorológica a nivel mundial, la evolución galáctica, diseño de aeronaves, dinámica molecular, análisis de sistemas de potencia de gran escala, entre otros [Quinn, 2004]. No obstante, el salto hacia esta tecnología ha impuesto un cambio drástico en el modo de concebir el software desde el punto de vista de los desarrolladores. Ya no es posible incrementar el rendimiento y la funcionalidad de las aplicaciones de una manera tan sustancial como hace unos años, simplemente esperando a que una nueva gama de procesadores aparezca en el mercado. Ahora, es el propio programador el que debe hacer el esfuerzo para que su aplicación se ejecute lo mejor posible y utilice todos los recursos, núcleos en este caso, que las nuevas CPUs ofrecen. Es por ello que también resulta importante abordar el aspecto de la programación paralela.

En este capítulo se da primeramente una idea de la organización de los computadoras paralelas y sus clasificaciones. Se definen y analizan los dos modelos básicos de computadoras paralelas: el modelo de memoria compartida y el modelo memoria distribuida. Se abordan los paradigmas de programación para ambos modelos, enfocándose principalmente en el que compete a la programación de Unidades de procesamiento Gráfico, en este caso CUDA.

2.2. Clasificación de computadoras paralelas

En esta tesis se citan dos de las clasificaciones más reconocidas: La clasificación de Hwang-Briggs y la de Flynn.

2.2.1. Clasificación de Hwang-Briggs

K. Hwang y F. A. Briggs [Hwang & Briggs, 1988] establecen una primera aproximación a las clases de computadoras paralelas, fijando tres configuraciones básicas: computadoras pipeline, computadoras matriciales y sistemas multiprocesadores. De esta forma se distingue entre computadoras basadas en paralelismo temporal, que son aquellas en los que solapan varias operaciones en el mismo instante de tiempo pero en una misma unidad funcional, y paralelismo espacial, que supone la existencia de elementos de proceso replicados (Figura 2.1).

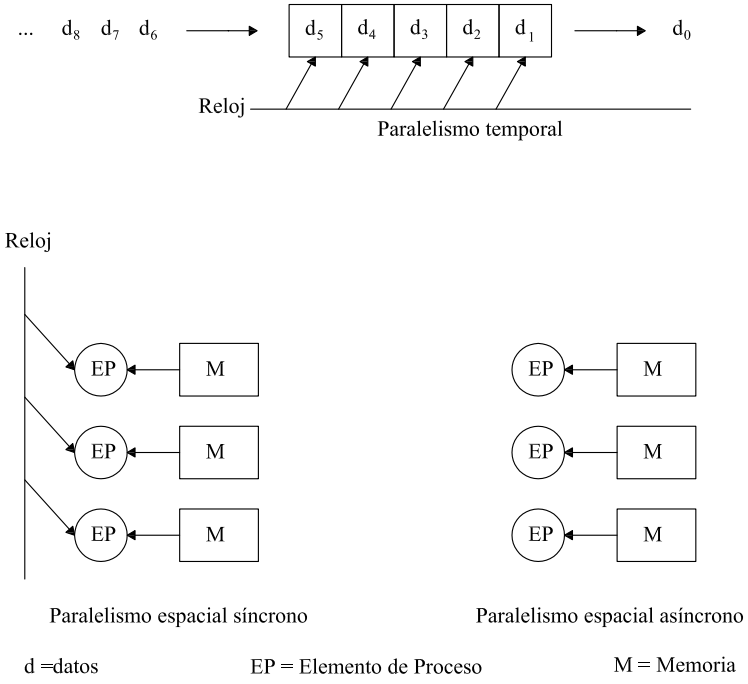


Figura 2.1: Clasificacion de Hwang-Brigss

En realidad la clasificación nos muestra las dos formas básicas de paralelismo, segmentación y réplica. Después se utiliza el concepto de sincronismo para diferenciar entre computadoras con paralelismo espacial síncrono (todos los elementos de proceso ejecutan sus instrucciones a una señal de un reloj único), y computadoras con paralelismo espacial asíncrono (cada elemento de proceso ejecuta su propias instrucciones sin una señal global de sincronización).

2.2.2. Clasificación de Flynn

Quizá la clasificación de computadoras paralelas más conocida y citada en la literatura sea la clasificación de Michael J. Flynn [Flynn, 1972]. En ella se utiliza la ausencia o presencia de un control global que permite regular el flujo de instrucciones de una computadora. Así se puede tener un flujo único (S= *Single*) o múltiple (M= *Multiple*) de instrucciones, ejecutándose sobre un conjunto único (S= *Single*) o sobre varios (M= *Multiple*) conjunto de datos. Con estas consideraciones, Flynn clasifica los sistemas en cuatro categorías (Figura 2.2):

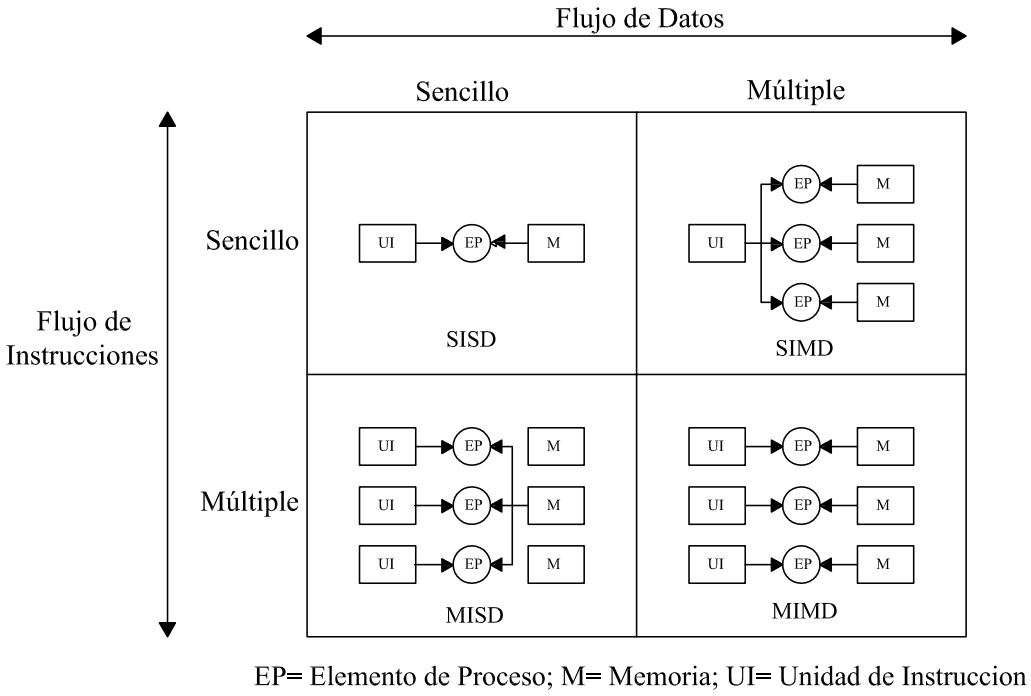


Figura 2.2: Clasificacion de Flynn

SISD (*Single Instruction stream, Single Data stream*) Flujo de instrucciones único y flujo único de datos. A esta clase pertenecen, evidentemente, las computadoras secuenciales o con arquitectura Von Neumann, en donde, en cualquier momento, solo se está ejecutando una única instrucción. Prácticamente ninguna computadora puramente SISD se fabrica hoy en día ya que la mayoría de procesadores modernos incorporan algún grado de paralelización como es la segmentación de instrucciones o la posibilidad de lanzar dos instrucciones a un tiempo (superescalares) [Quinn, 2004].

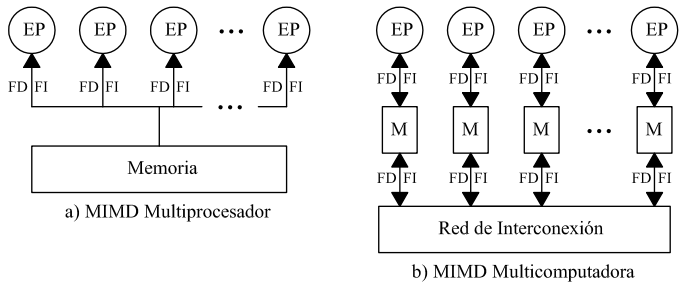
MISD (*Multiple Instruction stream, Single Data stream*) Flujo múltiple de instrucciones y único flujo de datos. Esta clasificación considera varias instrucciones que actúan sobre un único "paquete o segmento de datos". Sin embargo, el modelo MISD es considerada como una arquitectura impráctica ya que es difícil encontrar máquinas reales que respondan a este modelo [Pardo, 2002].

SIMD (*Single Instruction stream, Multiple Data stream*) Flujo de instrucción simple y flujo de datos múltiple. Esto significa que un flujo único de instrucciones se ejecuta sobre diferentes conjuntos de datos. Lógicamente es necesario replicar elementos de proceso para que las instrucciones se puedan ejecutar de forma simultánea. Las SIMD soportan procesamiento vectorial (matricial) asignando cada elemento del vector a una unidad funcional diferente para procesamiento concurrente. Se les llama también procesadores matriciales. En el mercado existen computadoras paralelas que pertenecen a esta clase, aunque actualmente no es común que se utilicen como computadoras de propósito

general [Almeida *et al.*, 2008].

MIMD (*Multiple Instruction stream, Multiple Data stream*) Flujo de instrucciones múltiple y flujo de datos múltiple. Corresponden a un ordenador con distintos elementos de proceso, donde cada uno de ellos ejecuta sus propias instrucciones de forma asíncrona [Almeida *et al.*, 2008]. Desde un punto de vista teórico representa el modelo más versátil de los cuatro ya que potencialmente ofrecen una mayor eficiencia en la ejecución concurrente o paralela. Aquí la concurrencia implica que no solo hay varios procesadores operando simultáneamente, sino que además hay varios programas (procesos) ejecutándose también al mismo tiempo. De cara a la resolución del problema, los diferentes procesos se tienen que comunicar y sincronizar, Esto se puede realizar mediante memoria compartida o por medio de una red de interconexión [Quinn, 2004].

Las computadoras paralelas MIMD que utilizan múltiples unidades funcionales o de procesamiento para acelerar los cálculos se les conoce como multiprocesadores (Figura 2.3a), mientras que los sistemas informáticos de procesamiento distribuido los cuales son conjuntos de ordenadores unidos por redes de alta velocidad de comunicación se les denomina multicomputadoras (Figura 2.3b) [Falcao, 1994].



EP= Elemento de Proceso; M= Memoria; FD= Flujo de Datos; FI= Flujo de Instrucciones

Figura 2.3: Arquitectura MIMD

De acuerdo a las ideas de Hwang y Briggs, los computadoras de la clase SIMD responderían a un paralelismo espacial síncrono, mientras que la clase MIMD (multiprocesador y multicomputadora) corresponderían a la filosofía de paralelismo espacial asíncrono [Almeida *et al.*, 2008]. Esta última clase de máquinas probablemente son las más comunes y utilizadas en la actualidad.

2.3. Multicomputadoras

En los multiprocesadores con memoria distribuida, llamados también multicomputadoras, cada elemento de proceso tiene su propia memoria local donde almacena sus datos, no existiendo una memoria global común. Una red de interconexión une los distintos elementos de proceso entre sí. La comunicación entre los procesadores se realiza mediante paso de mensajes.

El tiempo de ejecución de un algoritmo en este tipo de sistema paralelo depende fundamentalmente de dos componentes: uno debido al procesamiento de datos y otro debido a las comunicaciones. Las características de la red de interconexión condicionan mucho esta última componente.

La arquitectura de paso de mensajes muestra sus ventajas sobre los sistemas de memoria compartida cuando el número de procesadores es grande. Para sistemas multiprocesadores pequeños, los sistemas de memoria compartida presentaran probablemente un mejor rendimiento y mayor flexibilidad. La principal ventaja de esta arquitectura es que

es directamente escalable y presenta un bajo coste para sistemas grandes.

2.4. Multiprocesadores y Procesadores Multinúcleo

2.4.1. Multiprocesadores

Los multiprocesadores con memoria compartida (o simplemente multiprocesadores) son sistemas compuestos por varios procesadores completos, además disponen de una memoria común accesible mediante instrucciones de hardware. Existen dos posibilidades en cuanto la forma en que se realiza el acceso a la memoria en estas plataformas [Sanchez, 2010]:

- SMP (Symmetric Multiprocessor): El coste de acceso a cualquier posición de la memoria es uniforme.
- NUMA(Non-Uniform Memory Acces): La memoria está físicamente distribuida entre los procesadores, de modo que cada procesador dispone de una memoria local de acceso más rápido que el acceso a la memoria local de otro procesador.

Los multiprocesadores fueron una de las primeras arquitecturas paralelas que se diseñaron. Su fácil programación, basada en el paradigma de variables compartidas hace prever que seguirá siendo utilizada en los próximos años. Una de las principales críticas que se hace a los multiprocesadores es su escasa escalabilidad desde el punto de vista de hardware, ya que al incrementarse el número de procesadores, la memoria en el caso de SMP o la red de interconexión en las arquitecturas NUMA, se transforma en un cuello de botella. Sin embargo el número de procesadores que pueden soportar eficientemente estas plataformas es más que suficiente para muchas aplicaciones.

2.4.2. Procesadores Multinúcleo

Los procesadores multinúcleo (*multicore processors o chip multiprocessors*) incluyen en un solo chip muchas unidades de proceso autónomas, denominadas núcleos (*cores*). Cada núcleo es una unidad operacional completa, es decir, puede ser un procesador con técnicas sofisticadas de paralelismo y/o segmentación. Los problemas de disipación del calor y consumo de energía provocaron que a partir del 2005, los principales fabricantes de procesadores empezaran a incorporar diseños multinúcleo con la finalidad de obtener mayores rendimientos en sus productos [Sanchez, 2010].

Al día de hoy, los procesadores multinúcleo contienen un reducido número de núcleos (entre cuatro y ocho en la mayoría de los casos), pero se espera que se vea aumentada esta cantidad considerablemente en algunos años. Los procesadores multinúcleo, si bien pueden programarse del mismo modo que los multiprocesadores, poseen características propias que los hacen diferentes:

- Las tendencias de diseño apuntan a procesadores multinúcleo heterogéneos, con núcleos de distinta capacidad y/o naturaleza integrados en el mismo sistema.
- Las previsiones para los procesadores multinúcleo apuntan a cientos de núcleos en un chip, frente a los multiprocesadores que, en sus configuraciones comerciales más frecuentes, no superan los 16 o 32 procesadores.
- La organización habitual de los procesadores multinúcleo implica comunicaciones entre procesos mucho más eficiente (menor latencia, mayor ancho de banda y consumo más reducido) cuando los datos residen dentro del propio chip. Esta economía no es posible en los multiprocesadores y tampoco en los procesadores multinúcleo cuando los datos residen en niveles de memoria fuera del chip.

2.5. Plataformas operativas para el procesamiento en paralelo

Conforme a lo expuesto sobre sistemas Multiprocesador y Multimáquina, se pueden tener dos esquemas de programación: programación con variables compartidas y programación con paso de mensajes [Jin, 94].

En lo que respecta a los sistemas Multimáquina, los estándares más destacados de programación son PVM y MPI, mientras que para sistemas Multiprocesadores se encuentra Multithreading y OpenMP, finalmente para las Unidades de Procesamiento Gráfico que son un claro ejemplo de un procesador multinucleo, una de las plataformas más populares y sobresalientes es CUDA.

De acuerdo a lo planteado en el párrafo anterior se esquematiza en la Figura 2.4 las plataformas a ser analizadas.

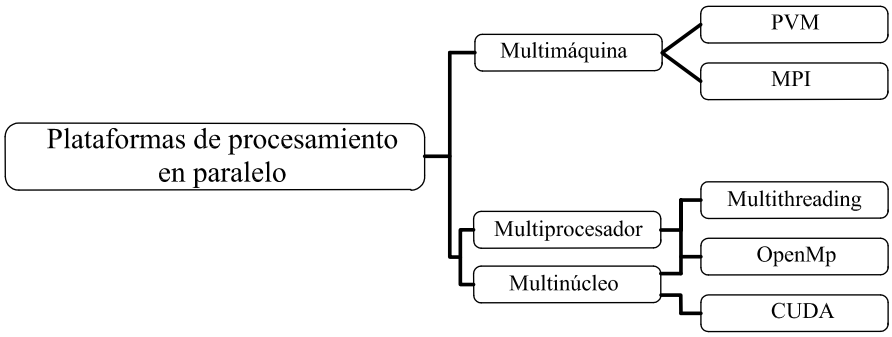


Figura 2.4: Plataformas de procesamiento en paralelo

2.5.1. PVM

La Máquina Virtual Paralela (PVM por sus siglas en inglés) es una herramienta de software que permite que una red generalmente heterogénea de computadoras y con sistema operativo bien sea Unix o Windows pueda trabajar como un solo ordenador paralelo [Geist *et al.*, 1994], es decir, compartan recursos de computo (como el procesador y la memoria RAM) con el fin de disminuir el tiempo de ejecución de un programa al distribuir la carga de trabajo en varias computadoras. El proyecto PVM comenzó en el verano de 1989 con la colaboración del Laboratorio Nacional de Oak Ridge y las universidades de Tennessee, Emory y Carnegie Mellon. Aunque se han desarrollado diferentes entornos de trabajo para paso de mensajes a lo largo de los años, PVM probablemente fue la primera biblioteca ampliamente adoptada y aceptada, debido a que proporciona un entorno de software portable para paso de mensajes en plataformas homogéneas y heterogéneas, y facilita una colección de rutinas de biblioteca que el usuario puede emplear en programas C o Fortran. Aunque la biblioteca surge en el ámbito de la computación sobre clúster de estaciones de trabajo, su uso se extiende a todo tipo de plataformas paralelas, en parte debido a su disponibilidad sin coste ya que puede obtenerse libremente en la red.

Esta plataforma se encuentra diseñada para vincular los recursos de cómputo de una red, independientemente del número de computadoras y su ubicación, por lo que es capaz de entregar altos niveles de rendimiento y funcionalidad [Almeida *et al.*, 2008].

Un programa en PVM es un conjunto de tareas que cooperan entre sí. Las tareas van a intercambiar información empleando paso de mensajes. La información es transferida con un protocolo de empaquetamiento-envió-recepción-desempaquetamiento [Medina *et al.*, 2006], la intención es proteger la información. Toda tarea en PVM puede incluir o eliminar máquinas, iniciar o parar otras tareas, mandar datos a otras tareas o sincronizarse con ellas [Mora, 2008].

2.5.2. MPI

MPI ("*Message Passing Interface*", Interfaz de Paso de Mensajes) es un estándar para la implementación de sistemas de paso de mensajes ampliamente utilizado en computadoras paralelas, especialmente en computadoras con memoria distribuida y en redes de estaciones de trabajo [Snir *et al.*, 1999]. La interfaz fue desarrollada por el MPI *Forum*, un grupo formado por investigadores de universidades, laboratorios y empresas involucrados en la computación de altas prestaciones. El propósito del desarrollo de dicha plataforma era contar con un esquema estándar que comprendiese las mejores características de varios sistemas de paso de mensajes que han sido desarrollados a lo largo de los años y mejorarlos donde fuese necesario.

Desde que fue completado en Junio 1994, MPI ha ido creciendo tanto en aceptación como en uso, considerándose actualmente como el estándar de paso de mensajes por una gran parte de la comunidad científica [Almeida *et al.*, 2008].

Se han realizado implementaciones en una gran variedad de plataformas que van desde máquinas masivamente paralelas hasta redes de estaciones de trabajo. Incluso, grandes compañías que fabrican Supercomputadoras incluyen a MPI como una de sus herramientas fundamentales [Hidrobo & Hoeger, 1998]. Además se puede encontrar muchas implementaciones gratuitas que pueden descargarse de internet.

El objetivo principal de MPI es lograr la portabilidad a través de diferentes máquinas, tratando de obtener un grado de portabilidad comparable al de un lenguaje de programación que permita ejecutar, de manera transparente, aplicaciones sobre sistemas heterogéneos; objetivo que no debe ser logrado a expensas del rendimiento.

Este modelo de programación también puede ser utilizado en arquitecturas de memoria compartida utilizando las bibliotecas que provee el estándar MPI. En estas arquitecturas las implementaciones de MPI, no utilizan una red de comunicaciones entre tareas, en su lugar, y por razones de rendimiento utilizan copias de memoria. Sin embargo, el uso de estas bibliotecas introduce grandes latencias que perjudican el rendimiento de la aplicación. Por esta razón, la plataforma de utilización de este estándar, está dirigida a sistemas de memoria distribuida [Quinn, 2004].

El estándar MPI define una biblioteca que puede ser utilizada para desarrollar programas portables de paso de mensajes utilizando lenguaje C, C++ o Fortran. Los programas se compilan utilizando los compiladores habituales y se enlazan con la librería MPI. Una comparación muy completa entre PVM y MPI puede ser vista en [Geist *et al.*, 1996].

2.5.3. Multithreading

La palabra Multithreading (MT) puede ser traducida como múltiples hilos de control o múltiples flujos de control. Mientras que un proceso tradicional UNIX siempre contiene un simple hilo de control, Multithreading separa un proceso en muchos hilos de ejecución, cada uno de los cuales se ejecuta por separado [Kleiman *et al.*, 1992]. Un hilo es un flujo de instrucciones independientes. En el ambiente de UNIX, un hilo existe dentro de un proceso y utiliza los recursos del proceso. Sin embargo, un hilo tiene su flujo de control propio e independiente. Pueden existir múltiples hilos dentro de un proceso. El concepto de un "procedimiento" el cual se ejecuta independientemente dentro de un proceso, puede describir de mejor manera el concepto de hilo [Ramos, 2007].

2.5.3.1. Concurrencia y paralelismo

Una ejecución concurrente se presenta cuando por lo menos 2 hilos o procesos están en progreso al mismo tiempo. Por ejemplo, un sólo procesador puede estar trabajando con múltiples tareas o procesos, pero la velocidad de conmutación entre los hilos es tan rápida que provoca que aparentemente todas las tareas se estén realizando de forma

simultánea. Por otro lado, si cada una de las tareas es ejecutada de manera independiente por un procesador, se habla entonces de ejecución paralela. La Figura 2.5 ejemplifica lo antes mencionado.

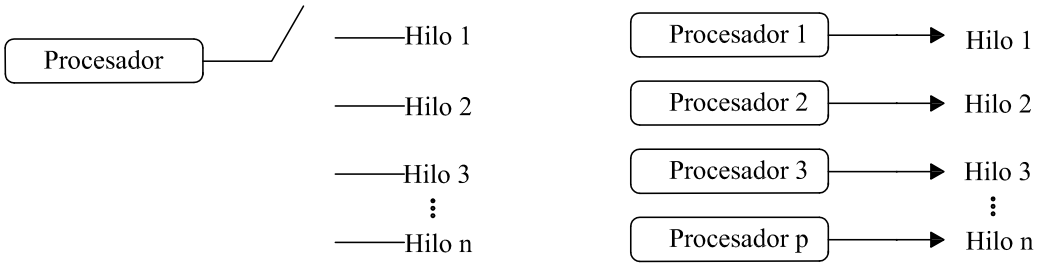


Figura 2.5: a)Ejecución concurrente y b) Ejecución paralela de p hilos

2.5.3.2. Estructura de MT

El UNIX tradicional soporta el concepto de hilos – cada proceso contiene un hilo, así que programar con múltiples procesos es programar con múltiples hilos. Pero un proceso es también un espacio de dirección, y crear un proceso involucra la creación de un espacio de dirección nuevo [Sun Microsystems, 1994].

La creación de un hilo es mucho menos costosa comparada con la creación de un proceso nuevo, debido a que los hilos de reciente creación utilizan el espacio de dirección del proceso actual. El tiempo que se ocupa para la conmutación entre hilos es mucho menor que el tiempo que se toma en la conmutación entre procesos, en parte debido a que la conmutación entre hilos no involucra la conmutación entre espacio de direcciones. La comunicación entre hilos de un proceso es simple debido a que los hilos comparten todo, en particular el espacio de direcciones. De tal forma que los datos producidos por un hilo están disponibles inmediatamente para todos los otros hilos [Ramos, 2007].

La programación multihilo es una herramienta poderosa. En maquinas monoprocesador, mediante su uso se consigue un mayor rendimiento efectivo pero un menor rendimiento computacional. Esto básicamente quiere decir que el sistema esta ocioso una menor parte del tiempo pero que el número de instrucciones útiles ejecutadas por unidad de tiempo desciende, ya que al propio proceso de los hilos hay que sumar el tiempo de conmutación entre ellos.

2.5.4. OpenMP

OpenMP (*Open Multi-Processing*) es una API (*Application Program Interface*) que puede ser utilizada para diseñar explícitamente programas paralelos *multithread* en memoria compartida. Básicamente, un programa en OpenMP es un programa secuencial con directivas OpenMP dispuestas en los puntos apropiados [Almeida *et al.*, 2008].

La meta de OpenMP es proveer una estándar y portable API para escribir programas paralelos enfocados a sistemas multiprocesador [Chandra *et al.*, 2001].

OpenMP está compuesto de tres elementos:

- Las directivas de compilación
- Las rutinas de librería (*Runtime Library*).
- Las variables de entorno

Estos tres elementos en conjunto forman lo que se denomina una interfaz de aplicación de programa o API por sus siglas en ingles.

La API ha sido especificada para C/C++ y Fortran y extendida a muchas otras plataformas. Ha sido definida conjuntamente y de forma colaborativa entre partes interesadas de la industria del hardware y del software, de entornos gubernamentales y del mundo académico. Se espera que posteriormente se convierta en un estándar ANSI [Almeida *et al.*, 2008].

En contraste a previas API, OpenMP facilita el incremento de la aproximación a la paralelización de un programa secuencial. El programador puede añadir directivas de paralelización a ciclos o declaraciones en el programa [Sato, 2002].

Las especificaciones de OpenMP son diseñadas por el ARB (*Architectural Review Board*), cuya misión es la de estandarizar APIs para el multiprocesamiento en memoria compartida [Almeida *et al.*, 2008].

La implementación de OpenMP utiliza *Multithreading* para obtener el paralelismo, mediante un modelo *fork-join,* Figura 2.6. De este modo, al encontrarse una directiva paralela que así lo indique, el *thread* maestro se divide en un número determinado de threads esclavos que se ejecutan concurrentemente y las tareas se distribuyen entre ellos. Dependiendo de los parámetros indicados, carga de la máquina y otros factores, el entorno de ejecución asignará uno o varios de los thread esclavos a los distintos procesadores disponibles [Dagum & Menon, 1998].

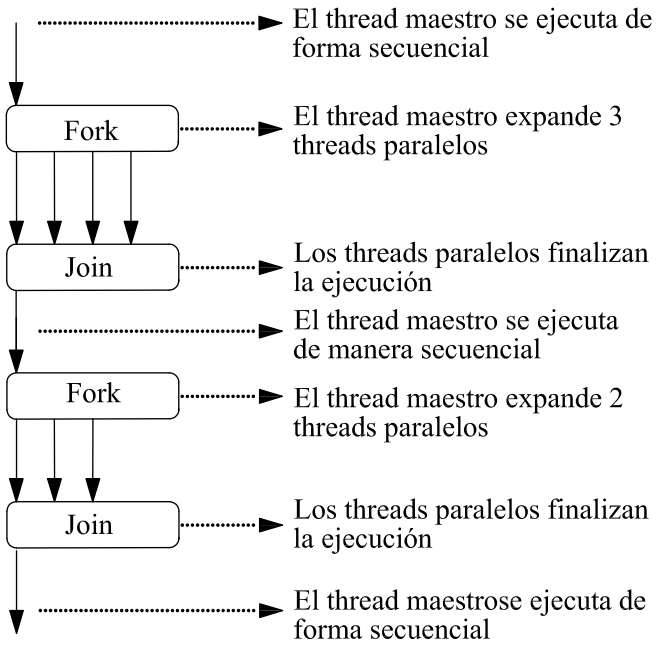


Figura 2.6: Modelo de Programación de OpenMP(*Fork-Join*)

El modelo de OpenMP se destaca por ofrecer a los usuarios una interfaz simple y sencilla para desarrollar aplicaciones paralelas en un amplio rango de arquitecturas de memoria compartida. Esta simplicidad se debe, en gran medida, al uso de las directivas de compilador que permiten partir de un código secuencial y paralelizarlo sin necesidad de realizar grandes cambios: basta insertar las directivas para marcar aquellas regiones susceptibles de ser paralelizadas, lo cual también reduce considerablemente el tiempo de depuración al disminuir las posibilidades de introducir nuevos errores en el código, si se parte de un programa secuencial correcto. De este modo, no es necesario lidiar con paso de mensajes ni con las estructuras de datos, función realizada automáticamente por las directivas.

En http://www.openmp.org pueden encontrarse los documentos y archivos oficiales de las especificaciones OpenMP

del ARB. Estas especificaciones están libremente disponibles y no requieren licencia para su uso.

2.5.5. Unidades de Procesamiento Gráfico

Las Unidades de Procesamiento Gráfico (GPUs por sus siglas en inglés), originalmente fueron desarrolladas por la necesidad de procesar de manera más eficiente gráficos de alta definición, requeridos en la industria de la animación y video juegos, y han llegado a convertirse probablemente, en una de las herramientas de hardware más poderosas instaladas en un ordenador de propósito general [Gopal *et al.*, 2007]. El rápido avance, tanto en la programabilidad de los procesadores gráficos como en su flexibilidad, ha permitido utilizarlos para resolver un amplio rango de complejos problemas con altas necesidades computacionales. Es lo que se conoce como GPGPU (*General-Purpose Computing on the GPU*) [Owens *et al.*, 2008].

2.5.5.1. Arquitectura de una GPU moderna

La Figura 2.7 muestra la arquitectura típica de una GPU actual. Está compuesta por un número escalable de multiprocesadores paralelos (SMs, por sus siglas en inglés). Estos multiprocesadores se agrupan de tres en tres (o de dos en dos en arquitecturas más antiguas) en lo que se llama Clúster de Procesado de Hilos (TPC, por sus siglas en inglés). El número de SMs varía desde las arquitecturas más antiguas, hasta las más modernas y de mayor gama [Sanchez, 2010].

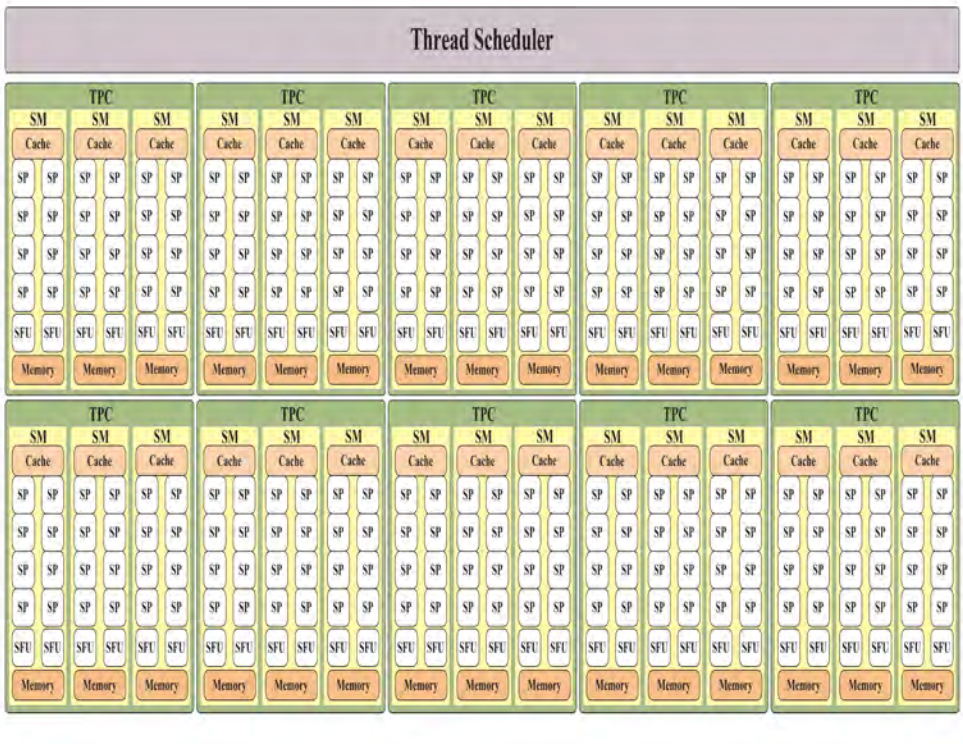


Figura 2.7: Arquitectura GPU modelo GTX 280 [Marandi & Dinavahi, 2009]

El diseño interno de cada SM es similar para todas las versiones, cada SM cuenta con 8 procesadores escalares

(SPs, por sus siglas en ingles), lo que hace un total de 240 núcleos en el modelo esquematizado en la Figura 2.7; 2 Unidades Especiales (SFUs, por sus siglas en inglés), capaces de realizar cálculos trascendentales (e.g. seno, coseno) y ejecutar operaciones en punto flotante [Marandi & Dinavahi, 2009]. También cuenta con una unidad de multiplicación y suma (MAD, por sus siglas en ingles) y una unidad adicional de multiplicación (MUL, por sus siglas en ingles) [Sanchez, 2010]. Los ocho procesadores de un multiprocesador comparten la unidad de búsqueda y lanzamiento de instrucciones de forma que se ejecuta la misma instrucción al mismo tiempo en los ocho procesadores.

La capacidad de cómputo de los procesadores varía en función del modelo, pero suelen funcionar a 1.3Ghz en la mayoría de modelos [Sanchez, 2010].

Desde el punto de vista de la memoria, existen seis tipos de recursos que los dispositivos nos ofrecen para almacenar información, y el ámbito de visibilidad de los mismos: registros (*registers*), memoria local (*local memory*), memoria compartida (*shared memory*), memoria constante (*constant memory*), memoria de texturas (*texture memory*) y memoria global (*global memory*). Estos elementos se muestran en la Figura 2.8.

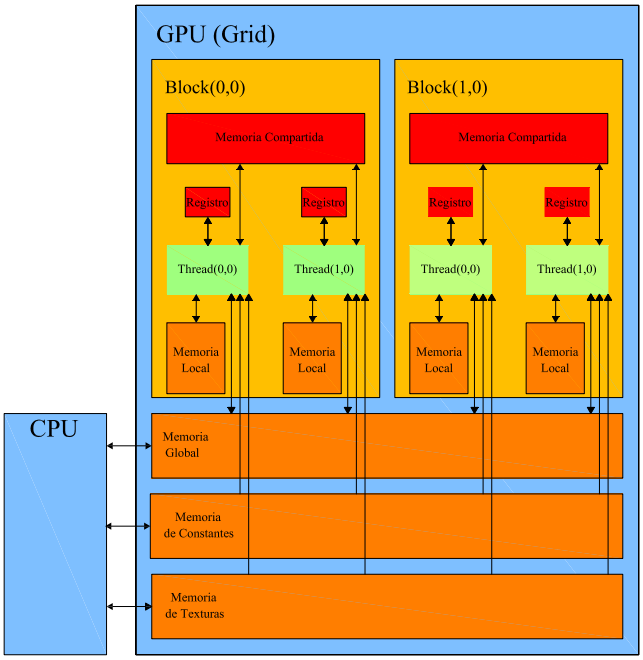


Figura 2.8: Modelo de memoria de una GPU.

De la Figura 2.8 las flechas indican el tipo de acceso, lectura o escritura o ambos. También indican los dispositivos o las unidades que pueden acceder a cada tipo.

La **memoria global** es la memoria más abundante, pero también la que se caracteriza por una mayor latencia y menor ancho de banda. Si la comparamos con una CPU, sería equivalente a la memoria RAM. Se encuentra fuera del chip de procesamiento y es el medio principal para comunicar datos entre el *host* y el dispositivo. Esta comunicación, por defecto, debe ser manualmente especificada por el programador. El tiempo de vida de los datos es de todo el programa, y estos son visibles por cualquier thread de cualquier bloque que se encuentre en ejecución en el dispositivo.

La **memoria local** es una manera alternativa de utilizar memoria global, acotando la visibilidad y el tiempo de vida de los datos a nivel de thread y de *kernel* respectivamente.

Los **registros** son el recurso de almacenamiento más rápido de las GPUs, junto con la memoria compartida. Se

encuentran dentro del chip de procesamiento y su visibilidad y tiempo de vida son similares a los de la memoria local.

La **memoria compartida** es uno de los recursos más valiosos del dispositivo. Se encuentra dentro del chip y, a diferencia de los registros y la memoria local, su visibilidad es a nivel de bloque. Esto quiere decir que todos los *threads* del mismo bloque pueden compartir información utilizando este medio. Además, su tiempo de vida es a nivel de *kernel* y su latencia puede equipararse a la de los registros, en el mejor de los casos.

La **memoria constante** es un tipo de memoria dentro del dispositivo, cuya función es ofrecer un nivel de caché por encima de la memoria global. Como su nombre indica, se trata de una memoria de sólo lectura a nivel de GPU, por lo que los datos deberán ser transferidos/asociados a ella desde el *host*, antes de proceder con la ejecución de un *kernel* que acceda a esos elementos. Su visibilidad y tiempo de vida son similares a los de la memoria global, pero hay que destacar que no guarda coherencia con la misma.

La **memoria de texturas** es utilizada nativamente para leer texturas de imágenes en aplicaciones gráficas. Podemos decir que esta memoria pretende suplir las mismas funcionalidades que la memoria constante (nivel de caché sobre memoria global), pero con restricciones de acceso diferentes.

La Tabla 2.1muestra un resumen de lo descrito sobre las memorias de dispositivos.

Cuadro 2.1: Memorias en una GPU

Memoria	Localización	Accesos a GPU	Visibilidad	Tiempode vida
Registros	Dentro del chip	R/W	Thread	Kernel (Thread)
Local	Fuera del chip	R/W	Thread	Kernel (Thread)
Compartida	Dentro del chip	R/W	Bloque	Kernel (Bloque)
Global	Fuera del chip	R/W	host+device	Programa
Constante	Caché dentro del chip	R	host+device	Programa
Texturas	Caché fuera del chip	R	host+device	Programa

A nivel global, las tarjetas graficas cuentan hasta con 6 GB de memoria dedicada, en los nuevos Modelos Fermi [http://www.nvidia.com]. Actualmente la comunicación de la GPU con la CPU se realiza a través de un bus PCI-Express.

La ejecución de los hilos en la GPU no se lleva a cabo de forma independiente. El planificador de hilos mostrado en la Figura 2.9 planifica y distribuye bloques de hilos entre los SM. Los *threads* son asignados a los SMs en grupos llamados bloques (máximo 8 bloques por SM).

El multiprocesador crea, gestiona y ejecuta hilos concurrentes en el *hardware* sin sobrecoste de planificación o cambios de contexto. Mapea cada hilo a un procesador, y cada hilo se ejecuta de forma independiente con su propia dirección de instrucción y registros de estado. Este nuevo modelo de ejecución se llama SIMT (*Single Instruction Multiple Thread*). Esta arquitectura, a diferencia de la SIMD, permite que *threads* paralelos puedan seguir diferentes caminos en caso de existir instrucciones condicionales [www.nvidia.com].

En primer lugar, el multiprocesador divide los bloques de hilos en grupos de 32 hilos llamados warp. Al lanzar una nueva instrucción, la unidad de planificación, selecciona un warp disponible y lanza la misma instrucción para todos los hilos de ese warp[www.nvidia.com].

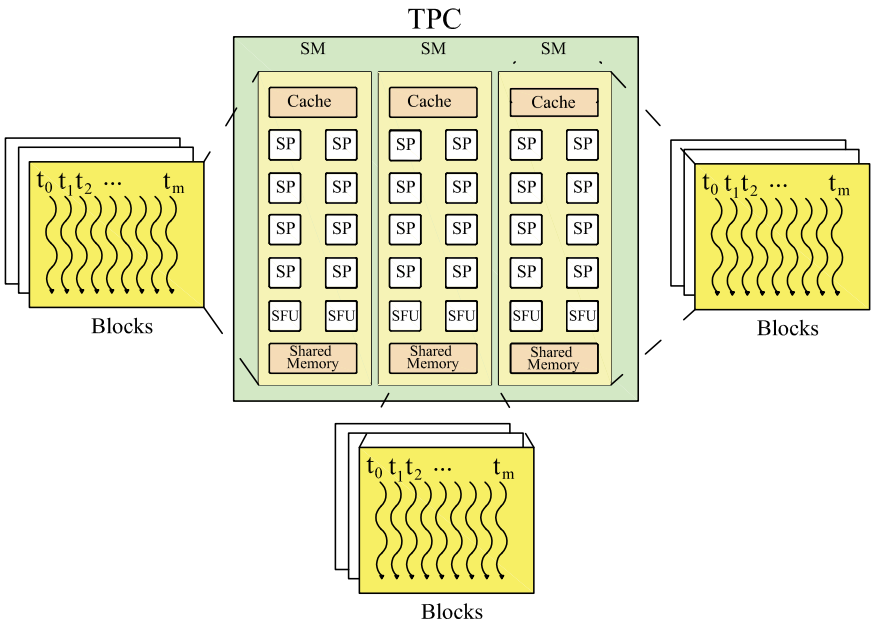


Figura 2.9: Asignación de bloques a los SMs (Streaming Multiprocessors) de un TPC.

Ya que hay 8 SPs por SM, se tardan cuatro ciclos en ejecutar una instrucción de un *warp*. Las instrucciones de salto suponen un problema ya que hilos de un mismo *warp* pueden tomar caminos de ejecución distintos. En este caso, la ejecución se serializa, ejecutando primero los hilos de un camino y después los hilos del otro. Además existen instrucciones para sincronizar todos los hilos de un mismo bloque, haciendo que *warps* enteros detengan su ejecución hasta que todos los *warps* del bloque alcancen el mismo punto de ejecución [Sanchez, 2010].

Las nuevas GPU de NVIDIA llamadas Fermi, esta implementada con millones de transistores, cuenta con hasta 512 núcleos CUDA organizados en 16 SM de 32 núcleos cada uno [Nickolls & Dally, 2010].

Cada núcleo de estas nuevas tarjetas es capaz de ejecutar una instrucción de punto flotante o entero por ciclo de reloj de un hilo. A diferencia de arquitecturas anteriores es capaz de ejecutar varios kernels en paralelo añadiendo otro planificador, tiene además un sistema de detección de errores, una nueva memoria cache para implementar algoritmos que no podían hacer un uso eficiente de la memoria compartida, mayor memoria compartida y global, etc. Además esta arquitectura permite una transferencia de datos bidireccional. En general la arquitectura Fermi proporciona la capacidad de un superordenador por tan solo una decima parte del coste y una vigésima parte del consumo de energía de los tradicionales servidores basados en CPUs [Sanchez, 2010].

Si desea verse más información sobre el funcionamiento de una GPU en [www.nvidia.com].

2.5.5.2. Modelo de programación CUDA

Actualmente existen distintas interfaces de programación para una GPU, pero dependiendo de la aplicación, pueden ser divididas en dos categorías principales: programación enfocada al procesamiento de gráficos y programación especializada para facilitar el desarrollo de aplicaciones de propósito general GPGPU [Marandi & Dinavahi, 2010], como lo es CUDA (*Compute United Device Architecture*) [www.nvidia.com]. El lenguaje de programación CUDA provee una sintaxis muy similar al lenguaje C, que incluye un set de directivas de compilación y llamadas a una Interfaz de Programación de Aplicaciones (API, por sus siglas en inglés) que permiten programar toda la gama de tarjetas graficas

NVIDIA. La (Figura 2.10) esquematiza el modelo de programación CUDA.

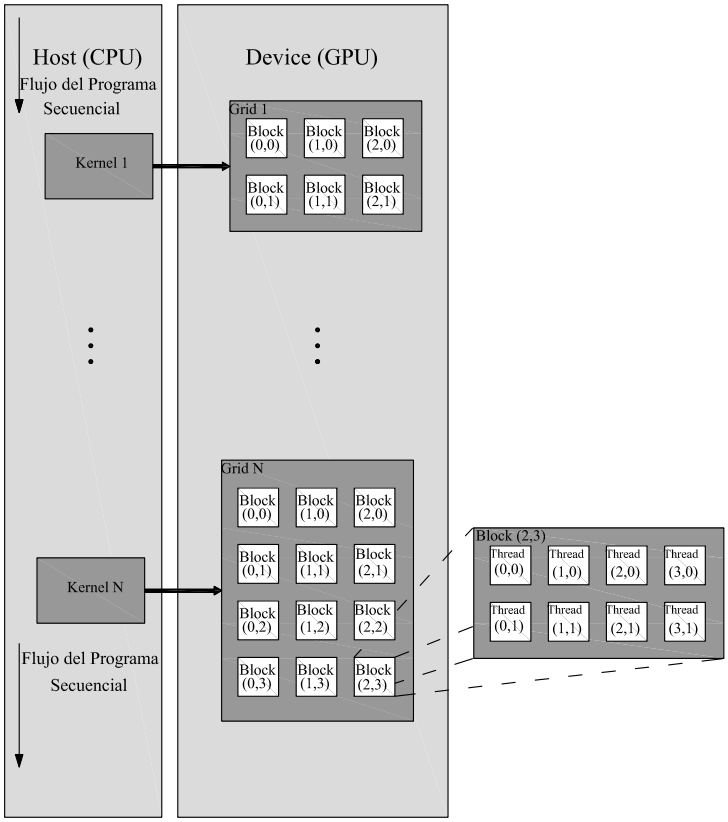


Figura 2.10: Modelo de programamción CUDA NVIDIA.

El modelo de programación de CUDA, supone que los hilos CUDA se ejecutan en una unidad física distinta, que actúa como coprocesador (*device*), al procesador (*host*) donde se ejecuta el programa. En un programa CUDA básicamente se especifican instrucciones que declaran la ejecución de código en paralelo y estructuras de datos a ser procesadas en el device. Estas instrucciones son llamadas *kernels*.

Un *kernel* es considerado como una función que describe el trabajo de un simple *thread* y pueden ser ejecutadas por cientos de ellos. Hay algunas restricciones en los *kernels*, por ejemplo, no pueden ejecutarse llamadas recursivas, las variables estáticas no pueden ser declaradas y el número de argumentos no puede ser variable [Ibarrola *et al.*, 2010]. Un programa completo en CUDA tiene diferentes fases a ser ejecutadas en la CPU o en el GPU. Cuando la fase tiene bajo o cero paralelismo es recomendable que sea asignada a la CPU. Pero de lo contrario, si la fase exhibe paralelismo resulta conveniente ser ejecutada en la GPU en forma de *kernel* [Ibarrola *et al.*, 2010].

El código host en CUDA, utiliza una sintaxis con funciones específicas para invocar el *kernel*. Cuando un *kernel* es invocado, bloques con igual número de *threads* se crean para ejecutar una tarea paralela [www.nvidia.com]. Estos bloques o vectores pueden ser de 1, 2 o 3 dimensiones. Los *threads* son las unidades más pequeñas de procesamiento en una GPU. Todos los *threads* en un bloque ejecutan la misma instrucción en un multiprocesador de flujo (SM), lo cual da a la GPU una arquitectura SIMD (*Single Instruction-Multiple Data*) [Marandi & Dinavahi, 2010]. Los hilos del mismo bloque pueden cooperar entre sí, compartiendo datos y sincronizando sus ejecuciones a través del comando *syncthreads*. Sin embargo, la comunicación entre *threads* que pertenecen a diferentes bloques de *threads* solo puede

realizarse haciendo uso de la memoria global, haciendo la comunicación lenta e inflexible [Sanchez, 2010].

La organización de los bloques de *threads* es una clave para el diseño de un algoritmo en una GPU, ya que todos los bloques de *threads* deben ser capaces de funcionar de forma totalmente independiente uno de otro. Bloques de *threads* forman un arreglo conocido como *Grid*, la cual puede ser de 1 o 2 dimensiones. Cada bloque en una *Grid* y cada *thread* en un bloque son identificados por índices individuales, que los hacen accesibles a través de comandos incorporados en CUDA [www.nvidia.com].

Como ejemplo, el siguiente código muestra como se define un *kernel* y como se llama desde un programa (Algoritmo 2.1):

Algoritmo 2.1 Ejemplo de Construcción de un Kernel

```
__global__ void vecAdd(floatA, floatB, floatC)
  {
     int i = threadIdx.x;
      C[i] = A[i] + B[i];
  }
intmain( )
  {
...
//Invocacion del Kernel con N threads
      vecAdd <<< 1, N >>> (A,B,C);
  }
```

Al comienzo y al término de un programa el *host* realiza una transferencia de los datos globales o el espacio de datos del GPU. El número de *Grids*, bloques e hilos afectan el rendimiento de las tareas, cada aplicación tiene una optima selección de estos parámetros. Como una buena regla, estos parámetros son determinados por experimentación [Ibarrola *et al.*, 2010].

Un *Kernel* se define utilizando el especificador de declaración *__global__*, los valores entre$<<< \cdots >>>$ que aparecen en código anterior se conocen como la configuración del *kernel*, y definen la dimensión del *Grid* y el numero de hilos de cada bloque. La Figura 2.11 muestra como se organizan los hilos en un *Grid* de 2×3 bloques y 3×4 hilos cada uno.

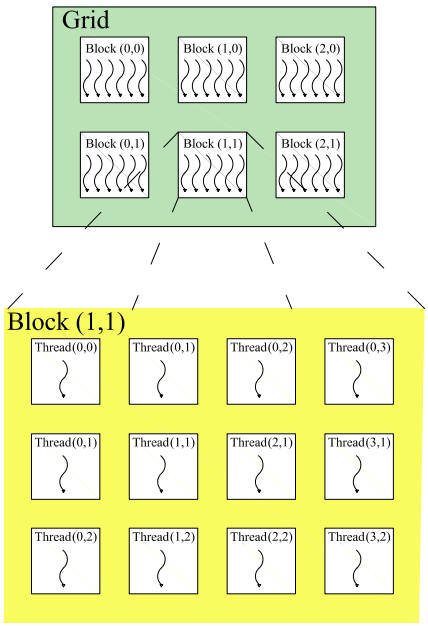


Figura 2.11: Grid de Bloques de hilos

Los hilos se organizan en dos niveles usando coordenadas para su indexación que se llaman "*blockId*" y "*threadId*". Estos valores son definidos por el sistema y accesibles a las funciones *kernel*. Todos los hilos de un mismo bloque tienen el mismo valor de "*blockId*". El rango de "*blockId*" viene definido por la dimensión de la malla o *Grid*. La organización de una malla se determina durante el lanzamiento del *kernel*. El primer parámetro especifica las dimensiones de una malla en función de los bloques que contiene. El segundo parámetro especifica las dimensiones de un bloque en función de los hilos que lo componen.

Por ejemplo:

dim3 dimBlock(4, 2, 2);

dim3 dimGrid(2, 2, 1);

KernelFunction<<<dimGrid, *dimBlock*,*>>>(...)*;

Este código define una malla bidimensional de 2x2 bloques y bloques tridimensionales de 4x2x2 hilos. Las dos primeras líneas inicializan las dimensiones y la tercera es la que arranca el *kernel* propiamente dicho. Lo esencial en cuanto a "*BlockId*" y "*ThreadId*" es indexar los hilos de manera unívoca y permitir así manejarlos de manera individualizada, concretamente, asignándoles los grupos de datos sobre los cuales operarán. Otro aspecto importante en la programación CUDA es la escalabilidad ya que es directa, es decir, el mismo código compilado puede correr en GPUs con diferente número de núcleos, tal que una GPU con mayor numero de núcleos y velocidad, ejecutará el programa en un menor tiempo (Figura2.12).

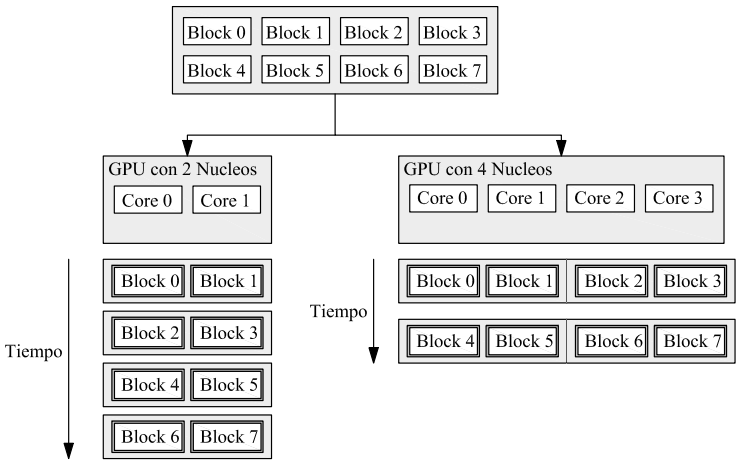


Figura 2.12: Escalabilidad de un Kernel CUDA en distintos dispositivos

2.5.5.3. Tipos de funciones

Los tipos de funciones, se utilizan para definir el tipo de función a aplicar:

- ***device*:** es una función que se ejecuta en la placa gráfica y se puede llamar solo desde la tarjeta.
- ***global*:** es una función *kernel*. Se ejecuta en el dispositivo y se puede llamar sólo desde un programa que corre en la CPU o *host*. Cualquier llamada a una función global debe especificar la configuración de ejecución (dimensiones de la malla y de los bloques).
- ***host*:** es una función que se ejecuta únicamente en la CPU y por lo tanto solo puede ser invocada desde código en ejecución en el CPU, del mismo modo que una función C tradicional.

2.5.5.4. Tipos de Variables

Los tipos de variables, especifican la locación de variable en la memoria del dispositivo.

- ***device*:** es una variable que reside en el espacio global de la memoria de la GPU. Es accesible desde todos los *threads* de la malla y desde el *host* a través de las bibliotecas, además tiene un tiempo de vida de la aplicación.
- ***shared*:** es una variable que se almacena en el espacio compartido de memoria de un bloque y es accesible únicamente desde todos los *threads* del bloque. El tiempo de vida de este tipo de variables es la del bloque mismo.
- ***constant*:** es un tipo de variable que reside únicamente en el espacio de memoria constante, tiene el tiempo de vida de la aplicación y es accesible por todos los *threads* de la *grid* y desde el *host* a través de las bibliotecas.

2.5.5.5. Administración de la memoria

Existen funciones específicas para la administración de la memoria del dispositivo:

- ***cudaMalloc(...)*:** ubica una cantidad específica de bytes de memoria consecutiva y retorna un puntero a ese espacio que puede utilizarse para cualquier tipo de variable.

- ***cudaMemcpy(. . .):*** copia bytes de un área de memoria a otra. Pudiendo copiar desde la memoria principal de la CPU a la memoria de la GPU, a la inversa y entre diferentes espacios de la memoria de la GPU.

- Ambas funciones solo pueden ser invocadas desde el código que es ejecutado en la CPU. Para más información se puede consultar el manual de CUDA [www.nvidia.com].

2.6. Diseño de algoritmos paralelos

Diseñar algoritmos paralelos no es tarea fácil y es un proceso altamente creativo. La mayoría de los problemas de programación tienen diferentes soluciones paralelas. La mejor solución puede diferir de la solución dada por los algoritmos secuenciales existentes. A continuación se describe una metodología para el diseño de algoritmos paralelos propuesta por Ian Foster [Foster., 1994]. Esta metodología está basada en cuatro procesos: Particionamiento, Comunicación, Aglomeración, Mapeo, los cuales se ilustran en la Figura 2.13 .

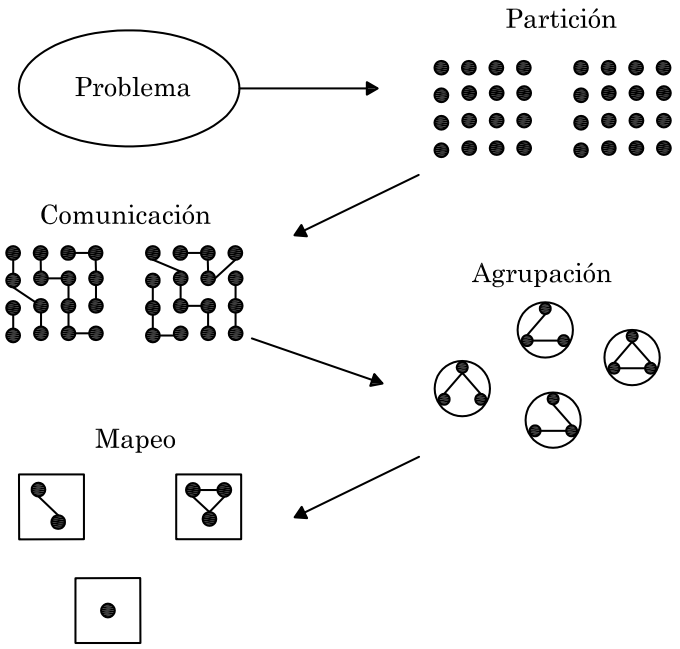


Figura 2.13: diseño de algoritmos paralelos

- **Partición:** El cómputo y los datos sobre los cuales se opera se descomponen en sub-tareas. Se ignoran aspectos como el número de procesadores de la máquina a usar y se concentra la atención en explotar oportunidades de paralelismo.

- **Comunicación:** La comunicación requerida para coordinar la ejecución de las tareas es determinada, al igual se definen una apropiada comunicación de las estructuras y algoritmos.

- **Agrupación:** El resultado de las dos etapas anteriores es evaluado en términos de eficiencia y costos de implementación. De ser necesario, se agrupan tareas pequeñas en tareas más grandes.

- **Mapeo:** Cada tarea es asignada a un procesador tratando de maximizar la utilización de los procesadores y de reducir el costo de comunicación. La asignación puede ser estática (se establece antes de la ejecución del programa) o a tiempo de ejecución mediante algoritmos de balanceo de carga .

2.7. Evaluación de algoritmos paralelos

En esta sección se presentarán algunos índices que se tienen en cuenta para analizar las prestaciones de los algoritmos paralelos.

2.7.1. Ganancia de velocidad (*Speed-Up*)

El *Speed-Up* para P procesadores S_P, es el cociente entre el tiempo de ejecución de un programa secuencial T_S y el tiempo de ejecución de la versión paralela de dicho programa en P procesadores T_P. Dado que puede haber distintas versiones secuenciales, se elige el T_S de la versión secuencial más rápida. Este índice indica la ganancia de velocidad que se ha obtenido con la ejecución en paralelo [Foster., 1994].

$$S_P = \frac{T_S}{T_P} \tag{2.1}$$

Por ejemplo, un *Speed-Up* igual a 2 indica que se ha reducido el tiempo a la mitad al ejecutar el programa con varios procesadores. En el mejor de los casos el tiempo de ejecución de un programa en paralelo con P procesadores será P veces inferior al de su ejecución en un sólo procesador, teniendo todos los procesadores igual potencia de cálculo.

Es por ello que el valor máximo que puede alcanzar el *Speed-Up* de un algoritmo paralelo es P. Generalmente el tiempo nunca se verá reducido en un orden igual a P, ya que hay que contar con la sobrecarga extra que aparece al resolver el problema en varios procesadores, debido a sincronizaciones y dependencias entre ellos.

2.7.2. Eficiencia

La eficiencia es el cociente entre el *Speed-Up* y el número de procesadores. Significa el grado de aprovechamiento de los procesadores para la resolución del problema. El valor máximo que puede alcanzar es 1, que significa un 100 % de aprovechamiento [Foster., 1994].

$$E = \frac{S_p}{N_P} \tag{2.2}$$

2.7.3. Escalamiento fuerte y la ley de Amdahl

El fuerte escalamiento es una medida de cómo para un problema general de tamaño fijo, el tiempo de solución disminuye a medida que se añaden más procesadores al sistema. Una aplicación que exhibe escalamiento lineal fuerte tiene una aceleración igual al número de procesadores utilizados [www.nvidia.com].

Un fuerte escalamiento generalmente se equipara con la Ley de Amdahl, que específica la máxima aceleración que se puede esperar por la paralelización de las porciones de un programa en serie. En esencia, establece que si un programa tiene una sección paralelizable, y otra sección la cual debe de ser ejecutada en forma secuencial, entonces el tiempo de ejecución del programa tendrá una aproximación asintótica hacia el tiempo de ejecución secuencial, esto

quiere decir que por más elementos de procesamiento que sean utilizados el tiempo de computo no decrecerá más [Lewis & Berg, 1998].

$$S_P = \frac{1}{(1-\gamma) + \frac{\gamma}{N_P}} \tag{2.3}$$

Aquí γ es la fracción del tiempo total de ejecución tomado por la porción del código que puede ser paralelizado y N_P es el número de procesadores sobre los cuales la sección paralela del código es ejecutada.

El comportamiento del tiempo de ejecución utilizando procesamiento en paralelo se puede observar en la Figura 2.14 .

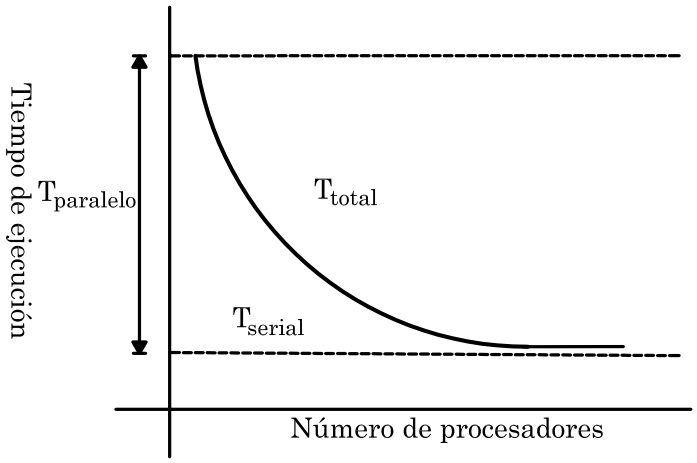


Figura 2.14: Ley de Amdahl

En la Figura 2.14 puede apreciarse que el uso de un mayor número de elementos de proceso contribuye a la disminución del tiempo total de ejecución, sin embargo, se observa que el tiempo total de ejecución no puede ser reducido indefinidamente debido a la limitante que es el tiempo secuencial del programa. Este tiempo surge de la dependencia de información del algoritmo a implementarse lo cual puede limitar el grado de paralelismo.

2.8. Conclusiones

En este capítulo se ha descrito de manera concisa los conceptos básicos sobre procesamiento en paralelo, se han abordado las arquitecturas y las plataformas, enfocándose principalmente en la plataforma CUDA que compete a la programación de tarjetas gráficas de la marca NVIDIA, de la cual, una clase de ellas es utilizada en esta tesis. De igual manera se han analizado las métricas, que son parte fundamental para evaluar la eficiencia que puede brindar un algoritmo paralelo.

Capítulo 3

Técnicas numéricas para determinar el EEP en el dominio del tiempo

3.1. Introducción

La simulación de circuitos eléctricos es una de las tareas primordiales en el diseño y planeación de los sistemas eléctricos de potencia. Su principal aplicación es la determinación del comportamiento de un sistema eléctrico con la finalidad de tomar las acciones y decisiones más pertinentes para que este opere en condiciones óptimas.

Recientemente, debido al incremento de la red y la adición de componentes y modelos no lineales, tales como dispositivos de electrónica de potencia y fuentes alternas de energía, así como la representación de fenómenos no lineales tales como la saturación magnética en transformadores, maquinas rotatorias, etc, ha ocasionado que el tiempo requerido de simulación y esfuerzo computacional para determinar el EEP en el dominio del tiempo se incremente. Así mismo, la distorsión armónica presente en las formas de onda de voltaje y corriente en los sistemas de potencia ha aumentado a consecuencia de la incorporación de dichos elementos.

La importancia de obtener el EEP radica en que diferentes tipos de estudios requieren el cálculo de este estado para poder ser realizados. Por ejemplo, en análisis de calidad de energía, el contenido armónico de una red eléctrica, solo puede ser determinado hasta haber conseguido dicha etapa [Segundo y Medina, 2010]. Otra aplicación práctica es para la inicialización de software de simulación como EMTP, que es utilizado para el análisis de transitorios electromagnéticos [Dommel *et al.*, 1995].

La solución en EEP puede ser determinada mediante distintas metodologías desarrolladas en el dominio de la frecuencia, dominio híbrido (frecuencia y tiempo) y dominio del tiempo [Kundert *et al.*, 1990]. Tanto en [Kundert *et al.*, 1990] y [Medina, 2001], Se detalla su aplicación en la obtención de la solución periódica en estado estacionario, al igual que se mencionan sus ventajas y limitantes en términos de eficiencia, necesidades computacionales y precisión.

En los siguientes apartados de este capítulo, se presentan las herramientas numéricas utilizadas para extrapolar las variables de estado de un sistema de potencia no lineal al Ciclo Límite en el dominio del tiempo. Se analiza la técnica Newton basada en Diferenciación Numérica propuesta en [Semlyen & Medina, 1995], de igual forma se describe una variante estratégica de iteración del método de Diferenciación Numérica propuesta en [Semlyen & Medina, 1995].

3.2. Determinación de la solución de estado estable en el dominio del tiempo de redes eléctricas

Convencionalmente, el EEP de un sistema de potencia no lineal, puede ser obtenido en el dominio del tiempo a través de la aplicación directa de un método numérico de integración de fuerza bruta (FB), tal como el Runge-Kutta o la regla trapezoidal, durante un lapso de tiempo suficientemente grande como para que el transitorio pueda ser despreciable [Parker & Chua, 1989]. Este procedimiento es usualmente muy lento, en especial en sistemas pobremente amortiguados [Dommel *et al.*, 1986]. Alternativamente, métodos Newton basados en el mapa de Poincaré [Parker & Chua, 1989] y extrapolación de las variables de estado al ciclo limite [Parker & Chua, 1989] se propone en [Semlyen & Medina, 1995], con la finalidad de eludir los inconvenientes de un método numérico de FB y así obtener una rápida solución en EEP en el dominio del tiempo de sistemas eléctricos no lineales.

Estos métodos de solución en el dominio del tiempo, se pueden utilizar para obtener el punto inicial del Ciclo Límite, independientemente de su estabilidad [Semlyen & Medina, 1995], además tienen la peculiaridad de poseer una característica de convergencia, generalmente cuadrática.

3.2.1. Métodos Newton para la obtención de una rápida solución al Estado Estacionario Periódico

Como se mencionó en el apartado anterior, una manera de obtener rápidamente el cálculo de estado estable en el dominio del tiempo de un sistema de potencia no lineal por extrapolación de las variables de estado al ciclo límite, puede lograrse mediante la aplicación de técnicas Newton, tales como: (AD, DN, ME) [Semlyen & Medina, 1995], (MED) [Segundo & Medina, 2010aa] y (DNM) [Segundo & Medina, 2010bb]. El objetivo fundamental de estas técnicas es obtener la solución en el dominio del tiempo de un sistema no lineal, cuyo comportamiento está descrito por un conjunto de EDO's alimentadas por una entrada periódica, en un mínimo de periodos de tiempo [Sigridt, 1998].

En esta Tesis se utiliza como método de acercamiento rápido al EEP, la técnica Newton de Diferenciación Numérica, ya que ha mostrado tener ventajas sobre otras metodologías, por ejemplo, para las alternativas de AD y ME tienen el inconveniente de requerir conocer la matriz Jacobiana, la cual usualmente puede ser obtenida analíticamente, pero no siempre es el caso, en especial en sistemas de potencia con componentes altamente no lineales o conmutados como los Sistemas Flexibles de Transmisión en CA o dispositivos FACTS [Medina *et al.*, 2006], por consiguiente, el método de DN al no precisar conocer la matriz Jacobiana, no resulta necesaria la determinación explícita de las derivadas parciales de los elementos de la matriz y la actualización de los elementos correspondientes a dispositivos no lineales y variables en el tiempo a lo largo del proceso de extrapolación [Segundo & Medina, 2008].

Por otro lado, la desventaja del método MED frente al método de DN radica en que la precisión de la solución obtenida en el método MED dependerá implícitamente del paso de integración, el cual tiene que ser mayor si se desea aumenar su presición, esto conlleva a un mayor esfuerzo computacional y por ende a un mayor tiempo de simulación comparado con el metodo de DN [Ortega, 2012].

El proceso de DN implica exclusivamente procesos de integración numérica además de un proceso de inversión matricial. La potencialidad y eficiencia de este método ha sido reportada en diversos artículos [Semlyen & Medina, 1995], [Garcia & Medina, 2003], [Medina *et al.*, 2003], [Medina *et al.*, 2006], por citar algunos.

3.2.1.1. Método Newton para el cálculo del EEP basado en el Mapa de Poincaré

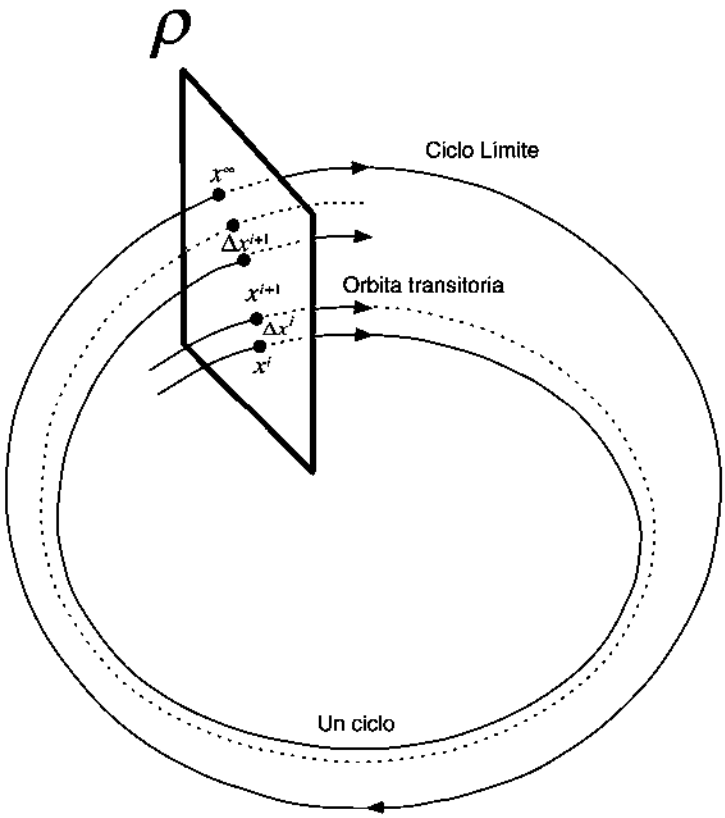


Figura 3.1: Plano de Poincaré

El comportamiento de componentes no lineales y cargas puede ser descrito en el dominio del tiempo por el siguiente sistema de EDO's no lineales,

$$\dot{x} = f(x,t); \qquad x_0 = x(t_0) \tag{3.1}$$

Donde x y f son vectores de dimensiones n, y $f(x,t)$ es una función continua en un rango de t. si la función (3.1) es periódica, entonces la solución en estado estable $x(t)$ es también periódica y puede ser representada como el Ciclo Límite para x_k en términos de otro elemento periódico de x o en términos de una función T-periódica, por ejemplo $sen(\omega t)$ [Semlyen & Medina, 1995], tal y como se muestra en la Figura 3.1, la cual representa esquemáticamente una órbita transitoria asociada a un sistema continuo y periódico a través de un hiperplano imaginario que corta perpendicularmente a los ciclos de dicha órbita. A este plano se le conoce como Plano de Poincaré (ρ) [Parker & Chua, 1989]. Otras variables tales como $i(t)$ se obtienen de ecuaciones de salida algebraicas [Semlyen & Medina, 1995].

Antes de alcanzar el Ciclo Límite, los ciclos de una órbita transitoria están cercanos a éste. Su posición está apro-

piadamente descrita por su representación sobre el plano de Poincaré ρ. Un ciclo sencillo mapea su punto de inicio x_i a su punto final x_{i+1} y mapea un segmento de perturbación Δx_i (desde este ciclo base [Semlyen y Medina, 1995]) a Δx_{i+1}, tal y como se aprecia en la Figura 3.1 , todos los mapeos cerca de un ciclo limite son quasi-lineales así que el método Newton o su aproximación pueden ser usados para obtener el valor de las variables de estado al Ciclo Límite x_∞. Esto es independiente de su estabilidad [Semlyen & Medina, 1995].

3.2.1.2. Método Newton de Diferenciación Numérica

La técnica Newton de DN está clasificada como un método de disparo tales como los propuestos por [Aprille & T. N. Trick, n.d.] [Nakhla & Branin, 1977] [Skelboe, 1980] [Semlyen & Medina, 1995]. El objetivo de estas técnicas, como ya se mencionó con anterioridad, es encontrar una condición inicial del vector $x(0)$ tal que cuando se integre el sistema de ecuaciones $x = f(x,t)$, sobre un periodo completo de tiempo T, a partir de la condición inicial $x(0)$ se obtenga $x(T) = x(0)$. El trabajo presentado por [Aprille & T. N. Trick, n.d.] es uno de los primeros en enfocarse a encontrar el vector $x(0)$ mediante un método Newton-Raphson. Al igual, en la contribución realizada por [Nakhla & Branin, 1977] se propone una técnica basada en una gradiente para encontrar la aproximación de $x(0)$. Por su parte, Skelboe en [Skelboe, 1980] plantea algoritmos de extrapolación para encontrar el EEP de redes no lineales. En [Semlyen & Medina, 1995] se proponen tres métodos Newton capaces de encontrar el EEP de redes de potencia no lineales, de los cuales, este trabajo se centra en el de DN por las razones expuestas en el apartado 3.2.1 de este capítulo.

El procedimiento del método de DN se realiza de la siguiente manera:

Si se considera que la ecuación 3.1 es periódica, entonces los parámetros variantes en el tiempo serian

$$f(x,t+mT) = f(x,t) \tag{3.2}$$

La integración de la ecuación 3.2 es llevada a cabo durante m periodos de tiempo T. Al final de m periodos, el vector de estado x_n en $t = mT$ es obtenido, y después de un periodo adicional se tiene x_{m+1}, esto es llamado ciclo base [Parker & Chua, 1989]. En Parker y Chua [Parker & Chua, 1989], se sugiere para sistemas pobremente amortiguados $m = 7$ y $m = 4$ para el caso contrario.

Si se considera una perturbación sobre la trayectoria original que sigue las variables de estado, tal que x_0 es perturbada como $x_0 + \Delta x_0$, o de manera general $x(t) \to x(t) + \Delta x(t)$, entonces la ecuación 3.1 toma la forma

$$\dot{x} + \dot{\triangle} x = f(x + \Delta_x, t) \tag{3.3}$$

La linealización de (3.3) con respecto a x tomando el primer orden de términos de la expansión de la serie de Taylor resulta en

$$\underline{\dot{x}} + \Delta\dot{x} \approx f\left(\underline{x,t}\right) + D_x f(x,t)\,\Delta_x \tag{3.4}$$

Puesto que la parte subrayada de la ecuación (3.4) es satisfecha por la ecuación (3.1), entonces la ecuación (3.4) puede ser reescrita como

$$\Delta_x \approx \dot{J}(t)\,\Delta_x \tag{3.5}$$

Del cual el Jacobiano está dado por

$$J(t) = D_x f(x,t) \tag{3.6}$$

De (3.6) D_x es la matriz de derivadas parciales de $f(x,t)$ con respecto a x. Así, una matriz de coeficientes variantes en el tiempo $J(t)$ se obtiene.

La solución de la ecuación (3.5) tiene la forma:

$$\Delta_x(t) = \Phi(t)\Delta x_0 \tag{3.7}$$

En la cual

$$\Phi = e^{\int_0^t J(t)dt} \tag{3.8}$$

Donde $\Phi(t)$ es la es la matriz de identificación o matriz de transición y es de orden *nxn* del cual *n* corresponde al número de variables de estado. El cálculo de dicha matriz puede realizarse con el método Newton propuesto basado en DN. La Figura 3.2esquematiza los pasos principales que requiere el cálculo de la matriz de transición Φ usando el procedimiento de DN.

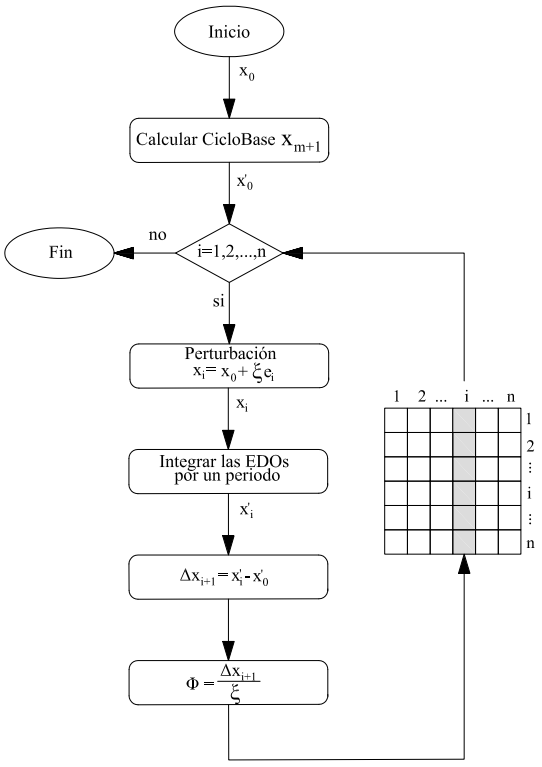


Figura 3.2: Procedimiento de Diferenciación Numérica

Este método calcula la matriz Φ por columnas, siguiendo una perturbación ξ secuencial de las variables de estado calculadas en el ciclo base x_{m+1} [Parker & Chua, 1989] donde ξ es un número pequeño, *e.g.* $1x10^{-6}$ *p.u.* [Medina *et al.*, 2006]. La perturbación de las variables de estado se expresa por

$$x_i = x_0 + \xi e_i \tag{3.9}$$

De (3.9) e_i representa la columna i-ésima de la matriz identidad I y x_0 es un vector de condiciones iniciales cuyo tamaño es igual al número de variables de estado. Posteriormente de (3.9) se obtiene

$$x_i - x_0 = \xi e_i \tag{3.10}$$

En general, para un problema de orden n, la relación estará dada en la forma

$$\Delta x_{i+1} = \Phi \Delta x_i \tag{3.11}$$

Calculando la diferencia de los 2 valores de x al final del ciclo, se obtienen las columnas de Δx_{i+1} de (3.11). Esto es, usando (3.10) en (3.11), resulta la relación

$$\Delta x_{i+1} = \Phi \xi e_i \tag{3.12}$$

Entonces para un problema de orden n con $i = 1, 2, \ldots, n$, la matriz de identificación puede ser calculada de (3.12) con la ecuación:

$$\Phi = \frac{\Delta x_{i+1}}{\xi} \tag{3.13}$$

Una vez que la matriz Φ es obtenida, una estimación de las variables de estado en el ciclo límite se consigue con el ecuación recursiva [Semlyen & Medina (1995)].

$$x_\infty = x_i + (I - \Phi)^{-1} (x_{i+1} - x_i) \tag{3.14}$$

ó

$$x_\infty = x_i + C (x_{i+1} - x_i) \tag{3.15}$$

Donde

$$C = (I - \Phi)^{-1} \tag{3.16}$$

De (3.14) x_∞ son las variables de estado al CL; x_i son las variables de estado al comienzo del ciclo base; x_{i+1} son las variables de estado al final del ciclo base; $\mathbf{\Phi}$ es la matriz de transición e I es la matriz identidad.

3.2.1.3. Estrategia Alternativa de iteración: Método de Diferenciación con Jacobiano Constante

La finalidad de este esquema de solución es reducir el número de Ciclos Completos de Integración (NCCI) obtenidos durante la aplicación del método de DN, lo cual significa una reducción del esfuerzo computacional y por lo tanto una disminución en el tiempo de computo. Sin embargo, esto se logra a expensas de degradar las propiedades de convergencia cuadrática natural del método Newton de DN [Semlyen & Medina, 1995]. La Figura 3.3 describe la aplicación de esta metodología que consiste en mantener constante la matriz C [Semlyen & Medina, 1995].

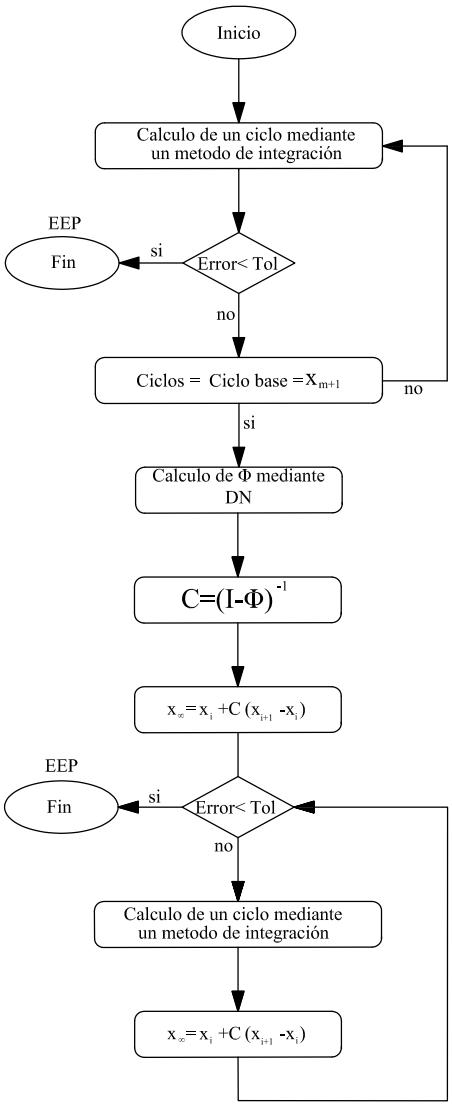


Figura 3.3: Método simple de diferenciación numérica

Con esta estrategia se obtiene una convergencia lineal, sin embargo es una estrategia efectiva cuando se tienen sistemas los cuales son representados por un conjunto de ecuaciones diferenciales muy grande, en los cuales el esfuerzo computacional para llegar al EEP utilizando la Técnica de DN puede ser más grande que incluso la misma Técnica de Fuerza Bruta, si el Jacobiano es actualizado en cada aplicación del método [Ramos, 2007].

Conforme a la Fig. 3.3 el método de solución se basa en una sola aplicación o calculo de Φ mediante el método de DN. La matriz Φ obtenida posteriormente es utilizada para conseguir una estimación inicial de las variables de estado al ciclo límite x_∞ , tal como es descrito con la ecuación (3.14). A continuación la matriz Φ es mantenida constante para posteriores evaluaciones de las variables de estado al ciclo límite. Como consecuencia de la aplicación de dicha estrategia, se requiere más aplicaciones de la ecuación (3.14) para obtener el EEP comparado con el método convencional de DN. Sin embargo, una rápida actualización de las variables al ciclo limite x_∞ es conseguido con este enfoque, ya que el proceso iterativo de identificación de la matriz Φ se omite. Este procedimiento resulta una alternativa

atractiva cuando el ciclo inicial base se localiza cercano al ciclo limite [Semlyen & Medina, 1995].

3.3. Conclusiones

En este capítulo se ha descrito de manera concisa las herramientas matemáticas y numéricas que son utilizadas para el desarrollo de este trabajo de investigación. Se describe la técnica de acercamiento rápido al estado estacionario periódico basadas en el método Newton de Diferenciación Numérica [Semlyen & Medina, 1995] que permite un análisis generalizado en el dominio del tiempo de redes eléctricas monofásicas de gran escala con componentes lineales, no lineales y variantes en el tiempo. La selección de esta técnica se hizo debido a que ha mostrado tener ventajas sobre otras técnicas en particular debido a que el proceso implica exclusivamente procesos de integración numérica además de un proceso de inversión matricial.

De igual forma se analiza una estrategia alternativa de iteración del método de Diferenciación Numérica, que tiene por objeto la disminución del número de ciclos completos de integración y por consiguiente una reducción en el esfuerzo y tiempo de computo.

Capítulo 4

Paralelización de la técnica Newton de Diferenciación Numérica utilizando la plataforma de procesamiento en paralelo CUDA

4.1. Introducción

Hoy en día, el mundo de la computación de propósito general se encuentra en un estado de continua evolución y está sufriendo cambios en su estructura interna. Cada vez son más los casos en los que supercomputadoras de gran escala utilizan tarjetas gráficas como uno de los pilares de su potencia de cálculo. Como ejemplo, podemos destacar el caso de la supercomputadora Tianhe-1, compuesto por 2560 tarjetas gráficas ATI dobles, como complemento a sus 5120 CPUs Intel Xeon, que es capaz de superar la barrera del *petaflop* (1000 trillones de operaciones en punto flotante por segundo), o de Roadrunner, que gracias a sus Power Cells, hace tan sólo unos meses se consagraba como la supercomputadora más potente del mundo [top 500].

El mayor aspecto positivo de la aparición de estas nuevas alternativas para el cómputo de carácter general, es su gran potencia de cálculo que puede ser adquirida por un bajo costo y además el consumo eléctrico originado por el uso de esta tecnología es mucho menor comparado con computadoras de características similares.

Es por ello que los recientes avances en la tecnología computacional, en específico en el ámbito del procesamiento en paralelo, presentan una oportunidad y un campo interesante para ser aplicado al análisis de sistemas de potencia. Con ello se busca afrontar la problemática anteriormente mencionada de disminuir los tiempos de simulación e incrementar la capacidad de cómputo, además resulta imperativo que el desarrollo de software para la simulación de sistemas eléctricos sea capaz de aprovechar los últimos avances en hardware de la computadora.

En este capítulo se presentan 2 propuestas de paralelización a la técnica Newton de DN utilizando la plataforma de procesamiento en paralelo multiprocesador CUDA. De igual forma se describe el método de DN con Jacobiano Constante que es mejorado con la adición de procesamiento en paralelo.

4.2. Propuestas de paralelización del cálculo de la matriz de transición de estados

La técnica Newton basada en DN requiere forzosamente el cálculo de la matriz de transición de estados $\mathbf{\Phi}$. Conforme a la subsección 3.2.1.2 del capítulo 3 de esta tesis, el proceso de identificación de dicha matriz se realiza por columnas, por lo que el esfuerzo computacional aumenta conforme el número de ecuaciones que representan el sistema es mayor. Es importante aclarar que el cálculo de cada columna es totalmente independiente del resto, por lo que resulta ideal para ser paralelizado [Medina *et al.*, 2006].

En la literatura existe una diversidad de trabajos en donde se hace uso de alguna técnica de procesamiento en paralelo para aumentar la eficiencia del método de DN, pero cabe señalar que durante la revisión del estado del arte, no se encontró trabajo asociado al uso de GPUs para este propósito.

Por consiguiente, en los siguientes apartados de este capítulo se describen 2 propuestas de paralelización al algoritmo de DN utilizando la plataforma multiprocesador CUDA. Las razones por las que se designó CUDA como plataforma de programación para la GPU son las siguientes:

- CUDA provee una sintaxis muy similar al lenguaje C y en consecuencia no resulta necesario aprender un nuevo paradigma de programación.

- Actualmente NVIDIA y sus dispositivos, se han establecido como líderes en la utilización de GPUs para el cómputo de propósito general, por ello existe una gran cantidad de información disponible sobre estos dispositivos en la pagina principal de esta empresa http://www.nvidia.com.

4.2.1. Primera Propuesta de Paralelización del cálculo de la matriz de transición de estados empleando CUDA

El primer modelo de programa paralelo del método de DN propuesto e implementado en CUDA consiste exclusivamente en el cálculo simultáneo de cada una de las columnas que comprenden a la matriz de transición $\mathbf{\Phi}$.

En la Figura 4.1 se ilustra en detalle la aplicación de esta primer propuesta paralela para el cálculo de la matriz de identificación $\mathbf{\Phi}$ utilizando GPUs.

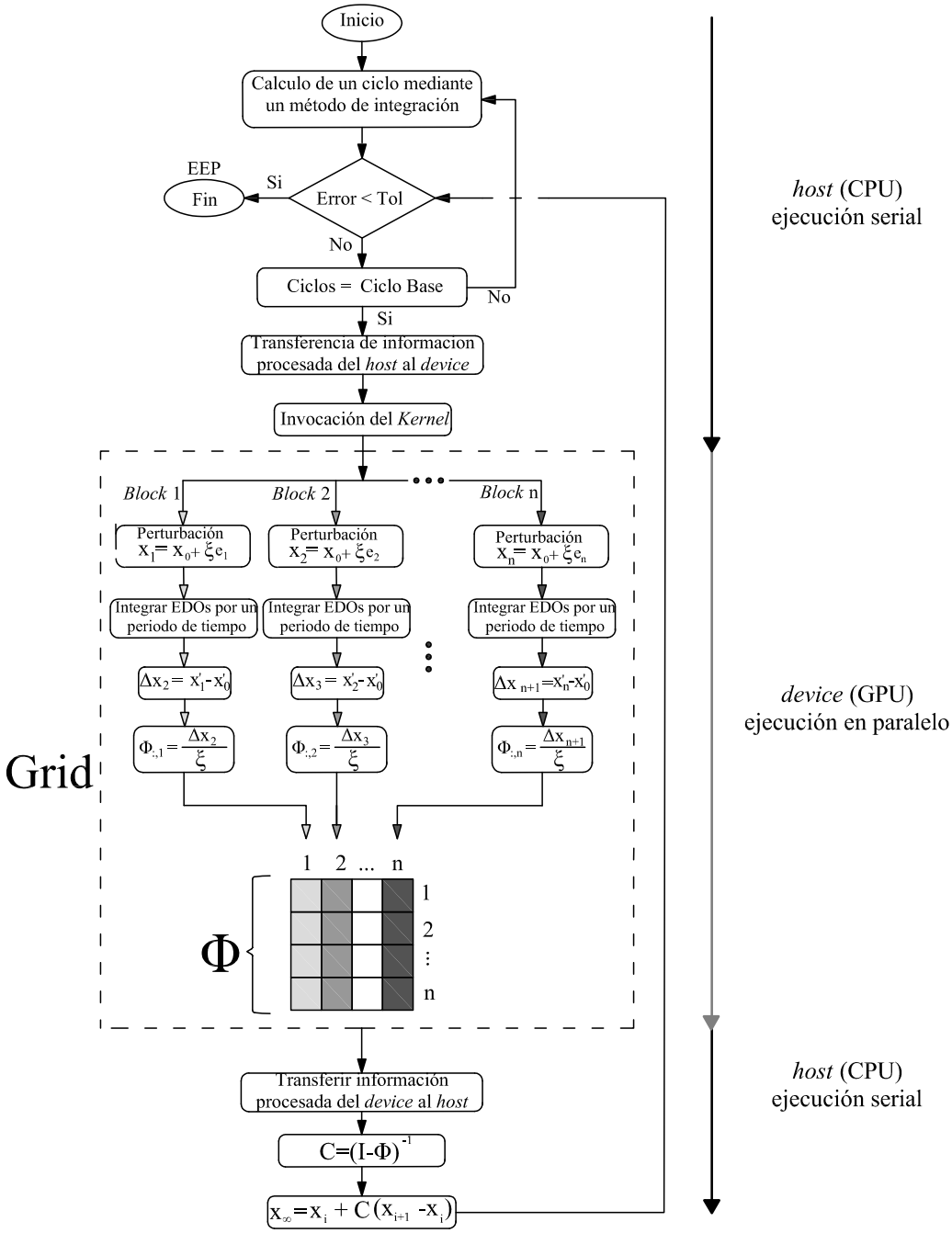


Figura 4.1: Primer propuesta de cálculo paralelo de la matriz de transición

Como se puede observar de la Figura 4.1, se asigna a cada bloque de *threads* en el GPU, el cálculo de una columna de la matriz de transición. Se puede inferir que la máxima eficiencia se lograría cuando el número de variables de estado a obtener sea igual al número de bloques de *threads* a utilizar en el GPU. Sin embargo, si se llegase a presentar que el número de variables fuese mayor al número de bloques de *threads* utilizables en el GPU, el cálculo de 2 o más columnas se realizaría en algunos bloques de *threads*. Por otro lado, si se presentase el caso contrario (el número de

variables es menor al número de bloques de *threads* del GPU), los bloques de *threads* adicionales no realizarían tarea alguna.

Conforme a la Figura 4.1, el proceso de solución general necesario para el cálculo de estado estable en el dominio del tiempo de un sistema de potencia no lineal sería el siguiente:

Primeramente el *host* se encarga de obtener el ciclo base x_{m+1} mediante un método de integración convencional, en este caso el metodo de *Runge-Kutta* de 4° orden, posteriormente el *host* compara el error con la tolerancia estipulada, de no cumplirse la tolerancia, se transfiere el vector con la información del ciclo base al *device*, el cual se encargará de obtener cada una de las columnas de la matriz de transición de estados $\mathbf{\Phi}$ de manera paralela. A continuación, esta información se regresa del device al *host*, en donde se continua con el procedimiento de DN, obteniendo con ello, la primer iteración del método, la cual servirá para comparar el error de las variables de estado respecto a la tolerancia estipulada. De no cumplirse lo anterior, se realizaría nuevamente el cálculo de la matriz $\mathbf{\Phi}$ en el device y así sucesivamente, hasta haber obtenido las variables de estado en el ciclo limite con una precisión de 10^{-10} *p.u.*

4.2.1.1. Descripción del primer Algoritmo paralelo para el cálculo de la matriz de identificación propuesto con CUDA

El Algoritmo 4.1 presentado, muestra los pasos principales de la primera implementación en la plataforma CUDA para obtener las variables de estado al CL mediante el uso del método Newton de DN.

Línea 1: Se reserva memoria en el *host* y en el *device* la memoria reservada está asociada a los vectores de condiciones iniciales de las variables de estado (x_i y x_{i+1}), la matriz de transición de estados $\mathbf{\Phi}$ y el vector de las variables al ciclo limite x_∞.

Línea 2: Se inicializan los vectores de tiempo inicial $t_0 = 0$, tiempo final $t_f = \frac{1}{60} \rightarrow T$, se igualan a 0 todos los elementos de los vectores de condiciones iniciales de las variables de estado.

Línea 3: Se compara un valor de error determinado de manera arbitraria y que ligeramente es mayor a la tolerancia estipulada, esto con la finalidad de asegurarse entrar al ciclo *while* e inicializar el método.

Línea 4: Se cálcula la solución del sistema de EDOs durante un periodo mediante un método de integración, en este caso el método de RK4.

Línea 5: Se verifica la condición si el número de ciclos iniciales es mayor o igual al ciclo base, de no ser cierta dicha condición, el algoritmo se regresa al paso número 3 y se calculara de nuevo la solución de las EDOs, en caso contrario, el algoritmo continua con el siguiente paso

Linea 6: Se transfiere la información procesada en el *host* que corresponde al vector con el ciclo base al *device*.

Línea 7: Es la parte medular del programa, ya que en este paso se invoca el *kernel* que tendrá la función de encontrar de manera paralela, cada una de las columnas de la matriz de transición de estados. Con la finalidad de obtener la máxima eficiencia, el número de bloques de *threads* creados corresponde al número de variables de estado y el número de *threads* por bloque únicamente es 1.

Línea 8: Se regresa la información procesada del *device* al *host,* en este caso se traslada la matriz $\mathbf{\Phi}$.

Líneas 9 y 10: Se obtiene la matriz *C*

Línea 11: Se cálculan las variables de estado al Ciclo Limite x_∞ mediante la ecuación (3.14)

Linea 12: Nuevamente se calcula el error máximo obtenido de las variables de estado con la tolerancia asignada, si el error aun no es menor a la tolerancia estipulada, entonces el algoritmo retorna al paso número 4 para posteriormente calcular de nuevo la matriz $\mathbf{\Phi}$ de manera paralela en el *device*. Este proceso se repetirá hasta que converja a la solución.

En resumen, este algoritmo consta de 3 partes que son ejecutadas por el *host* y el *device*. La primera parte que corresponde a los pasos 1-5 del Algoritmo 4.1 es ejecutada por el *host* de manera secuencial y conciernen al cálculo del ciclo base x_{m+1}. La segunda parte del algoritmo compete al cálculo de la matriz de transición de estados; estos pasos son ejecutados en el *device* de manera paralela y corresponde a los pasos 6-8. Finalmente los pasos 9-12 son ejecutados en el *host* nuevamente, dichos pasos se adjudican al cálculo de las variables al CL $\rightarrow x_\infty$.

Algoritmo 4.1 Primera implementación del cálculo paralelo de la matriz de transición de estados

1. Se reserva memoria en el *host* y *device*
2. Inicializa los vectores de condiciones iniciales $\left(t_0, t_f, x_i, x_{i+1}\right)$
3. ***while*** (error > tolerancia)
   {
4. Calcula solución del sistema de EDOs por un periodo mediante método de integración
5. ***if*** (ciclos iniciales >= ciclo base)
   {
6. Se transfieren los vectores (x_{i+1}, x_i) del *host* al *device*
7. Invocación de Kernel para el cálculo de $\mathbf{\Phi}$ en el device
   }
8. Se transfiere la matriz $\mathbf{\Phi}$ calculada del *device* al *host*
9. Calculo de la diferencia $(\mathbf{I} - \mathbf{\Phi})$
10. Obtención de inversa $(\mathbf{I} - \mathbf{\Phi})^{-1} = \mathbf{C}$
11. Calculo de variables al ciclo limite $x_\infty = x_i + \mathbf{C}(x_{i+1} - x_i)$
12. Comparación del error con la tolerancia estipulada y se detiene proceso si converge a la solución, de lo contrario regresa a paso 4
   }

4.2.2. Segunda Propuesta de Paralelización del cálculo de la matriz de transición de estados empleando CUDA

Esta segunda alternativa de solución es muy similar a la primera propuesta de paralelización de la matriz de identificación de estados, ya que de igual forma se cálcula cada una de las columnas que componen la matriz $\mathbf{\Phi}$ de manera simultánea. La diferencia de esta segunda propuesta radica, en que implícitamente el método de integración utilizado para evaluar cada una de las columnas de manera paralela, se encuentra también paralelizado mediante la plataforma CUDA[Magaña *et al.*, 2013] . El método de integración empleado en este trabajo es el Runge Kutta de 4° orden (RK4). Este método se representa mediante la expresión:

$$x_{i+1} = x_i + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \cdot h \tag{4.1}$$

donde:

$$\begin{aligned} k_1 &= f\left(t, x_i\right) \\ k_2 &= f\left(t + \tfrac{1}{2}h, x_i + \tfrac{1}{2}k_1 h\right) \\ k_3 &= f\left(t + \tfrac{1}{2}h, x_i + \tfrac{1}{2}k_2 h\right) \\ k_4 &= f\left(t + h, x_i + k_3 h\right) \end{aligned} \tag{4.2}$$

Los términos $x_i + \frac{1}{2}k_1h$, $x_i + \frac{1}{2}k_2h$, $x_i + k_3h$ y $\frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \cdot h$ que son usados para obtener los terminos k_2, k_3, k_4 y x_{i+1} respectivamente, necesitan ciclos para ser obtenidos.

El algoritmo 4.2 muestra un fragmento del código implementado en lenguaje $c/c++$ para el cálculo de cada una de las pendientes y el promedio de ellas en el método de RK4 de manera convencional.

Algoritmo 4.2 Cálculo convencional de las pendientes del método de Runge Kutta de 4° orden

```
void runge(double t, double *y0, double *yn, int var_est)
    {
      for (int io = 0; io <= hh; io++)
        {
            cir_ecuaciones(t, yn, k1);
                for (int i = 0; i < var_est; i++)
            aux[i] = yn[i] + h*k1[i]/2.0;
            cir_ecuaciones(t + h/2.0, aux, k2);
                for (int i = 0; i < var_est; i++)
            aux[i] = yn[i] + h*k2[i]/2.0;
            cir_ecuaciones(t + h/2.0, aux, k3);
                for (int i = 0; i < var_est; i++)
            aux[i] = yn[i] + h*k3[i];
            cir_ecuaciones (t + h, aux, k4);
                for (int i = 0; i < var_est; i++)
            yn[i] = yn[i] + h*(k1[i] + 2.0*k2[i] + 2.0*k3[i] + k4[i])/6.0;
            t += h;
        }
    }
```

Como se aprecia en el algoritmo 4.2 , se necesitan de 4 ciclos for para obtener los parametros k_2, k_3, k_4 y x_{i+1}, y cada uno de estos ciclos dependen del numero de variables de estado que modelan la dinamica del sistema y el paso de integración estipulado. En consecuencia, conforme se incrementa la dimension y complejidad de la red, el número de variables de estado que representan su dinámica tambien lo hace, convirtiendo a este proceso computacionalmente demandante.

Estos ciclos pueden ser omitidos si se aprovechan los *threads* de cada bloque.

Puede observarse en la ecuación 4.2 que para obtener el termino k_2 se necesitan dos parámetros. El primer parámetro es un escalar mientras que el segundo es un vector. En la expresión de k_2, el término $t + \frac{1}{2}h$ es el escalar mientras que el término $x_i + \frac{1}{2}k_1$ es el vector. De acuerdo al algoritmo 4.2 si creamos una variable auxiliar que almacene los valores del vector, es decir, $aux = x_i + \frac{1}{2}k_1h$, entonces la expresión para el calculo de $k_2$seria:

$$k_2 = f\left(t + \frac{1}{2}h, aux\right) \tag{4.3}$$

La Figura 4.2 muestra el proceso convencional para obtener el vector *aux* necesario para el cálculo del término k_2 del método RK4. Este proceso comienza determinando la posición cero del vector y continua hasta el cálculo de la posición *n* del vector. Donde *n* es igual al número de variables de estado menos uno.

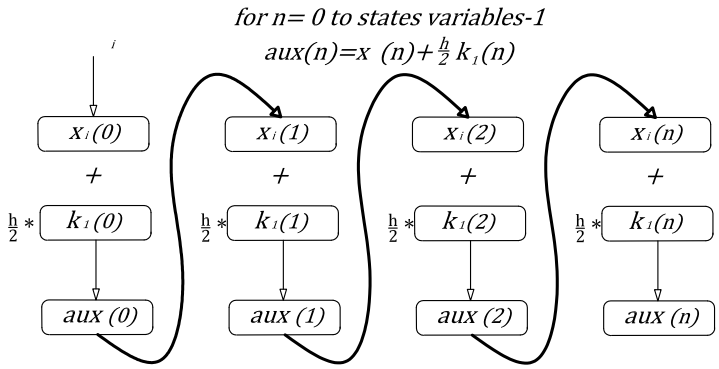


Figura 4.2: Ciclo para obtener k_2 en el metodo RK4 convencionalmente

Ahora en la Figura 4.3 se muestra el mismo proceso para obtener la variable *aux* para el calculo de k_2, pero esta vez con el apoyo y uso de los *threads* de cada bloque creado para calcular de manera paralela las columnas de la matriz de transicion de estados **Φ [Magaña *et al.*, 2013].**

Puede observarse que cada *thread* ejecuta una simple operación de la expresión $aux(n) = x_i(n) + \frac{h}{2}k_1(n)$, como se menciona n corresponde al numero de variables de estado que representan la dinámica del sistema, entonces el numero de *threads* que deben ser creados para obtener una máxima eficiencia de igual manera correspondería a n.

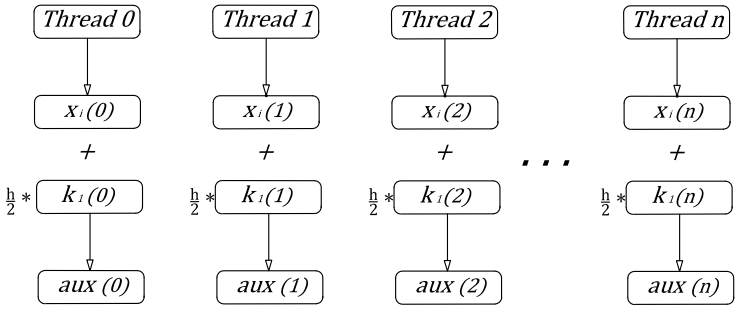


Figura 4.3: Ciclo para obtener k_2 de manera paralela con el uso de *threads* [*Magaña* et al., *2013*]

El procedimiento necesario para obtener los términos k_3, k_4 y x_{i+1}, con el uso de *threads,* es similar al empleado para obtener el vector *aux* de k_2. El algoritmo 4.3 muestra una pequeña sección del codigo $c/c++$ implementado para obtener de manera paralela mediante CUDA los vectores correspondientes a los terminos k_2, k_3, k_4 y x_{i+1} del método de RK4.

Algoritmo 4.3 Cálculo paralelo de las pendientes del método de Runge Kutta de 4° orden

```
__global__ void paralelo(double*B, double*dy0, double*yn, int var_est, double t)
  {
    for (int io = 1; io <= hh; io++)
      {
        cir_ecuaciones(yb, k1, var_est, w, t, hil);
        __syncthreads( );
        aux[hil] = yb[hil] + h*k1[hil]/2.0;
        __syncthreads( );
        cir_ecuaciones(aux, k2, var_est, w, t + h/2.0, hil);
        __syncthreads( );
        aux[hil] = yb[hil] + h*k2[hil]/2.0;
        __syncthreads( );
        cir_ecuaciones(aux, k3, var_est, w, t + h/2.0, hil);
        __syncthreads( );
        aux[hil] = yb[hil] + h*k3[hil]/2.0;
        __syncthreads( );
        cir_ecuaciones(aux, k4, var_est, w, t + h, hil);
        __syncthreads( );
        yb[hil] = yb[hil] + h*(k1[hil] + 2.0*k2[hil] + 2.0*k3[hil] + k4[hil])/6.0;
        __syncthreads( );
        t+ = h;
      }
  }
```

La Figura 4.4 representa la forma en como se obtiene de manera paralela mediante la segunda alternaiva de calculo la matriz de transición de estados $\mathbf{\Phi}$.

Conforme a la Figura 4.4 se observa que cada bloque del *Grid* obtiene una de las columnas correspondientes a la matriz de transición de estados. Posteriormente los *threads* de cada bloque evalúan de manera paralela cada una de las pendientes y promedio ponderado del método de integración, en este caso las del método de Runge Kutta de 4° orden Magaña *et al.*, 2013. Cabe aclarar que esto lo realiza con un paso de integración estipulado $\triangle t$ y durante un lapso de tiempo igual al periodo T. De acuerdo a lo mencionado y con la finalidad de obtener la máxima eficiencia, el número de bloques de *threads* creados corresponde al número de variables de estado y el número de *threads* por bloque de igual forma corresponde al número de variables de estado.

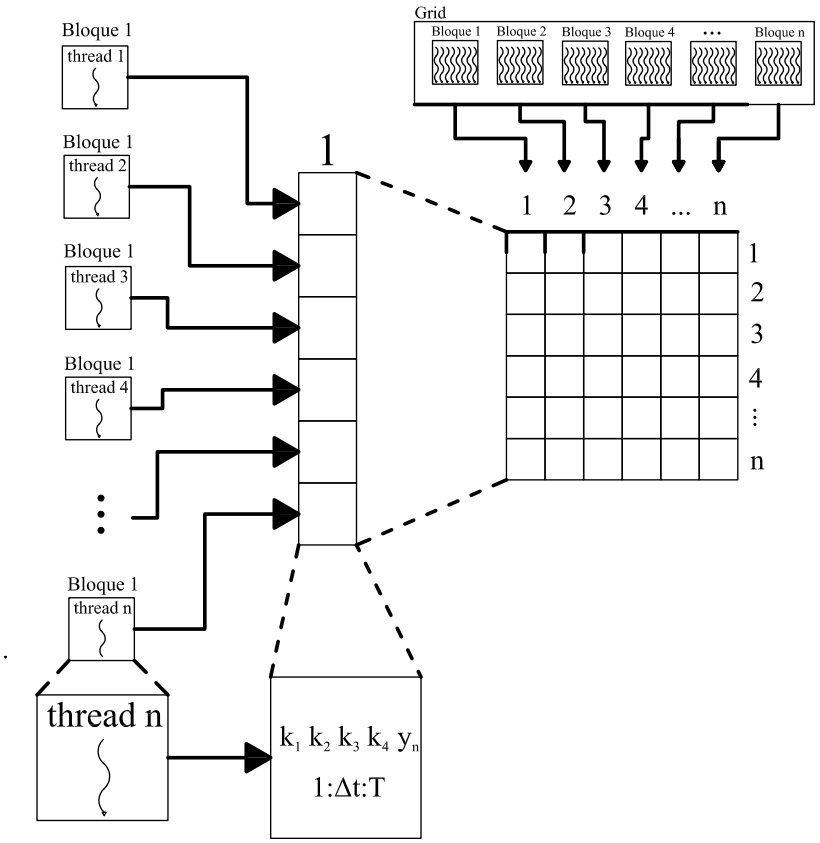


Figura 4.4: Segunda propuesta paralela de calculo de la matriz de transición [Magaña *et al.*, 2013]

4.2.2.1. Descripción del segundo Algoritmo paralelo para el cálculo de la matriz de identificación propuesto con CUDA

El Algoritmo 4.4, muestra los pasos principales de la segunda propuesta e implementación en la plataforma CUDA del método Newton de DN. El algoritmo es muy similar al algoritmo 4.1 y de igual forma consta de tres secciones.

Lineas 1-6: Esta primera sección del codigo es realizada en el *host* (CPU) y se obtiene el ciclo base, al igual, se transfiere la información de este vector del *host* al *device*.

Líneas 7-9: La segunda sección del programa es realizada por el *device* (GPU)*;* en esta sección del programa se presenta la diferencia entre las propuestas empleadas para el cálculo de $\mathbf{\Phi}$, ya que en esta segunda alternativa se invoca el *kernel* que tendrá la función de encontrar de manera paralela, cada una de las columnas de la matriz de transición de estados con el apoyo de los bloques de *threads* e implícitamente con el uso de *threads* se obtienen los terminos k_2, k_3, k_4 y x_{i+1} del metodo de integración RK4.

Líneas 10-14: La tercer sección del codigo es realizada por el *host* y corresponden al calculo de la matriz C, la obtención de las variables de estado al CL x_∞ y la comparativa del error máximo obtenido de las variables de estado con la tolerancia estipulada.

Algoritmo 4.4 Segunda implementación del cálculo paralelo de la matriz de transición de estados

1. Se reserva memoria en el *host* y *device*
2. Inicializa los vectores de condiciones iniciales (t_0, t_f, x_i, x_{i+1})
3. ***while*** (error > tolerancia)
   {
4. Calcula solución del sistema de EDOs por un periodo mediante método de integración
5. ***if*** (ciclos iniciales >= ciclo base)
      {
6. Se transfieren los vectores (x_{i+1}, x_i) del *host* al *device*
7. Invocación de Kernel para el cálculo de $\mathbf{\Phi}$en el device
8. Se calculan las columnas de la matriz $\mathbf{\Phi}$ mediante bloques de *threads*
9. Se obtienen los vectores auxiliares (aux) necesarios para obtener los terminos k_2, k_3, k_4 y x_{i+1} del metodo RK4 mediante *threads*
      }
10. Se transfiere la matriz $\mathbf{\Phi}$ calculada del *device* al *host*
11. Calculo de la diferencia $(\mathbf{I} - \mathbf{\Phi})$
12. Obtención de inversa $(\mathbf{I} - \mathbf{\Phi})^{-1} = C$
13. Calculo de variables al ciclo limite $x_\infty = x_i + \mathbf{C}(x_{i+1} - x_i)$
14. Comparación del error con la tolerancia estipulada y se detiene proceso si converge a la solución, de lo contrario regresa a paso 4
   }

4.3. Método paralelo de Diferenciación Numérica con Jacobiano Constante

Este procedimiento es similar al método de DN con Jacobiano Constante descrito en la Sección 3.2.1.3 del Capítulo 3 de esta tesis, la diferencia radica en que el cómputo de la matriz de transición de estados se realiza de manera paralela en la GPU mediante las propuestas antes mencionadas en los apartados anteriores de este capitulo.

4.4. Conclusiones

En este capítulo se describió la implementación de procesamiento paralelo basado en GPUs a las técnicas Newton de DN y DN con Jacobiano Constante, las cuales permiten hallar la solución en EEP en el dominio del tiempo de redes eléctricas con componentes no lineales y variantes en el tiempo. La adición de procesamiento paralelo a estos procedimientos se realizó con la finalidad de optimizar los tiempos de cómputo obtenidos de manera secuencial con estas metodologías.

Capítulo 5

Casos de estudio y su solución en el dominio del tiempo

5.1. Introducción

Con el propósito de demostrar la eficiencia de las metodologías propuestas, se han evaluado 5 casos de estudio monofásicos entre los que se encuentran los sistemas de prueba del IEEE de 14, 30, 57 y 118 nodos. Cabe aclarar que los sistemas de prueba han sido modificados mediante la adición de componentes no lineales como ramas magnetizantes y hornos de arco eléctrico, esto con la finalidad de aumentar el esfuerzo computacional y crear distorsión armónica en la red. En consecuencia a dichas modificaciones los sistemas evaluados poseen pobre amortiguamiento y por lo tanto el número de evaluaciones iniciales para obtener el ciclo base y comenzar los métodos Newton de DN y DN con Jacobiano Constante, será conforme a lo mencionado en [Semlyen & Medina, 1995].

También con el propósito de obtener la máxima eficiencia, el número de bloques de *threads* o núcleos a utilizar de la GPU corresponde al número de variables de estado que representan la dinámica de cada sistema. Se han elegido las corrientes en líneas de transmisión, flujos en las ramas no lineales, voltajes nodales y en el caso donde se cuente con un horno de arco eléctrico al radio del mismo como variables de estado.

Los elementos que conforman los casos de estudio se han modelado como se reporta en [Ramos, 2007], mismos que se describen en el Apéndice A, y las ecuaciones que describen la dinámica de los casos de estudio fueron obtenidas con la herramienta de simulación desarrollada por [Ramos, 2007]. De acuerdo a la metodología propuesta en [Ramos, 2007], se han añadido bancos de capacitores en derivación en todos los nodos de los sistemas de potencia a ser evaluados, esto con el propósito de contar con una variable de estado que represente el voltaje en los mismos.

Por otra parte para los diversos casos de estudio se ha especificado un criterio de convergencia para localizar el Ciclo Limite de 1×10^{-10} en $p.u.$ y se consideran fuentes de voltaje sinusoidales con amplitud de 1.0 en $p.u.$

Con la finalidad de facilitar la identificación de las metodologías empleadas para el análisis de los distintos casos de estudio de este capítulo, se propone la siguiente clasificación:

- Fuerza Bruta=FB
- Diferenciación Numérica=DN
- Diferenciación Numérica y empleando la primer propuesta de paralelización= DN-Φ

- Diferenciación Numérica y empleando la segunda propuesta de paralelización= DN-$\mathbf{\Phi_{RK}}$
- Diferenciación Numérica con Jacobiano Constante= DNJC
- Diferenciación Numérica con Jacobiano Constante y empleando la primer propuesta de paralelización= DNJC-$\mathbf{\Phi}$
- Diferenciación Numérica con Jacobiano Constante y empleando la segunda propuesta de paralelización= DNJC-$\mathbf{\Phi_{RK}}$

El equipo de computo empleado para este trabajo de tesis se compone de una CPU modelo E5606, con procesador Intel Xeon, con capacidad de memoria de 6 GB y velocidad de 2.2 GHz (Figura 5.1a), mientras que la tarjeta gráfica utilizada es una NVIDIA modelo Tesla C2075, con 448 núcleos, capacidad de memoria de 6 GB y velocidad por núcleo de 1.15 GHz (Figura 5.1b). La comunicación entre el CPU y el GPU se realiza mediante un cable PCIe x 16 de segunda generación, que admite un máximo de velocidad de transferencia de 5 GB/s. El sistema operativo usado es la versión 17 de fedora (64 bits). El driver de CUDA utilizado es el correspondiente a la versión 4.1 del toolkit.

Figura 5.1: a) CPU modelo E5606 b) NVIDIA GPU modelo Tesla C2075

Una vez obtenido el sistema de EDO's, se emplean los métodos propuestos para determinar el EEP del sistema eléctrico analizado.

En todos los casos de estudio se muestra:

- La comparación entre el Número de Ciclos Completos de Integración (NCCI) necesarios que ocupa cada una de las metodologías empleadas así como el tiempo de ejecución de estas.
- El contenido armónico y la distorsión armónica total (%THD, por sus siglas en inglés) de la forma de onda obtenida del sistema en EEP.
- La Eficiencia Relativa obtenida con la incorporación de procesamiento en paralelo usando la plataforma CUDA.

5.2. Sistema de 3 nodos

La Figura 5.2 está compuesta por un generador, tres líneas de transmisión, dos ramas magnetizantes y tres bancos de capacitores. La dinámica del sistema es representada por un conjunto de 9 EDO's.

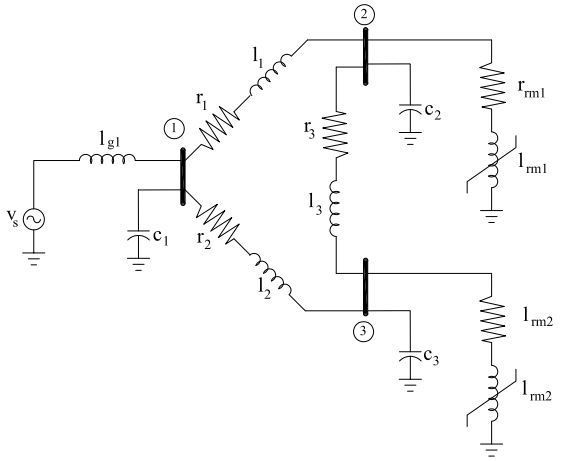


Figura 5.2: Caso de estudio 1: Sistema eléctrico de 3 nodos monofásico [Semlyen & Medina, 1995].

Las Tablas 5.2 y 5.2 proporcionan el error máximo obtenido de las variables de estado del sistema de 3 nodos al ser evaluado mediante las distintas metodologías de solución en el dominio del tiempo. Dichos resultados se encuentran en términos del NCCI requeridos para alcanzar el EEP. De acuerdo a los datos de la Tablas 5.2 y 5.2, el método de DN ya sea en su forma secuencial o paralela, necesita un total de 27 Ciclos de integración para obtener el EEP, que son el resultado de 2 iteraciones del método de DN más 8 ciclos iniciales necesarios para la obtención del ciclo base. Mientras que los métodos de FB y DNJC (secuencial y paralelo), requieren un total de 52 y 19 ciclos de integración respectivamente. Esto significa que los métodos de DN y DNJC requieren un esfuerzo computacional menor equivalente al 53.84 y 38.46 por ciento respecto del método de FB para obtener el EEP.

Tabla 5.1: Sistema de 3 nodos, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas en una computadora convencional

(NCCI)	$\triangle t$=16.27 μs		
	FB	**DN**	**DNJC**
8	7.5389e-03	7.5389e-03	7.5389e-03
17	5.7938e-05	3.4219e-05	3.4219e-05
19	1.9041e-05	⋮	1.6853e-13
27	1.3627e-06	6-4060e-12	
⋮			
52	9.7805e-11		
Tiempo (ms)	**142**	**78**	**58**

Tabla 5.2: Sistema de 3 nodos, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas mediante CPU-GPU.

(NCCI)	$\triangle t$=16.27 μs			
	DN-Φ	**DNJC-Φ**	**DN-Φ_{RK}**	**DNJC-Φ_{RK}**
8	7.5389e-03	7.5389e-03	7.5389e-03	7.5389e-03
17	3.4219e-05	3.4219e-05	3.4219e-05	3.4219e-05
19	⋮	1.6853e-13	⋮	1.6853e-13
27	6-4060e-12		6-4060e-12	
Tiempo (ms)	**155**	**97**	**167**	**105**

En las Figuras 5.3 y 5.4 se muestran las formas de onda representativas de las variables de estado asociadas con el caso de estudio analizado. La Figura 5.3 está relacionada al flujo que circula en la línea de transmisión que va del nodo 1 al nodo 2, mientras que la Figura 5.4 presenta el comportamiento del voltaje en el nodo 2.

La distorsión en la formas de onda de la corriente se debe a la presencia de las ramas magnetizantes en el sistema. En la Figura 5.3 se observa un elevado contenido armónico que provoca un THD aproximado a 47 %; el cual es un nivel muy por encima de la norma IEEE 519 vigente.

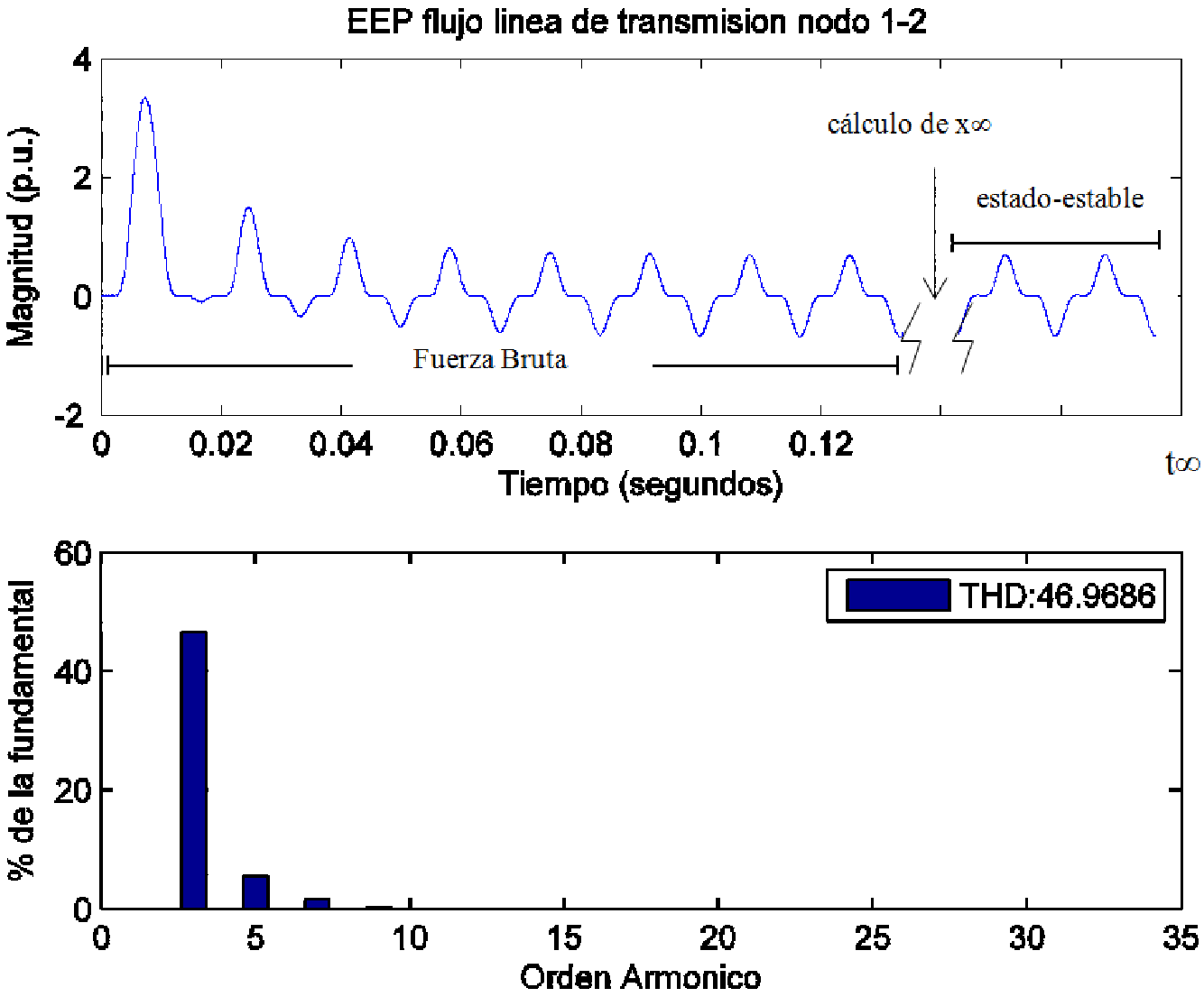
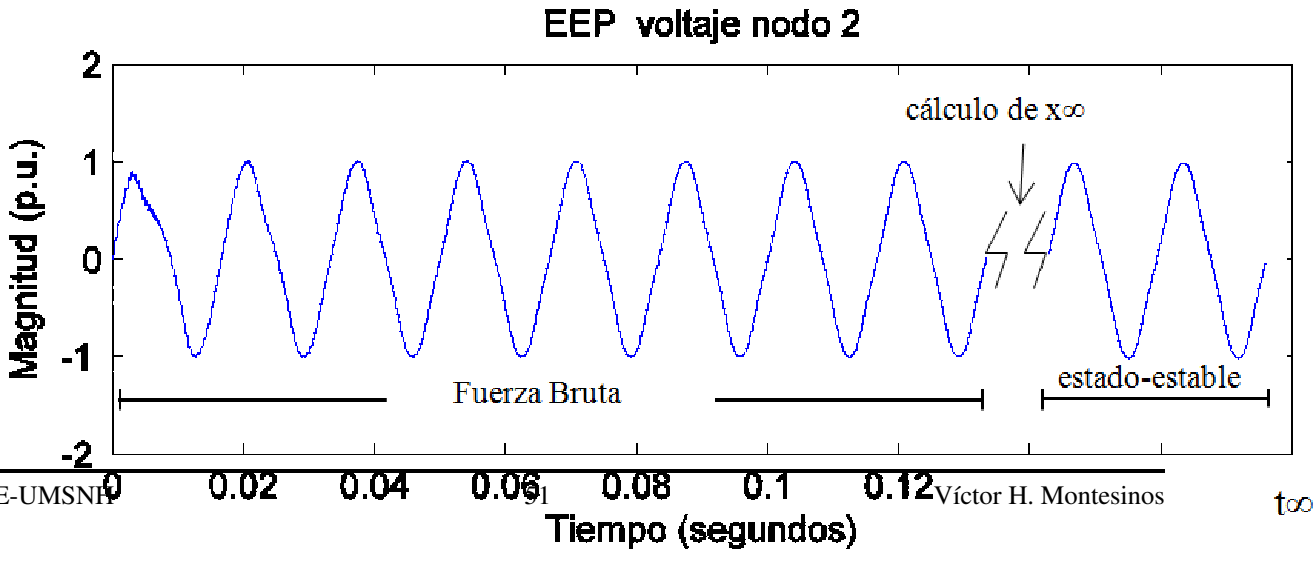


Figura 5.3: Forma de onda de la corriente en la LT de 1-2.



5.3. Sistema de IEEE de 14 nodos modificado

La Figura 5.5 muestra al sistema de potencia de 14 nodos del IEEE [ee.washington.edu]. Este sistema se encuentra compuesto por 20 líneas de transmisión, 2 generadores y 14 bancos de capacitores. El sistema ha sido modificado con la incorporación de dos ramas magnetizantes conectadas a los nodos 2 y 10. La dinámica de la red de potencia es representada por un conjunto de 38 EDO's.

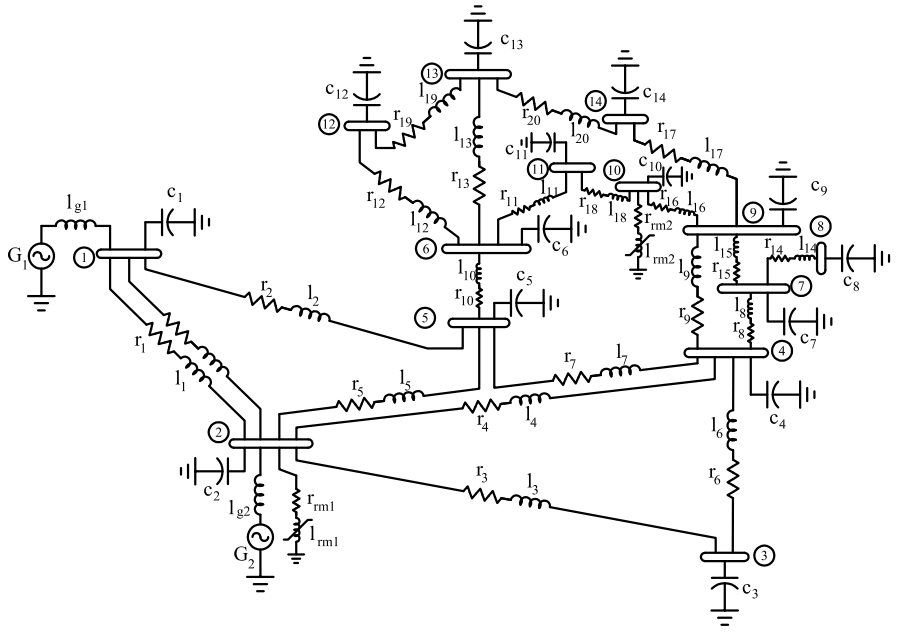


Figura 5.5: Sistema de 14 nodos modificado del IEEE

Las Tablas 5.3 y 5.3 muestran el error máximo obtenido de las variables de estado durante cada ciclo completo de integración. De acuerdo a la teoría del capítulo 3 de esta tesis y conforme a las características del sistema de la Figura 5.5, ha sido propuesto el cálculo de 7 ciclos iniciales mas uno para obtener el ciclo base y poder comenzar con los métodos de DN y DNJC. Cabe señalar de las Tablas 5.3 y 5.3 que los términos de la primera columna resaltados con negrita representan el NCCI que el método de DN necesita para obtener el EEP. En este caso 117 ciclos (3 iteraciones), ya sea ejecutado en forma secuencial o paralela, mientras que los métodos de FB y DNJC requieren un total de 4366 y 49 ciclos de integración respectivamente. Esto significa que los métodos de DN y DNJC requieren un esfuerzo computacional menor equivalente al 2.84 y 1.12 por ciento respecto del método de FB para obtener el EEP y representa un 39.51 por ciento de iteraciones del método de DNJC respecto del método de DN.

Tabla 5.3: Sistema de 14 nodos modificado del IEEE, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas en una computadora convencional

(NCCI)	$\triangle t$=32.55 μs		
	FB	**DN**	**DNJC**
8	22.053e-03	22.053e-03	22.053e-03
46	1.0364e-06	1.4464e-04	1.4464e-04
49	1.0297e-06	⋮	1.0141e-12
85	9.5339e-07	4.1529e-10	
124	8.7701e-07	9.6700e-14	
⋮	⋮		
4366	9.9623e-11		
Tiempo (ms)	**14023**	**401**	**166**

Tabla 5.4: Sistema de 14 nodos, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas mediante CPU-GPU.

(NCCI)	$\triangle t$=32.55 μs			
	DN-Φ	**DNJC-Φ**	**DN-Φ_{RK}**	**DNJC-Φ_{RK}**
8	22.053e-03	22.053e-03	22.053e-03	22.053e-03
46	1.4464e-04	1.4464e-04	1.4464e-04	1.4464e-04
49	⋮	1.0154e-12	⋮	1.0154e-12
85	4.1659e-10		4.1659e-10	
124	5.3624e-14		5.3624e-14	
Tiempo(ms)	**397**	**155**	**222**	**96**

En las Figuras se muestran las formas de onda representativas de las variables de estado asociadas con el caso de estudio analizado, la Figura 5.6 está asociada al voltaje del nodo 10 y la Figura 5.7 presenta el comportamiento del flujo en la rama magnetizante conectada al nodo 10.

La distorsión en las formas de onda se debe a la presencia de las ramas magnetizantes del transformador, cuyo punto de operación se encuentra en la región de saturación. En la Figura 5.6 se observa un elevado contenido armónico; así como un alto THD; ambos por encima de la norma IEEE 519 vigente.

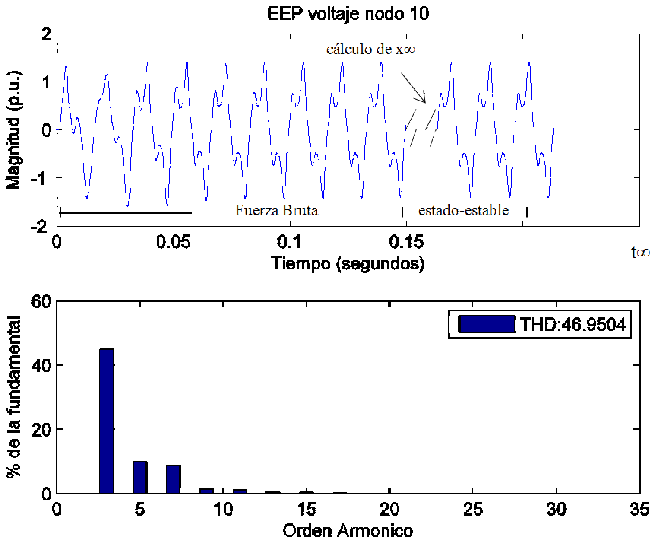


Figura 5.6: Comportamiento del Voltaje en el nodo 10

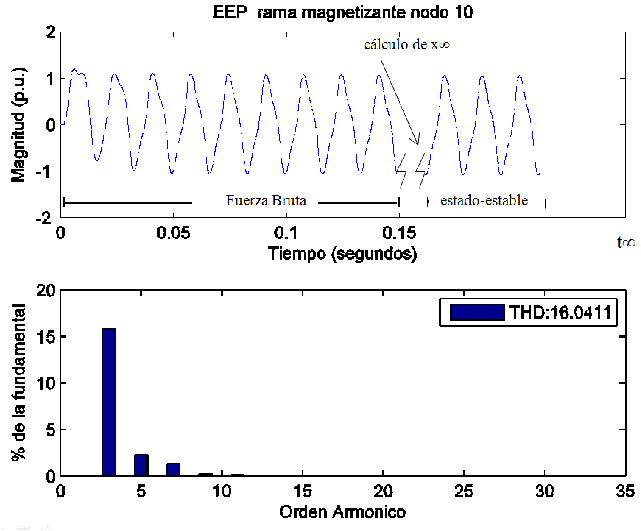


Figura 5.7: Flujo en la rama magnetizante nodo 10

5.4. Sistema de IEEE de 30 nodos modificado

El sistema eléctrico de análisis para este caso de estudio cuenta con 30 nodos y está estructurado por 41 líneas de transmisión, 30 bancos de capacitores y 6 unidades generadoras [ee.washington.edu]. Éste sistema ha sido modificado mediante la incorporación de un horno de arco conectado al nodo 22 y 3 ramas magnetizantes añadidas en los nodos 3, 7 y 18. El comportamiento del sistema eléctrico se puede representar por medio de 82 EDO's.

Tabla 5.5: Sistema de 30 nodos modificado del IEEE, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas en un computadora convencional

(NCCI)	$\triangle t$=32.55 μs		
	FB	**DN**	**DNJC**
8	17.36e-02	17.36e-02	17.36e-02
90	4.9751e-05	0.13647	0.13647
100	2.7831e-05	⋮	7.70583e-11
173	4.0097e-07	7.18951e-05	
256	4.0514e-08	2.12272e-09	
339	3.7599e-08	4.8739e-14	
⋮	⋮		
27853	9.99971e-11		
Tiempo(ms)	**187057**	**2352**	**694**

Tabla 5.6: Sistema de 30 nodos, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas mediante CPU-GPU.

(NCCI)	$\triangle t$=32.55 μs			
	DN-Φ	**DNJC-Φ**	**DN-Φ_{RK}**	**DNJC-Φ_{RK}**
8	17.36e-02	17.36e-02	17.36e-02	17.36e-02
90	0.13647	0.13647	0.13647	0.13647
100	⋮	7.70512e-11	⋮	7.70512e-11
173	7.18980e-05		7.18980e-05	
256	2.06716e-09		2.06716e-09	
339	4.5741e-14		4.5741e-14	
Tiempo(ms)	**1241**	**397**	**625**	**244**

De las Tablas 5.4 y 5.4 se observa que para este caso de estudio, el método de FB requiere de un total de 27853 ciclos para alcanzar el EEP. En contraparte el método de DN requiere de un total de 339 ciclos de integración para hallar la solución periódica, de los cuales 7 son asociados a la obtención del ciclo base más 332 ciclos equivalentes a 4 iteraciones del método de DN. Finalmente el método de DNJC requirio de 100 ciclos de integración para lograr la solución en estado estable, que al igual que el método de DN, 7 ciclos son asociados a la obtención del ciclo base, 83 ciclos proporcionales a 1 iteración del método de DN y por último 10 ciclos de integración manteniendo el Jacobiano constante para el cálculo del Ciclo Limite.

Analizando los datos de la Tabla 5.4 el método de DN, ya se ejecutado en su forma secuencial o en sus versiones paralelas, demandó un menor esfuerzo computacional equivalente al 1.2171 % del NCCI obtenidos por el método de FB para alcanzar el EEP. Por su parte, el método de DNJC en comparación con los métodos de FB y DN necesitó de un equivalente al 0.3590 %. y 29.4985 % respectivamente, por lo cual se ve reflejada la eficiencia de este método al reducir el NCCI y como consecuencia el tiempo.

Otros aspectos que puede denotarse de la Tabla 5.3 es el patrón de convergencia obtenido con el método de DN, "cuadrático", el cual es un comportamiento esperado para una técnica tipo Newton, mientras que el comportamiento del método de DNJC demuestra una característica súper-lineal al mantener constante el Jacobiano.

Las Figuras 5.8 y 5.9, muestran la evolución en el tiempo de algunas variables de estado representativas en EEP del sistema de 30 nodos modificado. En la Figura 5.8 se muestra el voltaje en el nodo 22, el cual tiene un bajo contenido

armónico reflejado en un THD de 1.2614 %, parámetro permisible por la norma IEEE-519. Por otra parte, la Figura 5.9 presenta la forma de onda de la corriente que circula en la línea de transmisión que une a los nodos 10 y 22 del sistema. El contenido armónico de ésta corriente está asociado principalmente con el tercer armónico con magnitud superior al 4 % y con un %THD = 4.3077 %.

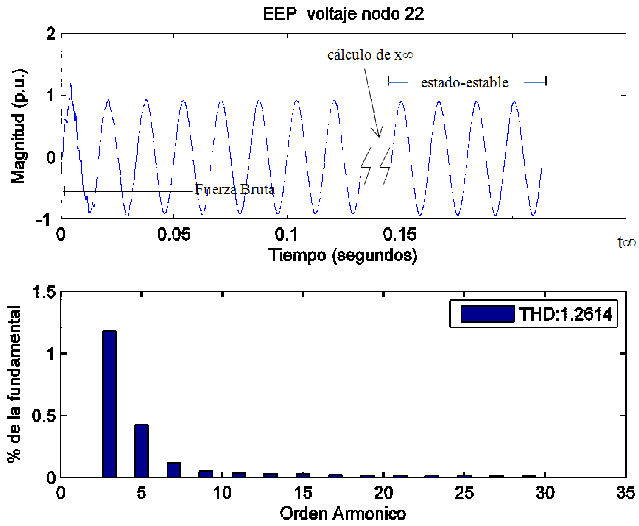


Figura 5.8: Voltaje en el nodo 22

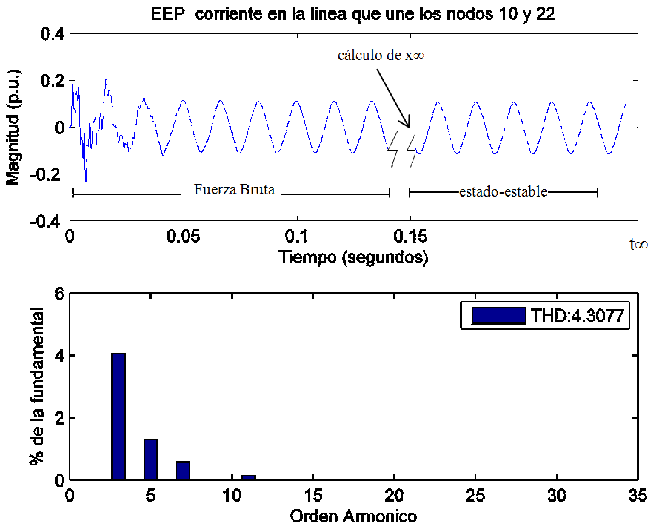


Figura 5.9: Corriente en la línea que une los nodos 10 y 22

5.5. Sistema de IEEE de 57 nodos modificado

Esta red de prueba tiene la topología del sistema IEEE-57 nodos [ee.washington.edu], que consta de 78 líneas de transmisión, 57 bancos de capacitores y 7 generadores. Este sistema fue modificado mediante la adición de 6 ramas magnetizantes conectadas a los nodos 9, 15, 18, 23, 38 y 45 y un horno de arco eléctrico instalado en el nodo 15. Su representación en espacio de estado se modela por medio de 150 EDO's.

Tabla 5.7: Sistema de 57 nodos modificado del IEEE, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas en una computadora convencional

(NCCI)	$\triangle t$=32.55 μs		
	FB	**DN**	**DNJC**
8	24.950e-02	24.950e-02	24.950e-02
158	22.194e-09	31.569e-03	31.569e-03
169	68.693e-10	⋮	19.589e-12
209	96.491e-12		
309		28.535e-05	
460		23.243e-09	
611		11.668e-14	
Tiempo (ms)	**2302**	**6966**	**1918**

Tabla 5.8: Sistema de 57 nodos, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas mediante CPU-GPU.

(NCCI)	$\triangle t$=32.55 μs			
	DN-Φ	**DNJC-Φ**	**DN-Φ_{RK}**	**DNJC-Φ_{RK}**
8	24.950e-02	24.950e-02	24.950e-02	24.950e-02
158	31.569e-03	31.569e-03	31.569e-03	31.569e-03
169	⋮	19.589e-12	⋮	19.589e-12
309	28.534e-05		28.534e-05	
460	23.381e-09		23.381e-09	
611	9.459e-14		9.459e-14	
Tiempo (ms)	**4067**	**1232**	**1794**	**606**

En las Tablas 5.5 y 5.5 se muestra el proceso de convergencia al EEP de la red de 57 nodos, se observa que el método de FB requiere de un total de 209 ciclos de integración para llegar a la solución, mientras que el método de DN necesitó de 611 ciclos, de los cuales 8 son asociados a los ciclos iniciales para obtener el ciclo base y 603 ciclos que son proporcionales a 4 iteraciones del cálculo de la matriz de transición de estados. Por último, el procedimiento de DNJC y sus contrapartes paralelas necesitaron de un total de 169 ciclos, que al igual que el método de DN, 8 periodos de integración son asociados a la obtención del ciclo base, más 150 son relacionados a 1 iteración del cálculo de la matriz de identificación y finalmente 11 ciclos que conciernen al cálculo de las variables de estado al ciclo limite manteniendo el Jacobiano constante.

De acuerdo a los datos presentados en las Tablas 5.5 y 5.5 , el método de DN y su contraparte paralela requirieron de un esfuerzo equivalente al 292.34 % de las iteraciones obtenidas por el método de FB, por lo que se traduce en un mayor tiempo y esfuerzo computacional. Por otro parte, el método de DNJC y sus contrapartes paralelas en comparativa con el método convencional de FB necesitaron de un 80.86 % de las iteraciones de este ultimo para hallar la solución

periódica, por lo cual esta metodología demuestra disminuir considerablemente el NCCI necesarios para la obtención del EEP.

Las Figuras 5.10 y demuestran algunas formas de onda representativas de las variables de estado asociadas con el caso de estudio analizado. La forma de onda del voltaje en el nodo 38 se presenta en la Figura 5.10; en la cual se aprecia que la tensión en ese punto no se ve afectada por la presencia de elementos no lineales como los transformadores y hornos de arco. Sin embargo, en la Figura 5.11 existe una pequeña distorsión en el voltaje del nodo, esto debido a la presencia del horno de arco.

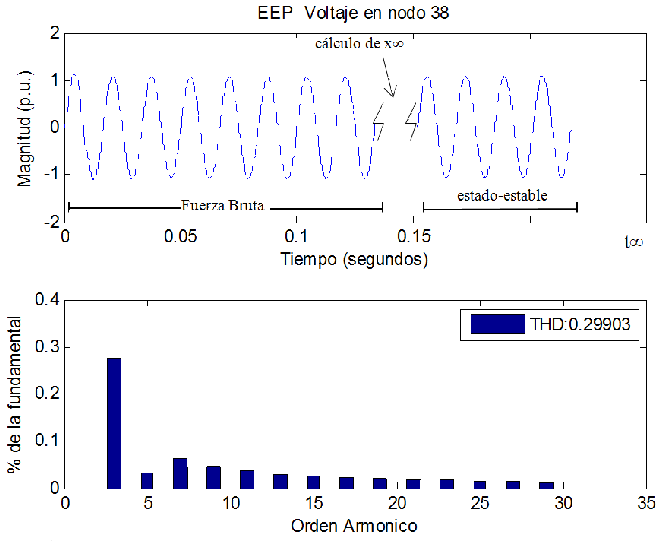


Figura 5.10: Voltaje en el nodo 38

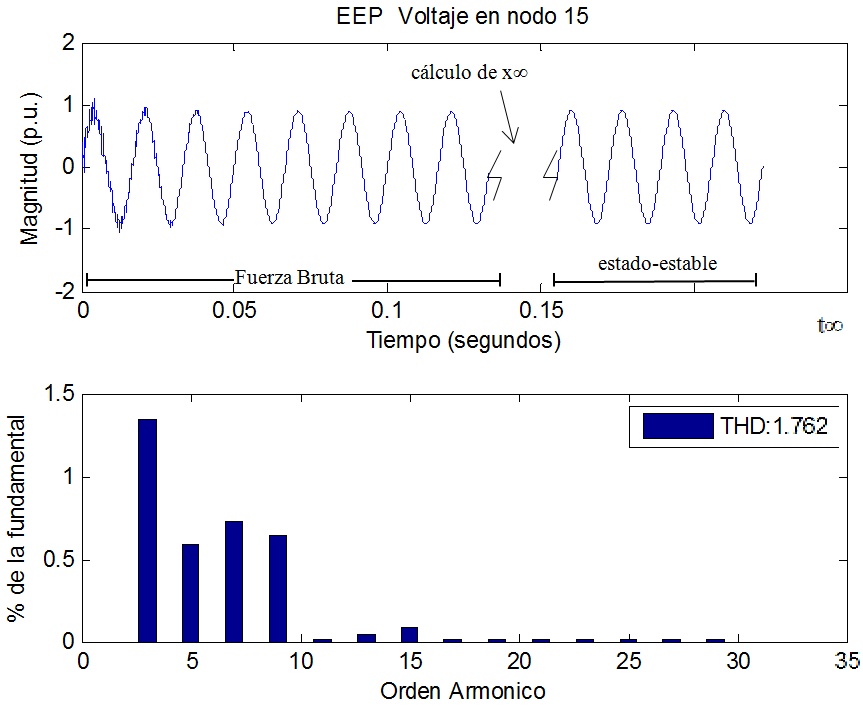


Figura 5.11: Forma de onda de del voltaje en el nodo 15. Caso de estudio 5.5

5.6. Sistema de IEEE de 118 nodos modificado

El sistema eléctrico de prueba para éste caso de estudio está formado por 184 líneas de transmisión, 118 bancos de capacitores, 53 generadores y 9 ramas magnetizantes, conectadas a los nodos 8, 26, 30 38, 63, 64, 65, 68 y 81. La dinámica del sistema presentado en este caso de estudio es representado por medio de 364 EDO's.

Las Tablas 5.6 y 5.6 muestran el número de aplicaciones requeridas por las distintas metodologías de solución en el dominio del tiempo. Se observa que los métodos de DN y DNJC requirieron de 737 y 374 ciclos de integración para llegar al EEP. En cambio, el método de FB necesitó de 4552 ciclos completos para determinar la solución periódica. Esto significa que el método de DN empleo un 16.19 % de las iteraciones necesarias por parte del método de FB para hallar la solución, mientras que el método de DNJC ejecutado bien sea en la CPU o GPU, necesitó de un equivalente al 8.21 % del NCCI calculados por el procedimiento de FB.

Tabla 5.9: Sistema de 118 nodos modificado del IEEE, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas en una computadora convencional

(NCCI)	$\triangle t$=32.55 μs		
	FB	**DN**	**DNJC**
8	14.41e-02	14.41e-02	14.41e-02
372	7.8147e-08	3.0028e-06	3.0028e-06
374	7.7789e-08	⋮	1.7008e-13
737	4.3407e-08	7.4429e-13	
⋮	⋮		
4552	9.7680e-11		
Tiempo(ms)	**123362**	**21025**	**10634**

Tabla 5.10: Sistema de 118 nodos, error de convergencia de distintas metodologías en el dominio del tiempo ejecutadas mediante CPU-GPU.

(NCCI)	$\triangle t$=32.55 μs			
	DN-Φ	**DNJC-Φ**	**DN-Φ_{RK}**	**DNJC-Φ_{RK}**
8	14.41e-02	14.41e-02	14.41e-02	14.41e-02
372	3.0029e-06	3.0029e-06	3.0029e-06	3.0029e-06
374	⋮	1.2634e-13	⋮	1.2634e-13
737	9.1282e-13		9.1282e-13	
Tiempo(ms)	**10598**	**5409**	**4875**	**2538**

En las Figuras 5.12 y 5.13 se muestra la evolución en el tiempo de algunas variables de estado representativas de la solución en EEP del caso analizado. La Figura5.12 muestra la forma de onda del voltaje existente en el nodo 64, el contenido armónico de esta forma de onda está asociado principalmente con el tercer armónico con magnitud cercana al 2.5 % por lo que se tiene un porcentaje del THD de 2.55 %. En la Figura 5.13 se presenta el flujo que circula por la rama magnetizante conectada al nodo 64 y su contenido armónico. Para ambos puntos analizados el porcentaje de THD se encuentra conforme a la Norma IEEE-519.

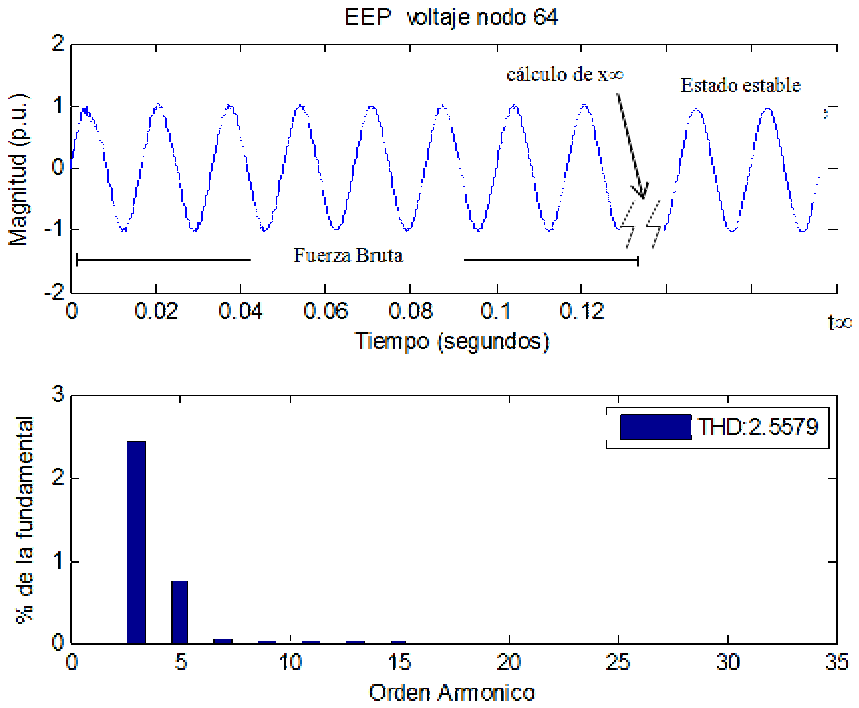


Figura 5.12: Voltaje en el nodo 64

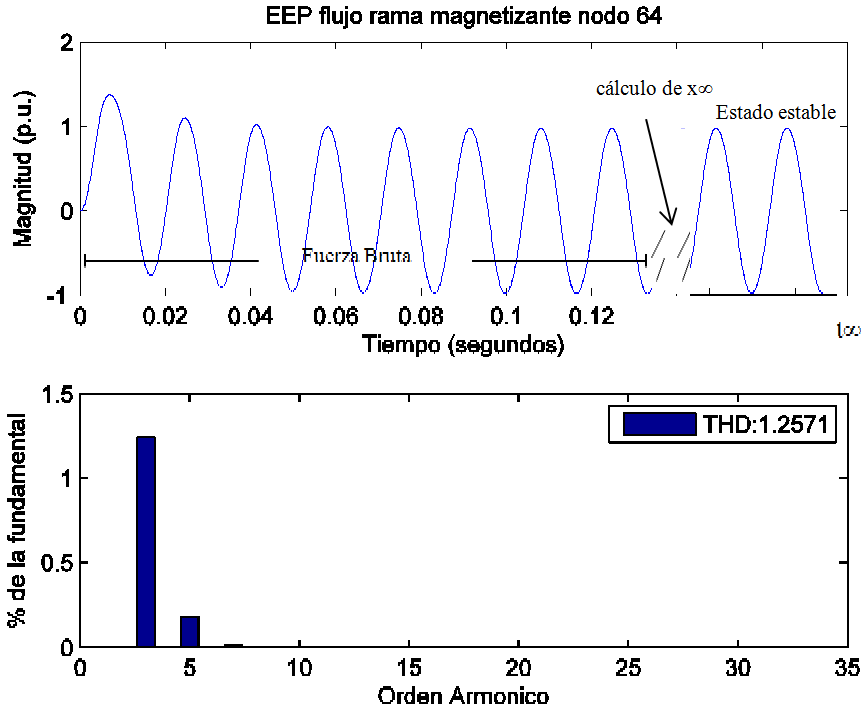


Figura 5.13: Flujo en la Rama Magnetizánte del nodo 64

5.7. Análisis de la eficiencia relativa (*Speed up*)

Conforme a las simulaciones realizadas y con la finalidad de investigar la eficiencia de la metodología propuesta, se presentan en las Tablas 5.11 y 5.12, los tiempos de simulación obtenidos y la eficiencia relativa (*Speed up*) para los distintos casos de estudio presentados.

La medición del *speed up* obtenido se realizó conforme a la metodología propuesta en [Marandi & Dinavahi (2010)],

$$E_{GPU-CPU} = \frac{T_{CPU}}{T_{GPU-CPU}} \tag{5.1}$$

Donde:

T_{CPU} : Tiempo de la simulación obtenido únicamente con CPU

$T_{GPU-CPU}$: Tiempo de la simulación obtenido con GPU-CPU

$E_{GPU-CPU}$: Eficiencia relativa obtenida mediante GPU-CPU

Tabla 5.11: Concentrado de tiempos de simulación de los casos de estudio

Sistema	**DN**	**DNJC**	**DN-Φ**	**DNJC-Φ**	**DN-Φ_{RK}**	**DNJC-Φ_{RK}**
3	78	58	155	97	167	105
14	401l	166	397	155	222	96
30	2352	694	1241	397	625	244
57	6966	1918	4067	1232	1794	606
118	21025	10634	10598	5409	4875	2538

Tabla 5.12: Eficiencia relativa obtenida en los casos de estudio

Sistema	**DN** *vs* **DN-Φ**	**DN** *vs* **DN-Φ_{RK}**	**DNJC** *vs* **DNJC-Φ**	**DNJC***vs* **DNJC-Φ_{RK}**
3	0.50	0.46	0.59	0.55
14	1.01	1.80	1.07	1.72
30	1.89	3.76	1.74	2.84
57	1.71	3.88	1.55	3.16
118	1.98	4.31	1.96	4.18

De acuerdo a la Tabla 5.12, se puede observar que para el caso del sistema de 3 nodos, en ninguna de las simulaciones se obtuvo resultados favorables, por el contrario se aumento aproximadamente al doble el tiempo de simulación utilizando las propuestas paralelas, por lo que se concluye que aplicar procesamiento en paralelo para sistemas de pequeña escala no resulta conveniente, ya que los tiempos de comunicación son mayores al beneficio obtenido.

Para el caso del sistema de 14 nodos, se observa que comienzan a obtenerse resultados favorables, siendo la propuesta del método de **DN-Φ_{RK}**, la que obtiene el *speed up* más elevado con 1.8, sin embargo, si se observa la Tabla 5.11, puede verse en los tiempos de simulación que el método de **DNJC-Φ_{RK}**, fue el que obtuvo el menor tiempo de simulación, por lo que se concluye que esta metodología resulta más conveniente de aplicar. Esto se debe a que el NCCI que necesita el método de DNJC comparado con el método de DN es 2.53 veces menor.

En el caso de los sistemas de 30, 57 y 118 nodos, de acuerdo a la Tabla 5.12, se observa un aspecto relevante. Al igual que para el sistema de 14 nodos, los *speed up* más grandes se obtuvieron en la primera y segunda propuesta

paralela del método de DN, sin embargo, a pesar de tener resultados de *speed up* menores, las propuestas paralelas del método de DNJC son más eficientes respecto al tiempo. Esto como se comentó anteriormente, debido a la reducción del NCCI necesarios para obtener el EEP que necesita el método de DNJC comparado con el método de DN.

5.8. Conclusiones

En el presente Capítulo se han presentado varios casos de estudio para ilustrar la eficiencia de las propuestas paralelas en CUDA de los métodos de DN y DNJC.

De igual forma, se comprobó la potencialidad del uso de procesamiento en paralelo basado en GPUs, obteniéndose una eficiencia relativa que oscila entre 1.01 y 4.31 para los distintos sistemas de prueba analizados. También se observa que la alternativa de iteración del método de Diferenciación Numérica planteada en [Semlyen y Medina, 1995] y que es evaluada en esta tesis, reduce considerablemente el NCCI. Por consiguiente esto se ve reflejado en una disminución del esfuerzo y tiempo computacional.

Por otro lado en base a las pruebas realizadas, se observó que conforme se incrementan las dimensiones del problema, resulta conveniente emplear procesamiento paralelo.

Al igual con los sistemas de prueba analizados, se demostró que al aplicar el método de DN se obtienen procesos de convergencia cuadrática que es la característica de convergencia típica de este método. Mientras que al mantener constante el Jacobiano, se obtiene una característica de convergencia lineal.

Capítulo 6

Conclusiones Generales y Sugerencias para Trabajos Futuros

6.1. Conclusiones Generales

- En esta Tesis se implementó y aplicó de manera eficiente el cálculo paralelo de la matriz de transición de estados, la cual es requerida para la obtención de la solución de las variables de estado al ciclo límite en el método Newton de Diferenciación Numérica. Para ello se utilizó como coprocesador una GPU y como plataforma de programación se manejo CUDA. Con la finalidad de obtener la máxima eficiencia de los problemas a evaluar el número de núcleos a utilizar del GPU seleccionado es el mismo que el número de variables de estado que modelan al sistema
- Esta técnica fue aplicada con éxito en el análisis de los sistemas de prueba modificados del IEEE de 14, 30, 57 y118 nodos monofásicos
- Se mostró en los casos de estudio que conforme se aumenta el número de EDO's que describen la dinámica del sistema analizado, se obtienen mejores eficiencias relativas.
- Con los resultados obtenidos se realizaron estudios de calidad de la energía asociados con el impacto en las formas de onda de variables tales como voltajes y corrientes en los diversos elementos del sistema debido a la incorporación de elementos no lineales y variantes en el tiempo, tales como hornos de arco eléctrico y ramas magnetizantes.
- En el desarrollo e implementación de esta técnica se aplicaron 2 herramientas de alto impacto: técnicas de procesamiento en paralelo y técnicas de acercamiento rápido al estado estacionario periódico. Se pudo comprobar que la incorporación de las herramientas computacionales anteriores ha redundado en una mejora significativa de la eficiencia computacional de los procesos de solución involucrados.
- Las técnicas de acercamiento rápido de las variables de estado al Ciclo Límite permitieron disminuir en promedio en un 30 % el número de periodos completos de integración requeridos con respecto a por la técnica FB. En todos y cada uno de los casos de estudio en los cuales se aplicó la técnica DN se observó la característica de convergencia cuadrática asociada con los métodos Newton.

- La eficiencia computacional de la técnica de aceleración de la convergencia de las variables de estado al Ciclo Límite DN fue incrementada con el uso de procesamiento en paralelo basado en GPUs. Se consiguió una eficiencia relativa máxima cercana a 4.31 para el caso del sistema de 118 nodos modificado.

6.2. Recomendaciones para trabajos futuros

Tomando como referencia la investigación realizada en esta tesis se proponen a manera de trabajos futuros:

- Con la finalidad de poder realizar un análisis comparativo entre GPU's de distintas compañías especializadas en su diseño y fabricación, tales como Intel, Ati-Radeon y AMD. Implementar la metodología propuesta en un lenguaje que promueva el cómputo paralelo heterogéneo.
- De igual forma otro de los procesos más demandantes durante la obtención de las variables de estado al ciclo limite es la inversión matricial por lo que sería interesante implementar de manera paralela esta metodología o inclusive utilizar otras técnicas de inversión más eficientes como es la factorización LU o el Gradiente Conjugado.
- Desarrollar modelos que permita incorporar dispositivos FACTS y Custom Power, con el fin de analizar sistemas de generación de energía alterna.
- Aplicar la metodología desarrollada en estudios dinámicos en tiempo real.
- Desarrollar modelos de fuentes de energía renovables tales como aerogeneradores, sistemas fotovoltaicos, celdas de combustible, etc.

Apendices

Apendice A

Modelado de los elementos monofásicos

A.1. Línea de Transmisión

El modelo de la línea de transmisión utilizado en este trabajo, se encuentra representado tal y como se modela una línea de transmisión corta, donde la resistencia R y la inductancia L en serie consideran la longitud total de la línea. El circuito equivalente para una línea de transmisión corta se muestra en la Figura A.1.

Figura 6.1: Modelo de la línea de transmisión monofásica

Al aplicar una LVK en el circuito que se muestra en la Figura A.1, se obtiene:

$$V_A - V_B = V_R - V_L \tag{6.1}$$

Donde:

$$V_R = iR \tag{6.2}$$

Y

$$V_L = L\frac{di}{dt} \tag{6.3}$$

Sustituyendo 6.2y 6.3 en 6.1y despejando se obtiene:

$$\frac{di}{dt} = \frac{V_A - V_B}{L} - \frac{iR}{L} \tag{6.4}$$

A.2. Generador

La Figura A.2 muestra el modelo del generador monofásico utilizado en este trabajo. El generador se modela por medio de una fuente sinusoidal de voltaje constante conectada a un inductor.

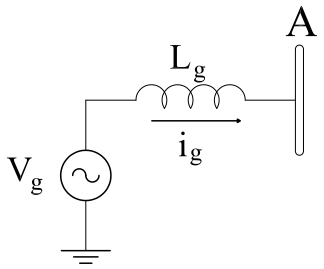


Figura 6.2: Modelo del generador monofásico

La aplicación de una LVK en el circuito de la Figura A.2 y despejando $\frac{di_g}{dt}$, proporciona:

$$\frac{di_g}{dt} = \frac{V_g - V_A}{L_g} \tag{6.5}$$

El voltaje en terminales del generador está determinado por la Ecuación (A.6)

$$V_g = Msin(\omega t + \phi) \tag{6.6}$$

donde:

M:Valor pico del voltaje

ϕ: Ángulo de fase expresado en radianes

ω: Velocidad angular

A.3. Banco de Capacitores

La Figura (A.3) muestra un banco de capacitores monofásico conectado a un nodo del sistema. El modelo considera tanto las corrientes incidentes, así como las corrientes que egresan del nodo, a excepción de la corriente en el capacitor.

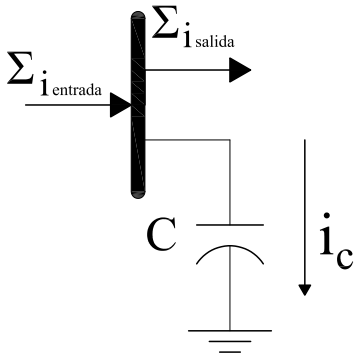


Figura 6.3: Modelo de Banco de capacitores monofásico

Donde :

$\sum_{ientrada}$: Sumatoria de corrientes que inciden en el nodo.

$\sum_{isalida}$: Sumatoria de corrientes que salen en el nodo.

Aplicando una LCK en el nodo se tiene,

$$\sum ientrada = \sum isalida - i_C \tag{6.7}$$

Por lo que el voltaje en el nodo se describe mediante la ecuación:

$$\frac{dV_C}{dt} = \frac{\sum ientrada - \sum isalida}{C} \tag{6.8}$$

A.4. Rama Magnetizante

En este trabajo el efecto de saturación en transformadores de potencia es representado mediante el modelo de una rama magnetizante. La Figura A.4 muestra la conexión en el nodo A de una rama magnetizante.

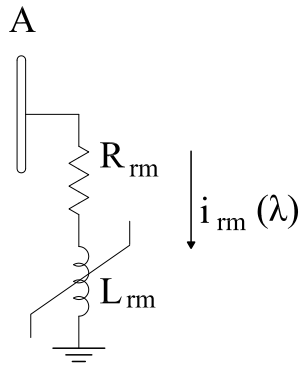


Figura 6.4: Modelo de rama magnetizante monofásica

Aplicando LVK al circuito de la Figura A.4 se obtiene la expresión (A.9):

$$V_A = V_{Rrm} + V_{Lrm} \tag{6.9}$$

Donde:

$$V_{Rrm} = R_{rm} \cdot i_{rm}(\lambda) \tag{6.10}$$

y

$$V_{Lrm} = L\frac{d\lambda}{dt} \tag{6.11}$$

Sustituyendo (A.10) y (A.11) en (A.9) y despejando $\frac{d\lambda}{dt}$ se tiene que:

$$\frac{d\lambda}{dt} = \frac{V_A - R_{rm} \cdot i_{rm}(\lambda)}{L} \tag{6.12}$$

El efecto de la saturación en la rama magnetizante es representado mediante una aproximación basada en un polinomio de grado n [Dick y Watson, 1981]. El comportamiento no lineal de la rama magnetizante esta expresado por medio de la relación no lineal:

$$i_{rm}(\lambda) = \lambda^n \tag{6.13}$$

En donde:

n: un valor impar, debido a que (A.13) tiene simetría impar. En este caso $n = 5$.

A.5. Horno de arco eléctrico

En la Figura A.5 se muestra la conexión de un horno de arco eléctrico al nodo A. El modelo del horno de arco eléctrico propuesto en [Acha et al., 1990] es utilizado en la presente tesis. Este modelo se basa en el principio de conservación de la energía y es representado por la Ecuación (A.14).

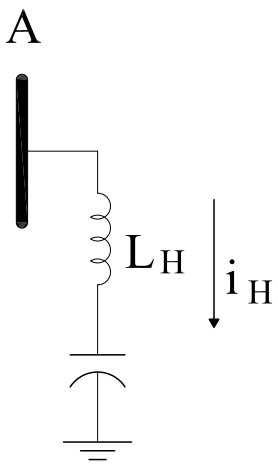


Figura 6.5: Modelo del horno de arco eléctrico monofásico

$$P_1 + P_2 = P_3 \tag{6.14}$$

Donde:

P_1: representa la potencia transmitida en forma de calor al ambiente externo.

P_2: representa la potencia que incrementa la energía interna en el arco y qué por consiguiente afecta su radio.

P_3: representa la potencia total desarrollada en el arco y convertida en calor.

En la Ecuación 6.14 se considera que el efecto de enfriamiento es únicamente una función del radio r del arco. Por lo tanto:

$$P_1 = k_1 r^n \tag{6.15}$$

De hecho, también es función de la temperatura del arco, esta dependencia se considera menos insignificante y por lo tanto es ignorada con el fin de mantener el modelo simplificado. Por lo tanto, únicamente el radio r aparece como variable de estado y k_1 es una constante de enfriamiento. Si el ambiente alrededor del arco es caliente el enfriamiento del arco no depende de su radio por completo, así que en este caso $n = 0$. Si este no es el caso y el arco es grande, entonces el área de enfriamiento es principalmente su superficie lateral, así que $n = 1$. Si el arco es pequeño, entonces el área de enfriamiento es proporcional a el área de sección transversal en los electrodos, así que $n = 2$ [Acha et al., 1990].

El término P_2 es proporcional a la derivada de la energía dentro del arco, la cual es proporcional a r^2.

$$P_2 = k_2 r \frac{dr}{dt} \tag{6.16}$$

Finalmente:

$$P_3 = v \cdot i = \frac{\frac{k_3}{r^m}}{r^2} i^2 \tag{6.17}$$

En esta expresión la resistividad de la columna del arco se considera que es inversamente proporcional a r^m, donde $m = 0.2$, para reflejar el hecho de que el arco puede ser más caliente en su interior si tiene un radio grande.

Sustituyendo (A.15), (A.16) y (A.17) en (A.14), se obtiene la ecuación diferencial del arco:

$$k_1 r^n + k_2 r \frac{dr}{dt} = k_3 r^{-(m+2)} i^2 \tag{6.18}$$

El voltaje en el arco está dado por:

$$v = k_3 r^{-(m+2)} i \tag{6.19}$$

Ahora considerando el circuito de la Figura A.5. Por medio de la aplicación de LVK se tiene:

$$V_A = V_{LH} + V_{RH} \tag{6.20}$$

Donde:

$$V_{LH} = L_H \frac{di_H}{dt} \tag{6.21}$$

$$V_{RH} = k_3 r^{-(m+2)} i_H \tag{6.22}$$

Sustituyendo (5.22) y (5.23) en (5.21) y despejando el término $\frac{di_H}{dt}$ se tiene que:

$$\frac{di_H}{dt} = \frac{-k_3 r^{-(m+2)} i_H + V_A}{L_H} \tag{6.23}$$

y la derivada del radio del arco está dada por:

$$k_1 r^n + r \frac{dr}{dt} = \frac{k_3}{k_2} r^{-(m+3)} i_H^2 - \frac{k_1}{k_2} r^{(n-1)} \tag{6.24}$$

Bibliografía

Almeida, F., Giménez, D., Mantas, José M., & Vidal, Antonio M. 2008. *Introducción a la programación paralela*. PARANINFO CENGAGE Learning.

Alvarado, Fernando, Betancourt, Ramon, Clements, Kevin, Heyd, Gerald T., Huang, Garng, Ilic, Maria, Scala, Massimo La, Pai, M.A., Pottle, Chris, Talukdar, Sarosh, Ness, James Van, & Wu, Felix. 1992. Parallel Processing in Power Systems Computation. *Transactions on Power Systems*, **vol. 7**(2), pp. 629–638.

Aprille, T. J., & T. N. Trick, title = Steady State Analysis of Nonlinear Circuits with Periodic Inputs, journal = Proc. IEEE volume = vol. 60 month=Enero year = 1972 number = 1 pages = pp. 108–114.

Arrillaga, J., Medina, A., Lisboa, M., & M. Cavia, P. Sanchez. 1995. The Harmonic Domain. A Frame of Reference for Power System Harmonic Analysis. *IEEE Transactions on Power Systems*, **vol. 10**(1), pp. 443–440.

Arrillaga, J., Smith, B. C., Watson, N. R., & Wood, A. R. 2000. *Power System Harmonic Analysis*. New York: Wiley.

Bedrosian, D., & Vlach, J. 1992. An Accelerated Steady State Method for Networks With Internally Controlled Switches. *IEEE Trans. Circ. And Syst.*, **vol. 16**(7), pp. 520–530.

Borges, Carmen, & Falcao, M. Djalma. 1999. A Parallelization Strategy for Power Systems Composite Reliability Evaluation. *Federal University of Rio de Janeiro, VECPAR'98, LNCS 1573*, pp. 640–651.

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

Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J., & Menon, R. 2001. *Parallel Programming in OpenMp*. Morgan Kaufmann Publishers, Academic Press, San Francisico, USA.

Chang, G. W., Chin, Y. C., & Lee, S. H. 2006. Efficient approach to characterizing harmonic currents generated by a cluster of three-phase AC/DC converters. *Proc. Inst. Elect. Eng.*, **vol. 153**(5), pp. 742–749.

Chua, L. O., & Ushida, A. 1981. Algorithms for Computing Almost Periodic Steady-State Response of Nonlinear Systems to Multiple Input Frequencies. *IEEE Transactions on Circuits and Systems*, **vol. CAS-28**(10), pp. 953–971.

Dagum, L., & Menon, R. 1998. OpenMP: An Industry-Standard API for Shared Memory Programming. *Computational Science and Engineering*, **vol. 5**(1), pp. 46–55.

Debnath, J. K., Fung, W. K., Gole, A., & Filizadeh, S. 2012. Electromagnetic Transient Simulation of Large-Scale Electrical Power networks using Graphics Processing Units. *IEEE, 25th Canadian Conference on Electrical and Computer Engineering (CCECE)*.

Dommel, H. W., Yan, A., & Wei, S. 1986. Harmonics from Transformer Saturation. *IEEE Trans. Power Delivery.*, **vol. 1**(2), pp. 209–214.

Dommel, H. W., Perkins, B. K., & Marti, J. R. 1995. Nonlinear Elements in the EMTP: Steady-State Inicialization. *IEEE Trans. Power Syst.*, **vol. 10**(2), pp. 593–601.

Donde, V., & Hiskens, I. A. 2006. Shooting Method for Locating Grazing Phenomena in Hybrid Systems. *International Journal of Bifurcation and Chaos*, **vol. 16**(3), pp. 671–692.

Falcao, Djalma M. 1994. Parallel and Distributed Processing Applications in Power System Simulation and Control. *COPPE, Universidad Federal de Rio de Janeiro*, pp. 125–143.

Flynn, M.J. 1972. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, pp. 948–960.

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

Garcia, N., Acha, E., & Medina, A. 2001. Swift Time Domain Solutions of Electric Systems Using Parallel Processing. *Proceedings of the Sixth International Conference IASTED, Rodas, Grecia*, Julio, pp. 172–177.

Garcia, Norberto. 2005. Parallel Harmonic-Oriented Method for Large-Scale Electric Systems Based on MPI Programming and the Limit Cycle Method. *IEEE PowerTech 2005, San Petersburgo, Rusia*, **vol. 1**(1), pp. 134–137.

Garcia, Norberto, & Acha, Enrique. 2004. Periodic Steady-State Analysis of Large Scale Electric Systems Using Poincare Map and Parallel Processing. *IEEE Transactions on Power Systems*, **vol. 19**(4), pp. 1784–1793.

Garcia, Norberto, & Medina, Aurelio. 2003. Swift Time Domain Solution of Electric Systems Including SVSs. *IEEE Transactions on Power Delivery*, **vol. 18**(3), pp. 921–927.

Geist, A., A., Beguelin, & J., Dongarra. 1994. *PVM: Parallel Virtual Machine*. MIT Press.

Geist, A., A., Kohl, & M., Papadopoulus. 1996. PVM and MPI: a Comparison of Features. *Applied Mathematical Sciences Subprogram of the Office of Energy Research, U.S. Department of Energy*, Mayo.

Gopal, A. 2010. *DC Power Flow Based Contingency Analysis Using Graphics Processing Units*. Tesis de Maestría, Drexel University.

Gopal, A., D., Niebur, & S, Venkatasubramanian. 2007. DC Power Flow Based Contingency Analysis Using Graphics Processing Units. *Power Tech, 2007 IEEE Lausanne*, Julio, pp. 731–736.

Grama, A., G., Karyspis, V., Kumar, & A., Gupta. 2003. *Introduction to parallel computing*. Addison Wesley, Reino Unido.

Happ, H. H. 1969. Multi-Computer Configurations and Diakoptics: Dispatch of Real Power in Power Pools. *IEEE Trans. On PAS*, **vol. 88**(5), pp. 764–772.

Happ, H. H., & Undrill, J. M. 1969. Multi-Computer Configurations and Diakoptics: Dispatch of Real Power in Power Pools. *IEEE Trans. On PAS*, **vol. 88**(6), pp. 789–796.

Hatcher, W. L., M., Brasch F., & E., Van Ness J. 1977. A feasibility study for the solution of transient stability problems by multiprocessor structures. *IEEE Trans. On PAS*, **vol. 96**(6), pp. 1789–1797.

Hidrobo, F., & Hoeger, H. 1998. *Introduccion a MPI (Message Passing Interface)*. Centro Nacional de Calculo Cientifico, Universidad de Los Andes, Merida, Venezuela.

Howroyd, D. C. 1982. Case studies in distortion on the public supply system. *IEE Conference on Sources and Effects of Power System Disturbances*, pp. 215–220.

Hwang, K., & Briggs, F.A. 1988. *Arquitectura de Computadoras Y Procesamiento Paralelo*. McGraw-Hill.

Ibarrola, Antonio Camarena, Miranda, Natalia, Piccoli, Fabiana, & Chávez, Edgar. 2010. Using GPU to Speed up the Process of Audio Identification.

Jin, Lan. 94. Parallel Processing: Exploring the architectures and algorithms close relationship. *IEEE Potentials*, Diciembre, pp. 17–20.

Jing, Shi, Qi, Huang, & Jianbo, Yi. 2013. A Novel Method of Parallel Computation for the Whole Scene Test in Power System. *International Journal of Smart Grid and Clean Energy*, **vol. 2**(1).

Kamiabad, Amirhassan Asgari. 2011. *Implementing a Preconditioned Iterative Linear Solver Using Massively Parallel Graphics Processing Units*. Tesis de Maestría, Graduate Department of Electrical and Computer Engineering University of Toronto.

Kleiman, S., B., Smaalders, D., Stein, & D., Shah. 1992. *Implementing a Preconditioned Iterative Linear Solver Using Massively Parallel Graphics Processing Units*. Writing Multithreading Code in Solaris.

Kundert, K. S., White, J. K., & Sangiovanni, Vincentelli A. 1990. *Steady-State Methods for Simulating Analog and Microwave Circuits*. Kluver Academic Publishers, USA.

Lewis, B., & Berg, D.J. 1998. *Multithreaded Programming with Pthreads*. Prentice Hall.

Lian, Kuo-Lung, & Noda, Taku. 2010. A Time-Domain Harmonic Power-Flow Algorithm for Obtaining Nonsinusoidal Steady-State Solutions. *IEEE Transactions on Power Delivery*, Enero.

Magaña, Ernesto, Montesinos, Víctor H., Ramos, Antonio, & Medina, Aurelio. 2013. Periodic Steady State Determination of Power Systems Using Graphics Processing Units. *10th. International Conference on Electrical Engineering, Computing Science and Automatic Control, CCE 2013*, 30 de septiembre al 4 de Octubre.

Mahmoud, A. A., & Schultz, R. D. 1982. A method for analyzing harmonic distribution in A.C. power systems. *IEEE Trans. on Power Apparatus and Systems*, **vol. 101**(6).

Marandi, V. Jalili, & Dinavahi, V. 2009. Large-Scale Transient Stability Simulation on Graphics Processing Units. *IEEE Trans. on Power Apparatus and Systems*, pp. 1–6.

Marandi, V. Jalili, & Dinavahi, V. 2010. SIMD-Based Large-Scale Transient Stability Simulation on the Graphics Processing Unit. *IEEE Trans. Power Syst.*, **Vol. 25**(3), pp. 1589–1599.

Marinos, Z. A., Pereira, J. L. R., & Carneiro, Jr. 1994. Fast Harmonic Power Flow Calculation Using Parallel Processing. *IEE Proc.-Gener. Transm. Distrib.*, **Vol. 141**(1), pp. 27–32.

Medina, A. 1992. *Power Systems Modelling in the Harmonic Domain*. Tesis de Doctorado, Universidad de Canterbury, Christchurch Nueva Zelanda.

Medina, A., & Ramos, A. 2005. PVM and MT Parallel Processing Platforms Applied to the Computation of Driving Point Impedances in Power Systems. *WSEAS Transactions on Circuits and Systems*, **Vol. 4**(11), pp. 1702–1709.

Medina, Aurelio. 2001. METODOLOGÍAS AVANZADAS PARA EL MODELADO Y ANÁLISIS DE ARMONICOS Y SU IMPACTO EN LA CALIDAD DE LA ENERGIA. *comité Nacional de CIGRE, segundo congreso Bienal, Irapuato, Gto.*, Junio.

Medina, Aurelio, & Arrillaga, Jos. 1990. Sparsity-oriented hybrid formulation of linear multiports and its application to harmonic analysis. *IEEE Trans. on Power Delivery*, **Vol. 5**(3), pp. 1453–1458.

Medina, Aurelio, Ramos, Antonio, & Fuerte, Claudio R. 2003. Periodic Steady State Solution of Electric Systems With Nonlinear Components Using Parallel Processing. *IEEE Transaction on Power Systems*, **Vol. 18**(2), pp. 963–965.

Medina, Aurelio, Ramos, Antonio, & Fuerte, Claudio R. 2006. Efficient computation of the periodic steady state solution of nonlinear electric systems applying parallel processing techniques. *COMPEL: The International Journal for Computation and Mathematics in Electrical and Electronic Engineering*, **Vol. 25**(4), pp. 900–915.

Mora, R. 2008. *Aplicación de Procesamiento en Paralelo MPI a la Solución en Estado Estacionario Periódico en el Dominio del Tiempo de Sistemas Eléctricos No Lineales*. Tesis de Maestría, DEP-FIE-U.M.S.N.H.

Nakhla, M. S., & Branin, F.H. 1977. Determining the Periodic Response of Nonlinear Systems by a Gradient Method. *Circuit Theory Appl*, **Vol. 5**, pp. 255–273.

Nickolls, John, & Dally, William J. 2010. The GPU Computing Era. *IEEE Computer Society*, **Vol. 30**(2), pp. 56–69.

Ortega, Félix G. 2012. *Determinación del Estado Estacionario Periódico de Sistemas Eléctricos No-Lineales Mediante un Método Newton Basado en la Matriz Exponencial Discreta, Incorporando Técnicas de Dispersidad y Procesamiento en Paralelo*. Tesis de Maestría, DEP-FIE-U.M.S.N.H.

Owens, John D., Houston, Mike, Luebke, David, Green, Simon, Stone, John E., & Phillips, James C. 2008. GPU Computing. *Proceedings of the IEEE*, **Vol. 96**(5), pp. 879–899.

Oyama, Tsutomu, Kitahara, Tatsuya, & Serizawa, Yasuo. 1990. Parallel Processing for Power System Analysis Using Band Matrix. *IEEE Transactions on Power Systems*, **Vol. 5**(3), pp. 1010–1016.

Pardo, C. Fernando. 2002. *Arquitecturas Avanzadas*. Universidad de Valencia.

Parker, T. S., & Chua, L. O. 1989. *Practical Numerical Algorithm for Chaotic Systems*. Springer-Verlag, New York.

Patel, Rakesh J., Patel, Bhavesh N., & Trivedi, Hemnag P. 1990. Parallel Processing: an overview. *IEEE Potentials*, **Vol. 9**(3), pp. 40–42.

Peña, Rafael, & Medina, A. 2010. Fast Steady-State Parallel Solution of the Induction Machine. *XIX International Conference on Electrical Machines - ICEM 2010, Rome,*.

Quinn, M. J. 2004. *Parallel Programming in C with MPI and OpenMP*. McGraw- Hill, New York, USA.

Ramos, A. 2002. *Aplicación de Técnicas de Procesamiento en Paralelo a la Solución en Estado Periódico Estacionario de Redes Eléctricas con Componentes no Lineales y Variantes en el Tiempo*. Tesis de Maestría, DEP-FIE-U.M.S.N.H.

Ramos, A. 2007. *Técnica para la Generación Automática de Ecuaciones Diferenciales No Autónomas para Representar el Comportamiento Dinámico de Sistemas Eléctricos No-Lineales Incorporando Herramientas Avanzadas de Computo*. Tesis de Doctorado, DEP-FIE-U.M.S.N.H.

Rico, Omar A. 2012. *Análisis de Redes Eléctricas Utilizando Técnicas de Procesamiento en Paralelo de Grano Fino Basadas en OpenMP*. Tesis de Maestría, DEP-FIE-U.M.S.N.H.

Rico, Omar A., & Paz, Antonio Ramos. 2011. Using the OpenMP Platform for Parallel Processing of Single and Three Phase Electrical Networks. *International Congress ROPEC 2011*, Noviembre, pp. 115–120.

Sanchez, Carla R. 2010. *Algoritmos Paralelos para la resolución de problemas de mínimos cuadrados basados en transformaciones ortogonales sobre GPUs y multiprocesadores*. Tesis Fin de Master en Computación Paralela y Distribuida, Universidad Politécnica de Valencia.

Sato, M. 2002. OpenMP: Parallel programming API for shared memory multiprocessors and on-chip multiprocessors. *University of Tsukuba, Kyoto, Japon*, Octubre, pp. 109–111.

Segundo, J., & Medina, A. 2008. Periodic Steady-State Solution of Electric Systems Including UPFCs by Extrapolation to the Limit Cycle. *IEEE Transaction on Power Delivery*, **Vol. 23**(3), pp. 1506–1512.

Segundo, J., & Medina, A. 2010a. Computation of the Steady-State Solution of Nonlinear Power Systems by Extrapolation to the Limit Cycle Using a Discrete Exponential Expansion Method. *Freund Publishing House Ltd., International Journal of Nonlinear Sciences and Numerical Simulation*, **Vol. 11**(8), pp. 655–660.

Segundo, J., & Medina, A. 2010b. An Enhanced Process for the Fast Periodic Steady State Solution of Nonlinear Systems by PoincarÃ© Map and Extrapolation to the Limit Cycle. *Freund Publishing House Ltd., International Journal of Nonlinear Sciences and Numerical Simulation*, **Vol. 11**(8), pp. 661–670.

Semlyen, A., & Medina, A. 1995. Computation of Periodic Steady State in System With Nonlinear Components Using a Hybrid Time and Frequency Domain Methodology. *IEEE Trans. Power Syst.*, **Vol. 10**(3), pp. 1498–1504.

Semlyen, A., & Shlash, M. 2000. Principles of modular harmonic power flow methodology. *Proc. of the IEE, Pt. C*, **Vol. 147**(1), pp. 1–6.

Sigridt, Garcia M. 1998. *Solución de redes eléctricas en el dominio del tiempo con aceleración de la convergencia al ciclo limite*. Tesis de Licenciatura, DEP-FIE-U.M.S.N.H.

Skelboe, S. 1980. Computation of the Periodic Steady-State Response of Nonlinear Networks by Extrapolation Methods. *IEEE Transactions on Circuits and Systems*, **Vol. 27**(Marzo), pp. 161–175.

Snir, M., Steve, Otto, David, Lederman Steven H.and Walker, & Jack, Dongarra. 1999. *MPI-The Complete Reference Volume 1, The MPI Core Second Edition*. Massachusetts Institute of Technology.

Stavrakakis, G. S., C., Lefas, & A., Pouliezos. 1990. Parallel Processing Computer Implementation of a Real Time DC Motor Drive Fault Detection Algorithm. *IEE Proceedings*, **Vol. 137**(5), pp. 309–313.

Sun Microsystems, Inc. 1994. *Multithreaded Programming Guide*. Sunsoft, A Sun Microsystems, Inc. Business.

Usaola, J. 1990. *Régimen Permanente de Sistemas Eléctricos de Potencia con Elementos No lineales Mediante un Procedimiento Híbrido de Análisis en los Dominios del Tiempo y de la Frecuencia*. Tesis de Doctorado, Universidad Politécnica de Madrid, Escuela Técnica Superior de Ingenieros Industriales.

Werler, K., & Glavitsch, H. 1993. Computation of Transients by Parallel Processing. *IEEE Transactions on Power Delivery*, **Vol. 8**(3), pp. 1579–1585.

Xia, D., & Heydt, G.T. 1982. Harmonic power flow studies, part 1-formulation and solution, part 2-implementation and practical application. *IEEE Trans. on Power Apparatus and Systems*, **Vol. 101**(Junio), pp. 1257–1270.