IMPLEMENTACIÓN DE UN CLUSTER COMPUTACIONAL PARA LA SIMULACIÓN PARALELA DEL MODELO MATEMÁTICO DEL NODO SINUSAL

RENÉ ALEXANDER BARRERA CARDENAS

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
UNIVERSIDAD INDUSTRIAL DE SANTANDER
BUCARAMANGA
2006

IMPLEMENTACIÓN DE UN CLUSTER COMPUTACIONAL PARA LA SIMULACIÓN PARALELA DEL MODELO MATEMÁTICO DEL NODO SINUSAL

RENÉ ALEXANDER BARRERA CARDENAS

Trabajo de Grado para optar al titulo de ingeniero electrónico

DIRECTOR
CARLOS ANDRÉS ANGULO JULIO
Magíster(c) Ingeniero Electrónico

CODIRECTOR
DANIEL ALFONSO SIERRA BUENO
Doctor(c) Ingeniero Electrónico y Electricista

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
UNIVERSIDAD INDUSTRIAL DE SANTANDER
BUCARAMANGA
2006

A mi madre, Gladys Cardenas Gamboa,
a mi hermana Maria Fernanda Peña
y mis abuelos, Pedro Elías Cardenas
y Ana Licenia Gamboa

Rene Alexander Barrera Cardenas

AGRADECIMIENTOS

El autor expresa sus agradecimientos a:

Al Ing. Carlos Andres Angulo Julio, director del proyecto, por su asesoría y dirección en este trabajo de grado

Al Mag. Daniel Alfonso Sierra Bueno, codirector del proyecto, por su apoyo y valiosos aportes en las fases iniciales del proyecto.

A toda su familia, la familia Cárdenas, por que sin ellos no seria la persona que es hoy en día.

RESUMEN

TITULO:

IMPLEMENTACIÓN DE UN CLUSTER COMPUTACIONAL PARA LA SIMULACIÓN PARALELA DEL MODELO MATEMÁTICO DEL NODO SINUSAL[*]

AUTOR:

RENE ALEXANDER BARRERA CARDENAS[**]

PALABRAS CLAVES:

Nodo sinoatrial, nodo SA, potenciales de acción, corrientes iónicas, sistemas paralelos, clusters de computadoras, algoritmos paralelos, metodología PCAM, programación paralela, paso de mensajes (MPI), MPITB.

DESCRIPCION:

En este trabajo de grado se propone un algoritmo paralelo que simule el funcionamiento del nodo sinoatrial del corazón a partir del modelo matemático planteado por H. Zhang, A. V. Holden y M. R. Boyett. Se estudia el uso de técnicas de programación paralela basadas en el paso de mensajes mediante el estándar MPI para aplicarla en la implementación del algoritmo propuesto.

Para el diseño del algoritmo paralelo se utiliza la metodología PCAM – Partición, Comunicación, Aglomeración y Mapeo – enfocado a un ambiente de memoria distribuida. La implementación del algoritmo paralelo se realiza en el ambiente de programación MATLAB mediante la herramienta MPITB la cual se basa en la técnica de paso de mensajes MPI.

Para la ejecución del resultante programa paralelo se realiza la configuración de un cluster no dedicado con el sistema operativo Linux y el Middleware LAM-MPI.

---

[*] Proyecto de Grado

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería Eléctrica, Electrónica y Telecomunicaciones.
Carlos Angulo Julio.

ABSTRACT

TITLE:

IMPLEMENTATION OF A COMPUTATIONAL CLUSTER FOR THE PARALLEL SIMULATION OF THE SINUSAL NODE MATH MODEL*

AUTHOR:

RENE ALEXANDER BARRERA CARDENAS**

KEY WORDS:

Sinoatrial node, SA node, action potentials, ionic currents, parallel systems, clusters of computers, parallel algorithms, methodology PCAM, parallel programming, message passing interface (MPI), MPITB.

DESCRIPCION:

In this document a parallel algorithm that simulates the operation of the heart's sinoatrial node is presented. The algorithm is based on the mathematical model proposed by H. Zhang, A. V. Holden and M. R. Boyett. The technique of parallel programming MPI (messages passing interface) is studied and applied to develop the proposed algorithm.

For to design the parallel algorithm is used the methodology PCAM - Partition, Communication, agglomeration and Map - , this is focused to the distributed memory environment. The programming of the parallel algorithm is carried out in MATLAB by means of the MPITB toolbox; this toolbox is based on the MPI technique.

For the execution of the resultant parallel program, the configuration of a cluster not-dedicated was carry out with the operating system Linux and the Middleware LAM-MPI.

* Degree project

** School of Electronic Engineering. Carlos Angulo Julio.

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

INTRODUCCION

Dada la importancia del estudio del sistema cardiovascular y la exigencia cada vez más apremiante de la investigación científica, se hace necesaria la adaptación de nuevas técnicas de programación para el modelado de dicho sistema y en particular del modelado del nodo sinusal; con lo cual se proporcionaría un mejor estudio acerca de los diferentes elementos que perturban su normal actividad, así como también el surgimiento de nuevas técnicas que hagan posible una disminución de las patologías relacionadas con su disfunción.

La investigación científica es cada vez más exigente y a pesar del surgimiento de nuevas metodologías y lenguajes de programación, que buscan emplear al máximo los recursos *hardware* existentes con un solo procesador, como lo es el uso de hilos, *pipeline*, la programación orientada a objetos y la programación concurrente, se observa que los sistemas que integran múltiples procesadores en la mayoría de los casos son más adecuados para implementar problemas que exijan gran demanda de computo.

En un principio los sistemas multiprocesador sólo eran implementados en supercomputadoras, y debido a su alto costo tenían un uso limitado siendo inaccesibles para proyectos de la academia. Gracias al desarrollo de los sistemas operativos, la supercomputación ofrece una nueva perspectiva, ya que con los denominados *clusters* se permite la integración de maquinas independientes interconectándolas en un solo conjunto, dando al programador la apariencia de un

sólo supercomputador, convirtiéndose ésta en una atractiva opción para el desarrollo de problemas que requieren alta demanda computacional debido a su gran escalabilidad y relativo bajo costo.

Una de esas técnicas es el procesamiento paralelo, el cual permite ejecutar de manera concurrente, simultánea y sincrónica múltiples procesos en diferentes procesadores. Sin embargo no todo algoritmo es paralelizable, y si lo es, no garantiza que sea más eficiente que el secuencial; por ello este proyecto busca proponer un algoritmo paralelo que permita simular el modelo matemático del nodo sinusal y comparar su eficiencia con la de un algoritmo secuencial ya existente.

Este libro esta constituido de seis capítulos descritos brevemente a continuación:
El capitulo 1 centra su atención en el modelado del nodo Sinoatrial presentando las principales características y componentes que influyen en el planteamiento del modelo estudiado las cuales fueron tenidas en cuenta por los autores del mismo.
El capitulo 2 presenta las diferentes organizaciones paralelas haciendo principal énfasis en los sistemas de memoria distribuida, en especial los clusters, ya que es el tipo de computador paralelo mas accesible por su relación costo/beneficio, lo que permitió su implementación para el objeto de este proyecto.
En el capitulo 3 describe la metodología PCAM (Particion-Communicate-Agglomerate-Map) utilizada para la creación de algoritmos paralelos, además se definen algunas medidas utilizadas para el análisis de rendimiento.
La descripción de los lineamientos básicos utilizados para el desarrollo del algoritmo paralelo del nodo sinoatrial se realizan en el capitulo 4, allí se analizan dos posibles alternativas de descomposición del problema que luego son fusionadas para dar la base del algoritmo finalmente implementado.
El capitulo 5 muestra los resultados de la implementación del algoritmo paralelo del nodo sinoatrial, comparando el tiempo de respuesta obtenido del algoritmo paralelo con el conseguido para el algoritmo secuencial.

Finalmente las conclusiones, aportes y algunas recomendaciones de este trabajo se presentan en el capitulo 6.

En total el libro contiene cinco anexos: el anexo A hace referencia a las convenciones utilizadas para el modelo matemático. El anexo B muestra los valores iniciales y parámetros necesarios para la simulación del nodo sinoatrial. En el anexo C se documenta la manera de configurar una red de computadores como un cluster no dedicado. El anexo D realiza la fundamentación de MPI, protocolo base en la implementación del algoritmo paralelo y por ultimo el anexo E describe la toolbox MPITB, la cual fue utilizada para implementar el algoritmo en entorno MATLAB.

Capítulo 1

MODELADO DEL NODO SINOATRIAL

1.1 INTRODUCCION

El corazón es básicamente un músculo hueco con cuatro cámaras y compartimentos – las dos aurículas (cámaras superiores) y los dos ventrículos (cámaras inferiores). El corazón es el responsable de bombear la sangre a todo el cuerpo, esto es necesario para que todos los órganos y tejidos sean abastecidos con el oxigeno necesario para funcionar. Para que la sangre sea acumulada y bombeada, el corazón necesita de minúsculos impulsos eléctricos conducidos de los compartimentos superiores a los inferiores, este movimiento hace que se contraigan y esta contracción es lo que se conoce como latido del corazón. Así, el sistema eléctrico del corazón controla la velocidad de su latido cardiaco. El sistema incluye una red de vías que portan las señales eléctricas del corazón. Cuando funciona correctamente, el sistema eléctrico del corazón responde automáticamente según varíen las demandas de oxígeno del organismo.

El sistema eléctrico del corazón se presenta en la figura 1.1, básicamente esta compuesto por:

- El nodo sinusal o sinoatrial (nodo SA): es un aglomerado de células especializadas ubicadas en la región superior de la aurícula derecha que generan espontáneamente el impulso cardiaco normal. Es denominado el "marcapasos natural" del corazón ya que genera su frecuencia cardiaca.
- Vías internodales o auriculares: Vías más funcionales que anatómicas, que llevan el impulso eléctrico desde el nodo SA al nodo AV.

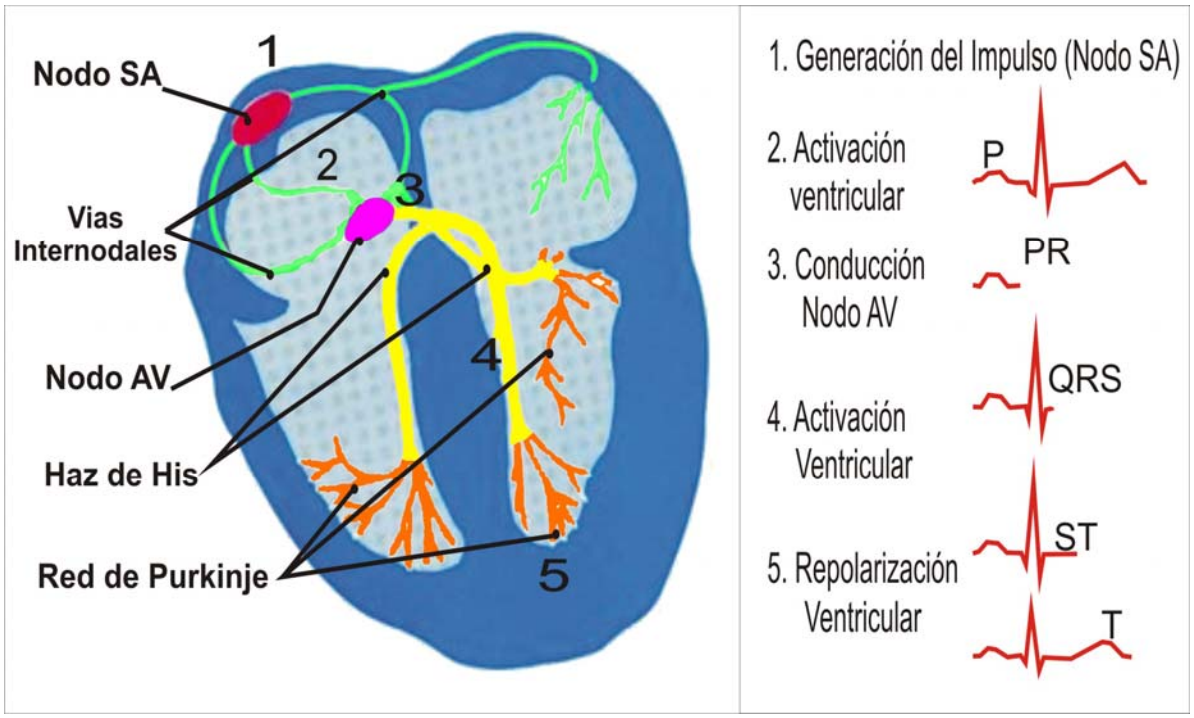

Figura 1.1. Sistema Eléctrico del Corazón y Relación con la onda electrocardiográfica (Fuente: Autor)

- El nodo auriculoventricular (nodo AV): es un conjunto de células especializadas que permiten el único paso de la electricidad entre las aurículas y los ventrículos. Por eso es denominado el "puente eléctrico" entre las aurículas y los ventrículos. Este paso acarrea un retraso fisiológico del impulso cardiaco lo cual logra un acoplamiento eficaz entre las contracciones auriculares y ventriculares.
- El haz de His: Sistema especializado en la conducción del impulso eléctrico desde el nodo AV a la red de fibras de Purkinje.
- La red de Purkinje: Red subendocárdica de células especializadas en la transmisión del impulso eléctrico para hacer que los ventrículos se contraigan.

El registro de la actividad eléctrica del corazón es llamado electrocardiograma (ECG). Este registro principalmente se divide en tres ondas: la onda P, asociada a las

contracciones auriculares, el complejo QRS, que se relaciona con las contracciones ventriculares y la onda T que muestra la relajación ventricular. En la figura 1.1 se relaciona el sistema eléctrico del corazón con las ondas del ECG. Básicamente las señales eléctricas creadas por el nodo S-A siguen una vía eléctrica natural atravesando las aurículas, el movimiento de las señales eléctricas hace que las aurículas se contraigan ayudando a impulsar la sangre hacia los ventrículos (Onda P). La señal se detiene al llegar al nodo AV para darle tiempo a los ventrículos de llenarse con sangre (Segmento entre la onda P y el complejo QRS). Luego, la onda eléctrica se propaga por el sistema His-Purkinje. El movimiento de la electricidad hace que los ventrículos se contraigan e impulsen la sangre hacia los pulmones y el cuerpo (complejo QRS). Por ultimo, las células ventriculares se repolarizan produciendo la relación ventricular (Onda T). Es importante notar que el primer evento electrocardiográfico (generación del impulso en el nodo SA) no se registra mediante un ECG convencional.

Este capitulo centra su atención en el modelado del nodo SA presentando las principales características y componentes que influyen en el planteamiento del modelo tenidas en cuenta por los autores del mismo [ZHANG00].

1.2 EL NODO SA

Como se comentó anteriormente, la actividad eléctrica del corazón se basa en la generación de impulsos eléctricos o potenciales de acción que son potenciales eléctricos presentes en la membrana celular debido a las diferencias de concentración iónica. El nodo sinoatrial (Nodo SA) es el punto de inicio de los potenciales de acción que controlan el ritmo cardiaco. Este tejido no es homogéneo, lo conforman tres tipos de células: células nodales (P), células de transición (T) y fibras ordinarias del

miocardio auricular. En circunstancias normales, el potencial de acción se inicia en una pequeña parte del nodo SA, llamado el sitio de marcapasos líder, ubicado aproximadamente en el centro del nodo SA. Una vez iniciado, el potencial de acción se propaga asimétricamente hacia la periferia y después al músculo atrial. El sitio de marcapasos líder es dinámico y se considera muy probable que no exista una única célula que sirva de marcapasos, por el contrario, las fibras del nodo sinusal funcionan como si fueran osciladores acoplados eléctricamente que se descargan en sincronía, logrando una oscilación intermedia entre las células de oscilación rápida y las células de oscilación lenta. [DASIERRA23]

1.2.1 DIFERENCIAS REGIONALES

La heterogeneidad de nodo SA genera una variación en la actividad eléctrica formando dos regiones características: el centro y la periferia. En la figura 1.2 se presenta una comparación entre los potenciales de acción de las células periféricas y centrales en el nodo sinusal, observándose diferencias significativas. Se puede resaltar que el potencial de acción central tiene con relación al periférico:

- Menor Amplitud
- Mayor duración
- Valor pico menos positivo
- Velocidad de despolarización más lenta
- Actividad espontánea más lenta

Las diferencias se deben principalmente a la falta de uniformidad, ya que el tamaño de las células aumenta del centro hacia la periferia y experimentalmente se ha probado que existe una correlación entre el tamaño de la célula del nodo SA y su

capacidad eléctrica, teniendo las células del centro del nodo una capacidad eléctrica de aproximadamente 20 pF, mientras que las periféricas tienen alrededor de 65 pF. Se ha encontrado que las células más grandes soportan una mayor densidad de corriente durante el ciclo cardiaco, siendo esta densidad de corriente definida como la corriente total que atraviesa la célula dividida entre su capacidad eléctrica. [ZHANG00]

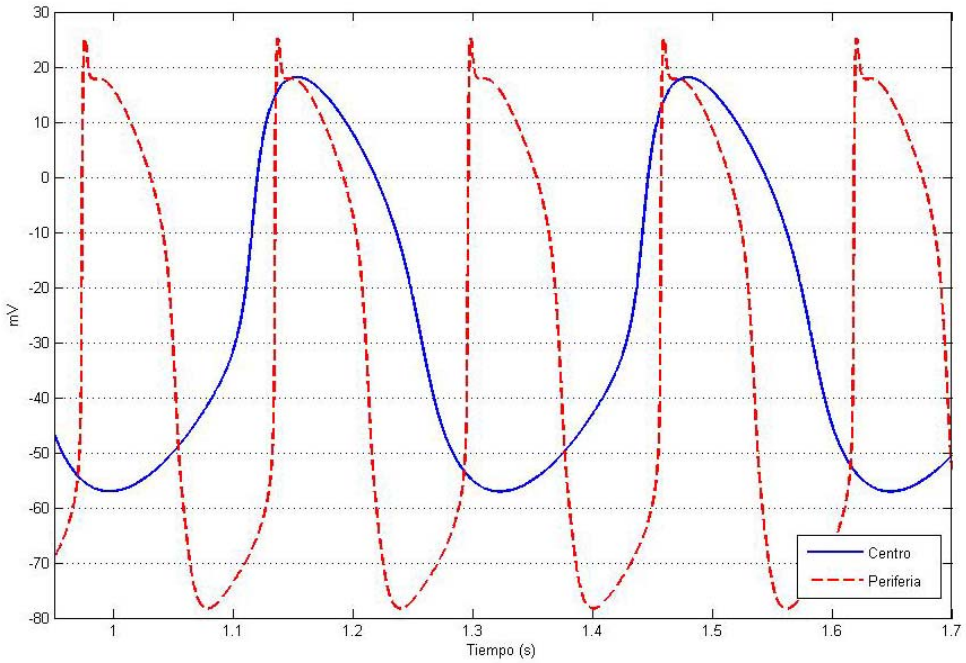

Figura 1.2 Potencial de acción en la periferia y centro del nodo SA. (Fuente: [ZHANG00])

1.2.2 PRINCIPALES COMPONENTES EN LA ACTIVIDAD ELECTRICA DEL NODO SA

La corriente a nivel celular se debe al movimiento de los iones a través de la membrana celular, por lo cual es llamada corriente iónica. Para el movimiento de los iones por medio de la membrana se tienen dos requisitos: un camino que les permita el paso y una fuerza que los impulse. Existen dos posibles fuerzas impulsoras a nivel de la membrana: la primera, dada por diferencias en la concentración de iones dentro y fuera de la célula generando corrientes de Difusión, y la segunda, produce corrientes de deriva por la diferencia de potencial existente a nivel de la membrana.

La figura 1.3 muestra un ejemplo de estas dos corrientes para los iones de sodio y cloro, nótese que para existir cualquiera de las dos corrientes es necesario un conducto que les permita el transito, dicho conducto en la membrana celular se llama canal iónico.

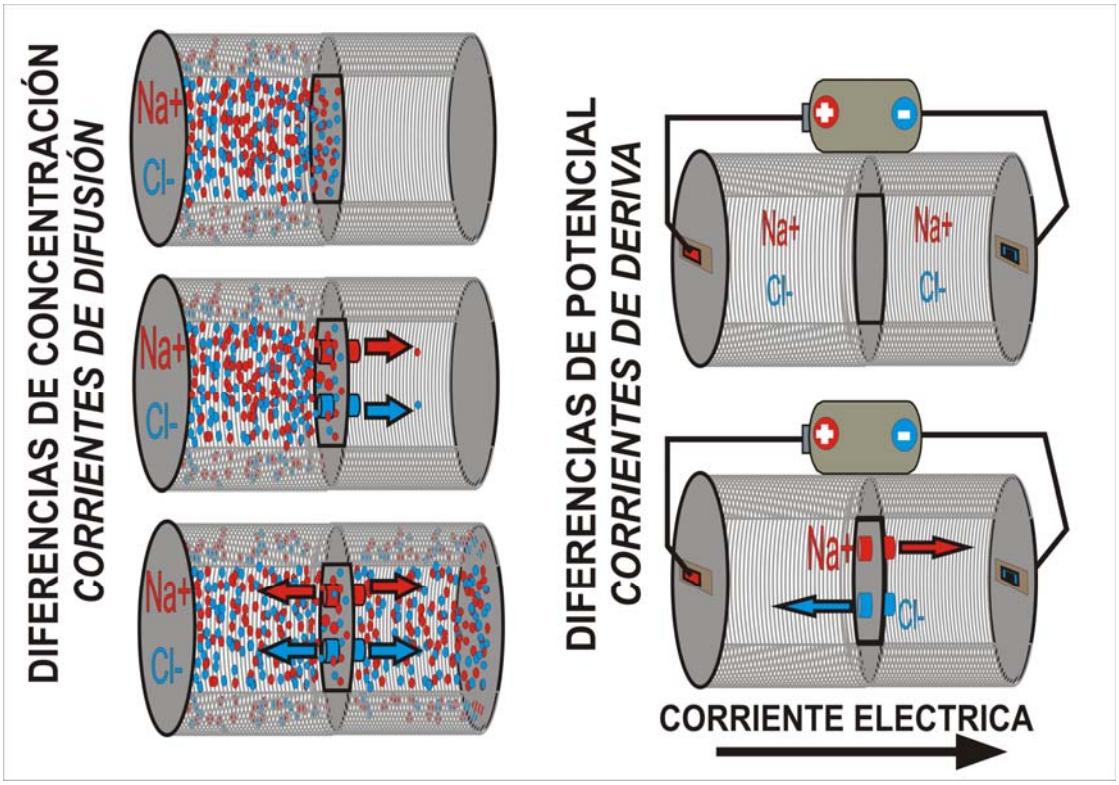

Figura 1.3. Ejemplo de Fuerzas impulsoras en la generación de corriente iónica

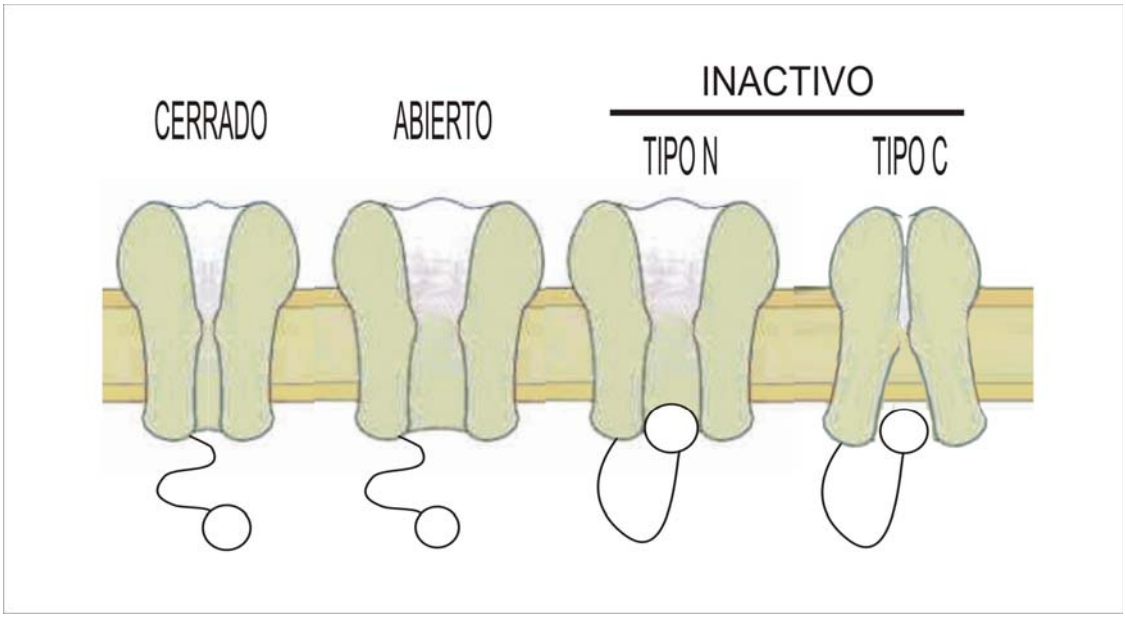

Figura 1.4. Canales Iónicos en sus diferentes estados. (Fuente: [VEGA01])

Los canales iónicos son túneles macro celulares de naturaleza proteica que atraviesan la membrana celular, su función principal es controlar el flujo de iones desde y hacia el interior de la célula para manipular su energía almacenada, posee una alta selectividad y se clasifican dependiendo del tipo de Ion que dejen pasar (sodio, potasio, calcio o cloro). Para cumplir con su función, los canales pueden conmutar entre distintos estados: abierto, cerrado o inactivo. La figura 1.4 muestra un canal iónico y sus posibles estados.

1.3 MODELADO DEL NODO SA

1.3.1 RELACIÓN BIOLÓGICA-ELÉCTRICA

La actividad de marcapasos del nodo SA es el efecto cooperativo de diferentes corrientes a través de canales iónicos en la membrana celular, lo cual genera los potenciales de acción para todo el sistema cardiaco. Dada la naturaleza biológica del nodo SA, para su modelamiento es conveniente relacionar cada uno de los componentes biológicos con sus equivalentes eléctricos. En la figura 1.5 se presenta un esquema que relaciona el modelo biológico de la electricidad con el modelo físico. De esta manera se pueden definir modelos para cada uno de sus componentes claves en la actividad eléctrica, estos son: canales iónicos, corrientes iónicas y potencial de acción.

1.3.2 MODELO DE LOS CANALES IÓNICOS

El movimiento de los iones a través de un canal es determinado por gradientes de concentración eléctrica y química. La diferencia de concentración genera un gradiente

químico que hace que los iones fluyan a través del canal iónico; de igual forma la diferencia de potencial genera un campo eléctrico que mueve los iones eléctricamente cargados. Cuando las fuerzas de impulso químico y eléctrico están exactamente balanceadas se genera un punto llamado potencial de Nernst o potencial de reversión E_{rev}.

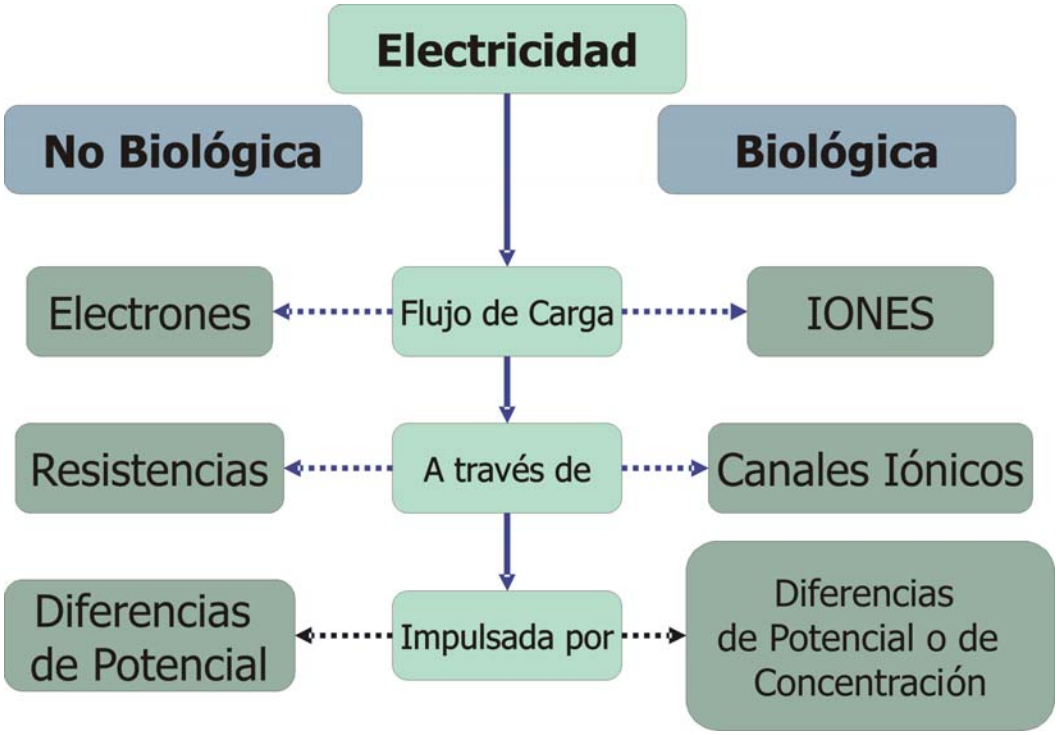

Figura 1.5. Modelo Biológico y No Biológico de la Electricidad [DASIERRA]

Si este potencial cambia, las fuerzas ya no están en equilibrio y los iones fluirán en dirección de la fuerza que domine. De esto se puede concluir que el flujo neto de electricidad a través de la membrana celular es función de las concentraciones iónicas y de las propiedades de los canales iónicos (Cantidad, conductancia, selectividad y conmutación).

Cuando un solo tipo de canal iónico se abre, el potencial de membrana de toda la

célula se va hacia el potencial de Nernst de ese canal. Los potenciales de Nernst de los cuatro iones principales a través de la membrana celular son aproximadamente los siguientes: sodio E_{Na}=+70 mV, potasio E_K=-98 mV, calcio E_{Ca}=+150 mV, cloro E_{Cl}=-30 a -65 mV. El signo positivo o negativo refleja el potencial intracelular relativo a un electrodo de referencia a tierra. Por lo tanto, el potencial de transmembrana total en un momento dado es determinado por cuáles canales están abiertos y cuáles cerrados, y por la capacidad y cantidad de los canales.

La conductancia de un canal (g) se define como la relación de la amplitud de la corriente en un canal (i), a la fuerza electromotriz, o voltaje (V).

$$g = \frac{i}{V} \tag{1.1}$$

Para modelar la apertura y el cierre de los canales iónicos, los autores del modelo introducen dos variables, una de activación y otra de desactivación en el canal, las cuales regulan el valor de conductancia del canal. La ecuación (1.2) relaciona el valor de conductancia en un canal según el valor de la variable de activación ($\mathbf{z_a}$) y la variable de desactivación ($\mathbf{z_d}$). Si este valor de conductancia se multiplica por la diferencia de potencial respecto al de equilibrio de Nernst se conocerá la corriente iónica a través del canal:

$$\begin{aligned} g &= g_{\max} \cdot z_a(V) \cdot z_d(V) \\ & z_a(V) \textit{ tiende a } 1 \\ & z_d(V) \textit{ tiende a } 0 \end{aligned} \tag{1.2}$$

Cada una de las variables de activación y desactivación es definida con un modelo de tipo Hodgkin-Huxley ***[DEM94],*** El cual utiliza una ecuación diferencial ordinaria de

primer orden de la forma 1.3, un valor de estado estacionario $Z_\infty(V)$ de la variable de activación/desactivación ***z*** con un valor de potencial de transmembrana ***V*** y una constante de tiempo $t_z(V)$ asociada.

$$\frac{dz(V,t)}{dt} = \frac{z_\infty(V) - z(V,t)}{\tau_z(V)} \qquad (1.3)$$

1.3.3 MODELO DE LA CORRIENTES IONICAS

Para obtener el modelo de las corrientes iónicas en la membrana, sus autores analizaron el efecto que tiene cada corriente sobre los potenciales de acción, para tal fin se estudió la influencia de bloquear cada corriente iónica sobre los potenciales de acción de la células periféricas y centrales, y así notar su posible contribución en la actividad eléctrica del nodo SA. Cada corriente iónica se modela con un conjunto de ecuaciones que acoplan el modelo de su respectivo canal iónico con el potencial de membrana y sus potenciales de equilibrio. Así cada corriente cuenta con variables de activación y desactivación correspondientes con la apertura y cierre de los canales iónicos. En la tabla 1.1 se presentan las corrientes iónicas involucradas en el modelo y en las tablas de la 1.2 a 1.9 se presentan sus respectivos modelos. En el anexo A se presentan las convenciones utilizadas para el modelo.

Tabla 1.1. Corrientes iónicas en la Membrana Celular

i_{Na}	Corriente de Na^{++}, sensitiva al TTX
$i_{ca,L}$	Corriente de Ca^{++}, de tipo lenta L ("long lasting")
$i_{ca,T}$	Corriente de Ca^{++}, de tipo rápida T ("Transient")
$i_{K,r}$	Corriente de K^{+}, de rectificación retardada rápida
$i_{K,s}$	Corriente de K^{+}, de rectificación retardada lenta
i_{sus}, i_{to}	Componentes sostenida y transitoria de la corriente sensitiva a la 4-AP (Canal K^{+})
i_f	Corriente activada por hiperpolarización (Canal para Na^{+} y K^{+})
i_b	Corriente de respaldo lineal "Background" (Na^{++} ,Ca^{++}, K^{+})
i_{NaCa}	Corriente de intercambio Sodio-Calcio Electrogenica (Na^{+} - Ca^{+})
i_P	Corriente de bomba Sodio-Potasio Electrogenica (Na^{+} / K^{+})
i_{CaP}	Corriente de bomba de Calcio dependiente de ATP

Tabla 1.2. Corrientes sensitivas a la 4-AP o 4 Amino-Piridina (i_{to} e i_{sus})

$$i_{to} = g_{to} qr(V - E_K) \quad (1.4)$$

$$q_\infty = \frac{1}{1 + e^{(V+59.37)/13.1}} \quad (1.5)$$

$$\tau_q = 10.1 * 10^{-3} + \frac{65.5 * 10^{-3}}{0.57 * e^{-0.08(V+49)} + 0.7174 * e^{0.1(V+50.93)}} \quad (1.6)$$

$$\frac{dq}{dt} = \frac{q_\infty - q}{\tau_q} \quad (1.7)$$

$$r_\infty = \frac{1}{1 + e^{-(V-10.93)/19.7}} \quad (1.8)$$

$$\tau_r = 2.98 * 10^{-3} + \frac{19.59 * 10^{-3}}{1.037 * e^{0.09(V+30.61)} + 0.369 * e^{-0.12(V+23.84)}} \quad (1.9)$$

$$\frac{dr}{dt} = \frac{r_\infty - r}{\tau_r} \quad (1.10)$$

$$i_{sus} = g_{sus} r(V - E_K) \quad (1.11)$$

Tabla 1.3. Corriente de potasio (K^+) de rectificación retardada lenta ($I_{K,s}$)

$$i_{K,s} = g_{K,s} x_s^2 (V - E_{K,s}) \quad (1.12)$$

$$E_{K,s} = \frac{RT}{F} Ln\left(\frac{[K^+]_o + 0.12[Na^+]_o}{[K^+]_i + 0.12[Na^+]_i}\right) \quad (1.13)$$

$$\alpha_{xs} = \frac{14}{1 + e^{-(V-40)/9}} \quad (1.14)$$

$$\beta_{xs} = e^{-V/45} \quad (1.15)$$

$$x_{s\infty} = \frac{\alpha_{xs}}{\alpha_{xs} + \beta_{xs}} \quad (1.16)$$

$$\tau_{xs} = \frac{1}{\alpha_{xs} + \beta_{xs}} \quad (1.17)$$

$$\frac{\mathrm{d}x_s}{\mathrm{d}t} = \frac{x_{s\infty} - x_s}{\tau_{xs}} \quad (1.18)$$

Tabla 1.4. Corriente de potasio (K^+) de rectificación retardada rápida ($I_{K,r}$)

$p_a = (1 - F_{K,r}) p_{a,f} + F_{K,r} p_{a,s}$	*(1.19)*
$i_{K,r} = g_{K,r} p_a p_i (V - E_K)$	*(1.20)*
$p_{a,f\infty} = p_{a,s\infty} = \frac{1}{1 + e^{-(V+14.2)/10.6}}$	*(1.21)*
$\tau_{p_{a,f}} = \frac{1}{37.2e^{(V-9)/15.9} + 0.96e^{-(V-9)/22.5}}$	*(1.22)*
$\tau_{p_{a,s}} = \frac{1}{4.2e^{(V-9)/17} + 0.15e^{-(V-9)/21.6}}$	*(1.23)*
$\frac{\mathrm{d}\, p_{a,f}}{\mathrm{d}\, t} = \frac{p_{a,f\infty} - p_{a,f}}{\tau_{p_{a,f}}}$	*(1.24)*
$\frac{\mathrm{d}\, p_{a,s}}{\mathrm{d}\, t} = \frac{p_{a,s\infty} - p_{a,s}}{\tau_{p_{a,s}}}$	*(1.25)*
$p_{i\infty} = \frac{1}{1 + e^{-(V+19.6)/10.1}}$	*(1.26)*
$\tau_{p_i} = 0.002$	*(1.27)*
$\frac{\mathrm{d}\, p_i}{\mathrm{d}\, t} = \frac{p_{i\infty} - p_i}{\tau_{p_i}}$	*(1.28)*

Tabla 1.5. Corriente activada por hiperpolarización (i_f)

$i_f = i_{f,Na} + i_{f,K}$	*(1.29)*
$i_{f,Na} = g_{f,Na}\, y (V - E_{Na})$	*(1.30)*
$i_{f,K} = g_{f,K}\, y (V - E_K)$	*(1.31)*
$\alpha_y = e^{-(V+78.91)/26.62}$	*(1.32)*
$\beta_y = e^{(V+75.13)/21.25}$	*(1.33)*
$y_\infty = \frac{\alpha_y}{\alpha_y + \beta_y}$	*(1.34)*
$\tau_y = \frac{1}{\alpha_y + \beta_y}$	*(1.35)*
$\frac{\mathrm{d}\, y}{\mathrm{d}\, t} = \frac{y_\infty - y}{\tau_y}$	*(1.36)*

Tabla 1.6. Corriente de Na^+ sensitiva a la TTX

$$i_{Na} = g_{Na} m^3 h \left[Na^+\right]_o \frac{F^2}{RT} \frac{e^{(V-E_{Na})F/RT} - 1}{e^{VF/RT} - 1} V \quad ,|V| \neq 0 \qquad (1.37)$$

$$i_{Na} = g_{Na} m^3 h \left[Na^+\right]_o F.\left(e^{-E_{Na}F/RT} - 1\right) \quad ,|V| = 0$$

$$h = (1 - F_{Na}) h_1 + F_{Na} h_2 \qquad (1.38)$$

$$F_{Na} = \frac{9.52 * 10^{-2} * e^{-6.3*10^{-2}(V+34.4)}}{1 + 1.66 * e^{-0.225(V+63.7)}} + 8.6393 * 10^{-2} \qquad (1.39)$$

$$m_\infty = \left(\frac{1}{1 + e^{-(V+30.32)/5.46}}\right)^{1/3} \quad \text{Periferia}$$

$$m_\infty = \left(\frac{1}{1 + e^{-(V+25.32)/5.46}}\right)^{1/3} \quad \text{Centro} \qquad (1.40)$$

$$\tau_m = 4.56 * 10^{-5} + \frac{0.6247 * 10^{-3}}{0.832 * e^{-0.335(V+56.7)} + 0.627 * e^{0.082(V+65.01)}} \qquad (1.41)$$

$$\frac{dm}{dt} = \frac{m_\infty - m}{\tau_m} \qquad (1.42)$$

$$h_{1\infty} = \frac{1}{1 + e^{(V+66.1)/6.4}} \quad \text{Periferia}$$

$$h_{1\infty} = \frac{44.9 * e^{-(V+66.9)/5.57}}{44.9 * e^{-(V+66.9)/5.57} + \frac{1491}{323.3 * e^{(-(V+94.6)/12.9+1)}}} \quad \text{Centro} \qquad (1.43)$$

$$\tau_{h_1} = 5.977 * 10^{-4} + \frac{3.717 * 10^{-6} * e^{-0.2815(V+17.11)}}{1 + 3.732 * 10^{-3} * e^{-0.3426(V+37.76)}} \quad \text{Periferia}$$

$$\tau_{h_1} = 3.5 * 10^{-4} + \frac{0.03}{1 + e^{(V+40)/6}} \quad \text{Centro} \qquad (1.44)$$

$$\frac{dh_1}{dt} = \frac{h_{1\infty} - h_1}{\tau_{h_1}} \qquad (1.45)$$

$$h_{2\infty} = h_{1\infty} \qquad (1.46)$$

$$\tau_{h_2} = 3.556 * 10^{-3} + \frac{3.186 * 10^{-8} * e^{-0.6219(V+18.8)}}{1 + 7.189 * 10^{-5} * e^{-0.6683(V+34.07)}} \quad \text{Periferia}$$

$$\tau_{h_2} = 2.95 * 10^{-3} + \frac{0.12}{1 + e^{(V+60)/2}} \quad \text{Centro} \qquad (1.47)$$

$$\frac{dh_2}{dt} = \frac{h_{2\infty} - h_2}{\tau_{h_2}} \qquad (1.48)$$

Tabla 1.7. Corriente de Ca^{2+} de tipo lenta $i_{Ca\,L}$

$$i_{Ca,L} = \left[f_L d_L + \frac{0.006}{1+e^{-(V+14.1)/6}} \right](V - E_{Ca,L}) \qquad (1.49)$$

$$\alpha_{d_L} = -14.19\frac{(V+35)}{e^{-(V+35)/2.5}-1} - \frac{42.45V}{e^{-0.208V}-1}, V+35 \neq 0 \; y \; V \neq 0$$

$$\alpha_{d_L} = -14.19 * 2.5 - \frac{42.45V}{e^{-0.208V}-1}, \quad V+35=0 \qquad (1.50)$$

$$\alpha_{d_L} = -14.19\frac{(V+35)}{e^{-(V+35)/2.5}-1} - 42.45 * 0.208, \; V=0$$

$$\beta_{d_L} = \frac{5.71(V-5)}{e^{0.4(V-5)}-1}, \; V-5 \neq 0$$
$$\beta_{d_L} = 5.71/0.4, \qquad V-5=0 \qquad (1.51)$$

$$\tau_{d_L} = \frac{1}{\alpha_{d_L} + \beta_{d_L}} \qquad (1.52)$$

$$d_{L\infty} = \frac{1}{1+e^{-(V+22.3)/6}} \qquad (1.53)$$

$$\frac{\mathrm{d}\,d_L}{\mathrm{d}\,t} = \frac{d_{L\infty} - d_L}{\tau_{d_L}} \qquad (1.54)$$

$$\alpha_{f_L} = \frac{p_1(V+28)}{e^{-(V+28)/4}-1}, \quad V+28 \neq 0 \qquad \alpha_{f_L} = p_1 * 4, \; V+28=0$$
$$p_1 = 3.75 \;\; \textit{Periferia}; \;\; p_1 = 3.12 \;\; \textit{Centro} \qquad (1.55)$$

$$\beta_{f_L} = \frac{25}{1+e^{-(V+28)/4}} \text{ Centro} \qquad \beta_{f_L} = \frac{30}{1+e^{-(V+28)/4}} \text{ Periferia} \qquad (1.56)$$

$$\tau_{f_L} = \frac{1}{\alpha_{f_L} + \beta_{f_L}} \qquad (1.57)$$

$$f_{L\infty} = \frac{1}{1+e^{(V+45)/5}} \qquad (1.58)$$

$$\frac{\mathrm{d}\,f_L}{\mathrm{d}\,t} = \frac{f_{L\infty} - f_L}{\tau_{f_L}} \qquad (1.59)$$

Tabla 1.8. Corriente de Ca^{2+} de tipo rápida $i_{Ca\,T}$

$i_{Ca,T} = g_{Ca,T} d_T f_T (V - E_{Ca,T})$	*(1.60)*
$\alpha_{d_T} = 1068 e^{(V+26.3)/30}$	*(1.61)*
$\beta_{d_T} = 1068 e^{-(V+26.3)/30}$	*(1.62)*
$\tau_{d_T} = \frac{1}{\alpha_{d_T} + \beta_{d_T}}$	*(1.63)*
$d_{T\infty} = \frac{1}{1 + e^{-(V+37)/6.8}}$	*(1.64)*
$\frac{\mathrm{d}\, d_T}{\mathrm{d}t} = \frac{d_{T\infty} - d_T}{\tau_{dT}}$	*(1.65)*
$\alpha_{f_T} = 15.3 e^{-(V+71.7)/83.3}$	*(1.66)*
$\beta_{f_T} = 15 e^{(V+71.7)/15.38}$	*(1.67)*
$\tau_{f_T} = \frac{1}{\alpha_{f_T} + \beta_{f_T}}$	*(1.68)*
$f_{T\infty} = \frac{1}{1 + e^{(V+71)/9}}$	*(1.69)*
$\frac{\mathrm{d}\, f_T}{\mathrm{d}t} = \frac{f_{T\infty} - f_T}{\tau_{f_T}}$	*(1.70)*

Tabla 1.9. Corrientes de respaldo (background), bomba e intercambio

$i_{b,Na} = g_{b,Na}(V - E_{Na})$	*(1.71)*
$i_{b,K} = g_{b,K}(V - E_K)$	*(1.72)*
$i_{b,Ca} = g_{b,Ca}(V - E_{Ca})$	*(1.73)*
$i_{NaCa} = k_{NaCa} \frac{[Na^+]_i^3 [Ca^{++}]_o e^{0.03743 V \cdot \gamma_{NaCa}} - [Na^+]_o^3 [Ca^{++}]_i e^{0.03743 V \cdot (\gamma_{NaCa} - 1)}}{1 + d_{NaCa}\left([Na^+]_o^3 [Ca^{++}]_i + [Na^+]_i^3 [Ca^{++}]_o\right)}$	*(1.74)*
$i_p = \overline{i}_p \left(\frac{[Na^+]_i}{K_{m,Na} + [Na^+]_i} \right)^3 \left(\frac{[K^+]_o}{K_{m,K} + [K^+]_o} \right)^2 \frac{1.6}{1.5 + e^{-(V+60)/40}}$	*(1.75)*
$i_{CaP} = \frac{0.0339.[Ca^{++}]_i}{0.0004 + [Ca^{++}]_i}$ Periferia $\quad i_{CaP} = \frac{4.17 * 10^{-3}.[Ca^{++}]_i}{0.0004 + [Ca^{++}]_i}$ Centro	*(1.76)*

1.3.4 MODELO DEL NODO SA COMPLETO

Para el modelado matemático del nodo SA, el autor no sólo tiene en cuenta la naturaleza multicelular del tejido sinusal sino también la interacción electro-iónica entre el nodo SA y el músculo atrial que lo rodea. Estas características sumadas a las diferencias regionales de potencial de acción correlacionadas con la capacidad de la célula es la base para la generación del modelo matemático de la actividad eléctrica del nodo sinusal. El planteamiento del modelo presenta una serie de consideraciones de las cuales principalmente se destacan:

- Las medidas experimentales son obtenidas del corazón de un conejo, por lo tanto el modelo relacionado corresponde con el nodo sinoatrial del conejo.
- El nodo sinusal y el músculo atrial se consideran como una cadena de células de una longitud especifica L de 12,6 mm. La cadena de células del nodo tiene una longitud (L^s) de 3 mm y corresponde a la distancia del centro del nodo al músculo atrial. La cadena celular del músculo atrial tiene una longitud (L^a) de 9,6 mm, la cual es la longitud propia del músculo atrial.
- Dada la heterogeneidad del nodo SA, la capacidad eléctrica de las células en la cadena sinusal (C^s_m) varía de 20 pF a 65 pF desde centro hasta la periferia, sin embargo las células atriales se consideran homogéneas con una capacidad (C^a_m) de 65 pF para toda la cadena atrial.
- Las interacciones electro-iónicas entre las células son modeladas por la interacción difusiva de los potenciales de membrana.
- Se considera cada célula como un capacitor que recibe una corriente iónica total más una corriente de deriva debido a la variación espacial del potencial.

En la Tabla 1.10 se presentan las ecuaciones del nodo SA completo. Este modelo utiliza dos ecuaciones diferenciales parciales unidimensionales que corresponden al

modelo de una célula del nodo sinusal (1.77) y una célula del músculo atrial (1.78). En las ecuaciones el superíndice *s* hace referencia al nodo SA y el superíndice *a* corresponde al músculo atrial, *t* es el tiempo, *x* es la distancia respecto al centro del nodo, i^s_{tot} *o* i^a_{tot} es la corriente total (cuyos componentes se describen en la sección 1.3.3) y D^s *o* D^a es el coeficiente de acoplamiento que modela la interacción electrotónica entre las células.

Tabla 1.10. Ecuaciones Generales del Modelo del Nodo Sinusal Analizado.

$$\frac{\partial V^s(x,t)}{\partial t} = -\frac{1}{C^s_m(x)} \cdot i^s_{tot}(x,t) + D^s \frac{\partial^2 V^s(x,t)}{\partial x^2} \qquad 0 \le x \le 3\_mm \qquad (1.77)$$

$$\frac{\partial V^a(x,t)}{\partial t} = -\frac{1}{C^a_m(x)} \cdot i^a_{tot}(x,t) + D^a \frac{\partial^2 V^a(x,t)}{\partial x^2} \qquad 3 \le x \le 12.6\_mm \qquad (1.78)$$

$$C^s_m(x) = 20 + \frac{1.07 \cdot (x - 0.1)}{L^s (1 + 0.7745 e^{-(x-2.05)/(0.295)})} (65 - 20) \qquad (1.79)$$

$$\left. \frac{\partial V^s}{\partial x} \right|_{x=0} = 0 \qquad (1.80)$$

$$\left. \frac{\partial V^a}{\partial x} \right|_{x=L} = 0 \qquad (1.81)$$

$$i_{tot} = i_{Na} + i_{Ca,L} + i_{Ca,T} + i_{to} + i_{sus} + i_{K,r} + i_{K,s} + i_t + i_{b,Na} + i_{b,Ca} + i_{b,K} + i_{NaCa} + i_p + i_{CaP} \qquad (1.82)$$

$$E_{Na} = \frac{RT}{zF} Ln\left(\frac{[Na^+]_o}{[Na^+]_i}\right) \qquad (1.83)$$

$$E_{Ca} = \frac{RT}{zF} Ln\left(\frac{[Ca^{++}]_o}{[Ca^{++}]_i}\right) \qquad (1.84)$$

$$E_K = \frac{RT}{zF} Ln\left(\frac{[K^+]_o}{[K^+]_i}\right) \qquad (1.85)$$

La ecuaciones 1.80 y 1.81 consideran como condiciones de frontera la no existencia de flujo en los extremos. La ecuación 1.79 corresponde con el modelo de la capacidad

de las células del nodo SA el cual muestra una dependencia con la ubicación espacial (x) para estimar la variación de la capacidad con el tamaño de celular, con esto se tiene una capacidad de las células centrales (x=0) de aproximadamente 20 pF, en cambio que para las células de la periferia el valor es aproximadamente 65 pF. Como se mencionó anteriormente la capacidad eléctrica de las células del músculo atrial se considera constante (65 pF). Las ecuaciones de la 1.83 a 1.85 corresponden con el potencial de equilibrio para los iones sodio, calcio y potasio, respectivamente. Las convenciones utilizadas para el modelo del nodo SA son relacionas en el anexo A. En el Anexo B se encuentran los valores típicos de densidad de corriente de cada una de las regiones del nodo y además los valores iniciales del modelo original del nodo Sinusal [ZHANG00].

1.4 ALGORITMO SECUENCIAL PARA EL MODELO DEL NODO SA

El algoritmo secuencial presentado en esta sección corresponde con el algoritmo secuencial del nodo SA descrito en [DASIERRA23], el autor define cuatro pasos principales dentro del algoritmo de los cuales dos son iterativos y corresponden al calculo de las corrientes y potencial para cada valor de tiempo y espacio (x,t); sin embargo, para poder realizar un mejor análisis se especificarán en detalle las etapas iterativas separándolas en subetapas específicas de acuerdo a la secuencialidad y dependencia del algoritmo. En la figura 1.6 se presenta el diagrama de flujo secuencial para el algoritmo de simulación del nodo SA que es punto de comparación para el algoritmo paralelo generado dentro de este trabajo.

Como primer paso se tiene la definición de condiciones iniciales del potencial de membrana y cada una de las variables de activación y desactivación de las corrientes

iónicas, las cuales varían cumpliendo con su ecuación diferencial ordinaria de primer orden. El segundo paso consiste en la determinación de los valores de los parámetros involucrados en el modelo tales como: conductancias asociadas a los canales iónicos y concentraciones interna y externa de Ca, Na, y K. Dichos valores se relacionan en el anexo B en las tablas B.1 y B.2 respectivamente.

Una vez asignadas las condiciones iniciales y la determinación de los parámetros se realiza la etapa iterativa del cálculo de las corrientes iónicas. Esta etapa en realidad consiste en varias subetapas asociadas con cada corriente iónica (tablas 1.2 a 1.9) y subdivididas en tres pasos específicos:

1) Determinar la variable de activación asociada resolviendo la ecuación diferencial ordinaria de primer orden.
2) Resolver la ecuación diferencial de primer orden para encontrar el valor de la variable de desactivación.
3) Encontrar el valor de la corriente según su ecuación algebraica.

Es de recordar que las corrientes iónicas son dependientes del potencial de acción y del comportamiento dinámico de los canales iónicos asociados. Conocido el valor de cada corriente iónica se procede con la cuarta etapa, la determinación del potencial según la ecuación diferencial parcial (1.77), a ésta se asocian la estimación de la corriente iónica total en un determinado punto (x_i,t_i) y la estimación del valor de capacidad eléctrica en el espacio x_i.

Del algoritmo se resalta que el lazo asociado con el espacio es definido por la longitud de la cadena del nodo SA, mientras que el lazo asociado al tiempo de simulación es definido arbitrariamente por el usuario. [DASIERRA03] considera una duración de simulación de dos segundos para garantizar la finalización de los transitorios en la señal (primer segundo de simulación) y evaluar la oscilación de estado permanente. El paso que utiliza es 0.1 ms y 0.1 mm para tiempo y espacio

respectivamente; Se utiliza el método de Runge-Kutta de orden 4 para resolver las ecuaciones diferenciales de primer orden, mientras que para las ecuaciones parciales se utiliza el método de Euler. Más adelante se retoman todas estas consideraciones para realizar la comparación con el algoritmo paralelo

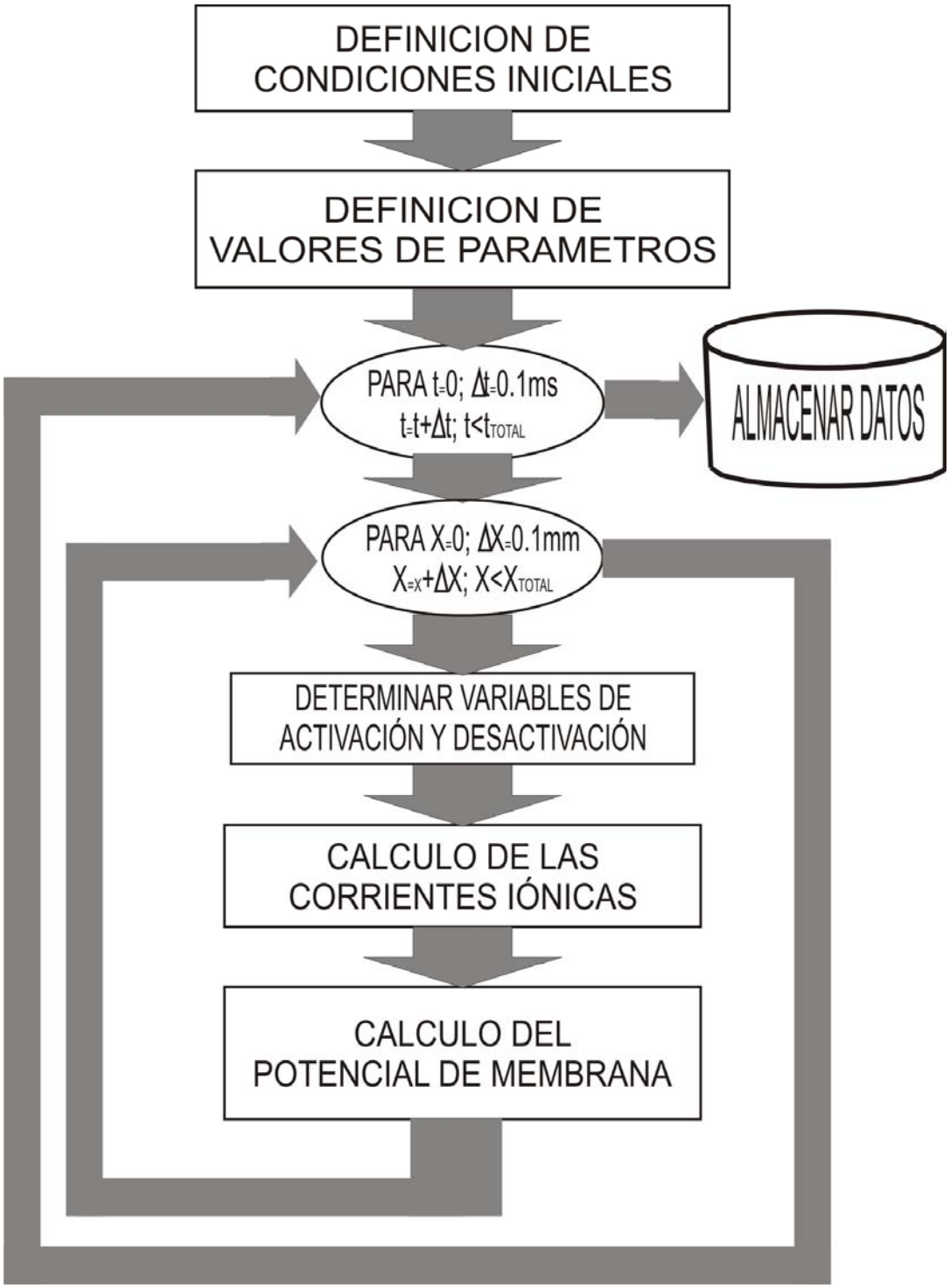

Figura 1.6. Diagrama de flujo secuencial para el modelo del nodo SA.

(Fuente: [DASIERRA23])

Capítulo 2

CLUSTERS DE COMPUTADORAS

2.1 INTRODUCCIÓN

La mayoría de los lenguajes de programación requieren que se especifiquen los algoritmos mediante una secuencia de instrucciones, así los procesadores ejecutan los programas encausando instrucciones maquina de una en una, cada una se ejecuta mediante una serie de rutinas (captar instrucciones y operandos, realizar la operación y almacenar los resultados). Esto ha generado que tradicionalmente el computador sea visto como una maquina secuencial. Esta visión del computador no es completamente cierta, ya que a nivel de micro-operación se generan al mismo tiempo múltiples señales de control, además la segmentación de las instrucciones en lo que se refiere al solapamiento de las operaciones de captación y ejecución, se ha utilizado desde hace tiempo. Ambos casos son ejemplos de funciones que se realizan en paralelo y es el mismo enfoque de la organización superescalar, que aprovecha el paralelismo de instrucciones disponiendo de varias unidades de ejecución, las cuales pueden realizar en paralelo varias instrucciones del mismo programa.

A medida que la tecnología de los computadores se ha desarrollado, y ha disminuido el costo del hardware del computador, los diseñadores de computadores han visto más y más posibilidades de paralelismo, normalmente para mejorar las prestaciones y, en algunos casos, para mejorar la fiabilidad. Uno de los primeros supercomputadores de histórica significancia fue el CRAY-1, el cual fue usado con bastante éxito en muchas aplicaciones que involucraban simulación de gran escala a principios de la década de los ochenta. El CRAY-1 no fue un computador paralelo, sin embargo, éste empleó un poderoso (en ese tiempo)

procesador vectorial con muchos registros vectoriales unidos a la memoria principal. Hoy, todos los supercomputadores son computadores paralelos. Algunos están basados en procesadores y redes especializadas, pero la mayoría están basados en hardware de consumo y sistemas operativos y aplicaciones software del tipo *open-source*.

Este capítulo presenta las diferentes organizaciones paralelas haciendo principal énfasis en los sistemas de memoria distribuida, en especial los clusters, ya que es el tipo de computador paralelo mas accesible por su relación costo/beneficio, pudiéndose crear a partir de una red de computadores, lo que permitió su implementación para el objeto de este proyecto.

2.2 TIPOS DE SISTEMAS PARALELOS

Comúnmente cuando se habla de un sistema paralelo se hace referencia a un conjunto de procesadores capaces de cooperar en la solución de un problema. Esta definición incluye supercomputadoras con cientos de procesadores, redes de estaciones de trabajo y máquinas con múltiples procesadores. Los computadores paralelos pueden ser clasificados de acuerdo a una variedad de características arquitecturales y modos de operación. En particular estos criterios incluyen:

- El tipo y el número de procesadores.
- La interconexión entre los procesadores, y
- el correspondiente esquema de comunicaciones, el control y sincronización globales y las operaciones de entrada/salida.

Una popular taxonomía para computadores paralelos es la descripción introducida por Michael Flynn a mediados de los sesentas. Basado en el número de datos y el flujo de instrucciones Flynn propone las siguientes categorías:

1. *SISD (Single Instruction, Single Data):* Un único procesador interpreta una única

secuencia de instrucciones, para operar con los datos almacenados en una única memoria. Los computadores monoprocesador caen dentro de esta categoría.

2. *SIMD (Single Instruction, Multiple Data):* Una única instrucción maquina controla paso a paso la ejecución simultánea y sincronizada de un cierto número de elementos de proceso. Cada elemento de proceso tiene una memoria asociada, de forma que cada instrucción es ejecutada por cada procesador, con un conjunto de datos diferentes. Los procesadores vectoriales y los matriciales pertenecen a esta categoría.

3. *MISD (Multiple Instruction, Single Data):* Se transmite una secuencia de datos a un conjunto de procesadores, cada uno de los cuales ejecuta una secuencia de instrucciones diferente. Esta estructura nunca ha sido implementada.

4. *MIMD (Multiple Instruction, Multiple Data):* Un conjunto de procesadores ejecuta simultáneamente secuencias de instrucciones diferentes con conjunto de datos diferentes. Los SMP (Multiprocesadores simétricos), los *clusters* y los sistemas NUMA (Acceso a memoria no uniforme) son ejemplos de esta categoría.

En la figura 2.1 se presenta un esquema de esta clasificación relacionando tres de los componentes básicos de un sistema de cómputo: unidad de control, memoria y unidad de proceso.

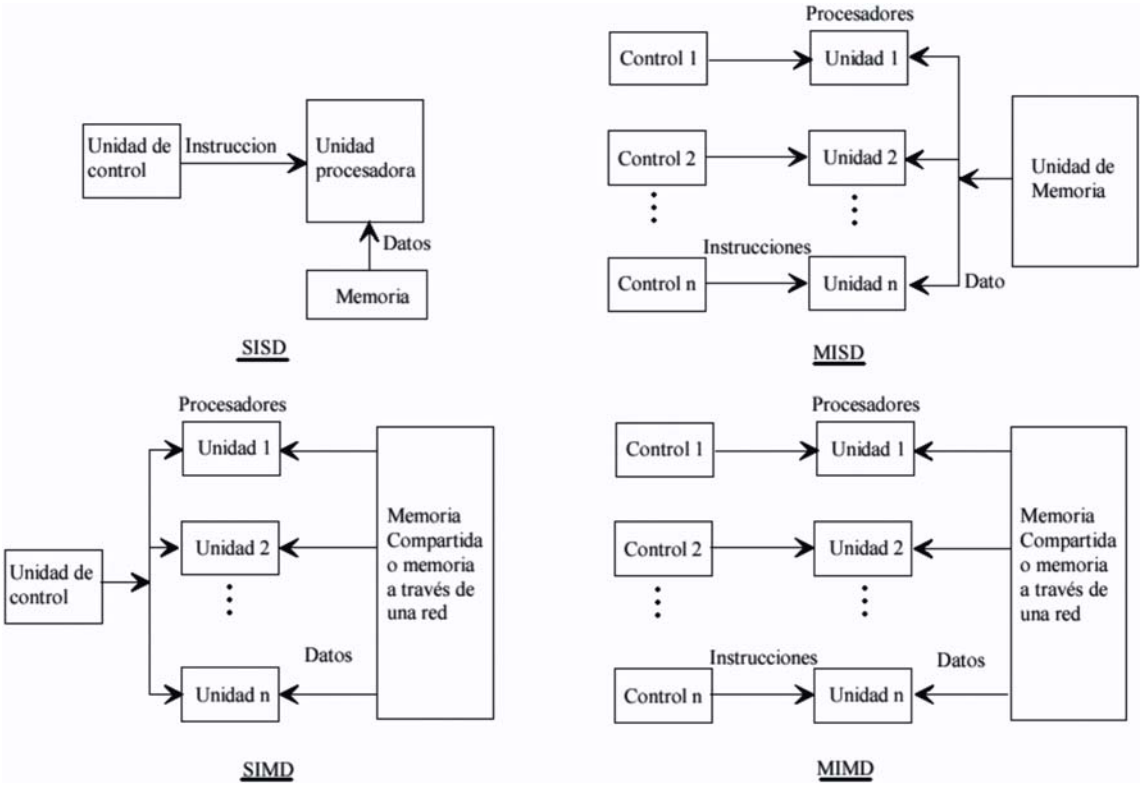

Figura 2.1. Taxonomía de Flynn (Fuente: [STALL20])

En la organización MIMD, los procesadores son de uso general y cada uno es capaz de procesar todas las instrucciones necesarias para realizar las operaciones apropiadas de los datos. Los computadores MIMD se pueden subdividir según la forma en que se comunican los procesadores; si estos comparten una memoria común, cada uno accede a los programas y datos almacenados en la memoria compartida mientras se comunican unos con otros. La forma mas común de este tipo de sistema se conoce como multiprocesador simétrico (SMP). En un SMP, varios procesadores comparten una única memoria mediante un bus compartido u otro tipo de mecanismo de interconexión. Una característica distintiva de estos sistemas es que el tiempo de acceso a memoria principal es el mismo para cualquier procesador. Un desarrollo mas reciente es la organización de acceso no uniforme a memoria (NUMA), que como el propio nombre lo indica, el tiempo de acceso a zonas de memoria diferentes puede diferir de un procesador a otro. Un conjunto de computadores monoprocesadores o de SMP, pueden interconectarse para formar un cluster. Se puede definir un *Cluster* como un grupo de

computadores completos interconectados, que trabajan conjuntamente como un único recurso de cómputo, creándose la ilusión de que se trata de una sola maquina. La comunicación entre los computadores se realiza mediante conexiones fijas o mediante un tipo de red, lo cual desde el punto de vista del ingeniero de hardware le da mejor escalabilidad que los MIMD de memoria compartida. Un nuevo paradigma de computación distribuida es la computación en G*rid* o malla en el cual todos los recursos de un indeterminado número de computadoras son englobados para ser tratados como un único supercomputador de manera transparente; estas computadoras englobadas no están conectadas o enlazadas firmemente, es decir; no tienen porque estar en el mismo lugar geométrico.

2.3 CLUSTERS DE COMPUTADORAS

Un *cluster* es una solución computacional conformada por un conjunto de sistemas computacionales, interconectados mediante alguna tecnología de red de alta velocidad, configurados de forma coordinada para dar la ilusión de un único recurso; cada sistema estará proveyendo un mismo servicio o ejecutando una (o parte de una) misma aplicación paralela. La característica inherente de un cluster es la de compartir recursos: ciclos de CPU (*Central Processing Unit*), memoria, datos y servicios. Los clusters están conformados por computadoras genéricas con uno o más procesadores, denominados nodos. Dichos nodos pueden estar dedicados exclusivamente a realizar tareas para el cluster, por lo que no requieren de monitor, teclado o Mouse; o pueden estar dedicados a diferentes actividades y se utilizarán los ciclos libres del procesador para realizar las tareas que requiera el cluster.

La idea de los clusters tomó impulso en los 90s, cuando se dispuso de microprocesadores de alto rendimiento, redes de alta velocidad, y herramientas estándar para computación distribuida (MPI[1], PVM[2]) a costos razonables. Hoy en día juegan un papel importante en la solución de problemas de las ciencias, las ingenierías y del comercio moderno. La tecnología

[1] ***Message Passing Interface***
[2] ***Parallel Virtual Machine***

de clusters ha evolucionado en apoyo de actividades que van desde aplicaciones de súper computo y software de misiones críticas, servidores Web y comercio electrónico, hasta bases de datos de alto rendimiento, entre otros usos.

En 1994, T. Sterling y D. Becker, trabajando en CESDIS (*Center of Excellence in Space Data and Information Sciences*) bajo el patrocinio del Proyecto de la Tierra y Ciencias del Espacio (ESS), construyeron un *cluster* de computadoras que consistía de 16 procesadores 486DX4, usando una red Ethernet a 10Mbps, con un costo relativamente bajo para época. El rendimiento del *cluster* era de 3.2 Gflops. Ellos llamaron a su sistema Beowulf, un éxito inmediato, y su idea de proporcionar sistemas basados en COTS (*Components Of The Shelve*) para satisfacer requisitos de cómputo específicos, se propagó rápidamente a través de la NASA y en las comunidades académicas y de investigación. En la actualidad, muchos *clusters* todavía son diseñados, ensamblados y configurados por sus propios operadores; sin embargo, existe la opción de adquirir *clusters* prefabricados.

En **[BREW97]** se enumeran cuatro beneficios que pueden conseguirse con un *Cluster*:

1. Escalabilidad Absoluta: Es posible configurar *Clusters* grandes, que incluso superan las prestaciones de los computadores independientes más potentes. Un *Cluster* puede tener decenas de maquinas, cada una de las cuales puede ser un multiprocesador.

2. Escalabilidad Incremental: Un *Cluster* se configura de forma que sea posible añadir nuevos sistemas al *Cluster* en ampliaciones sucesivas. Así un usuario puede comenzar con un sistema modesto y ampliarlo a medida que lo necesite, sin tener que sustituir el sistema de que dispone por uno nuevo que proporcione mayores prestaciones.

3. Alta Disponibilidad: Puesto que cada nodo del *Cluster* es un computador autónomo, el fallo de uno de los nodos no significa la perdida del servicio. En muchos casos, es el software el que proporciona automáticamente la tolerancia a fallos.

4. Mejor Relación Precio/Prestaciones: Al utilizar elementos estandarizados, es posible

configurar un *Cluster* con mayor o igual potencia de cómputo que un computador independiente mayor, a mucho menos costo.

Los ordenadores del cluster pueden tener, todos, la misma configuración de hardware y sistema operativo (cluster homogéneo), diferente rendimiento pero con arquitecturas y sistemas operativos similares (cluster semi-homogéneo), o tener diferente hardware y sistema operativo (cluster heterogéneo), lo que hace más fácil y económica su construcción.

2.4 CLASIFICACION DE LOS CLUSTERS

El término cluster tiene diferentes connotaciones para diferentes grupos de personas. Los tipos de *clusters*, según el uso que se les de y los servicios que ofrecen, determinan el significado del término para el grupo que lo utiliza. Tres distinciones básicas en los cluster son la base para su clasificación, el rendimiento, el balanceo de carga y la disponibilidad; si bien todas se pueden lograr con un cluster al optimizar una se pierden características de las otras, por lo tanto, según la aplicabilidad de los clusters se han desarrollado diferentes líneas tecnológicas.

2.4.1 ALTO RENDIMIENTO (High Performance)

Esta línea surge frente a la necesidad de súper computación para determinadas aplicaciones, lo que se persigue es conseguir que un gran número de máquinas individuales actúen como una sola máquina muy potente, este tipo de clusters se aplica mejor en problemas grandes y complejos que requieren una cantidad enorme de potencia computacional. Entre las aplicaciones más comunes de clusters de alto rendimiento se encuentra el pronóstico del estado del tiempo, astronomía, investigación en criptografía, simulación militar, simulación

de recombinaciones entre moléculas naturales y el análisis de imágenes. Este tipo de clusters en general está enfocado hacia las tareas que requieren gran poder computacional, grandes cantidades de memoria, o ambos a la vez, teniendo en cuenta que las tareas podrían comprometer los recursos por largos periodos de tiempo.

2.4.2 BALANCEO DE CARGA (High Throughput)

Este segundo tipo de tecnología de clusters, es el destinado al balanceo de carga. Surge el concepto de "clusters de servidores virtuales", lo cual es un cluster que permite que un conjunto de servidores de red compartan la carga de trabajo y de tráfico de sus clientes, aunque aparezcan para estos clientes como un único servidor. Al balancear la carga de trabajo en un conjunto de servidores, se mejora el tiempo de acceso y la confiabilidad. Además, como es un conjunto de servidores el que atiende el trabajo, la caída de uno de ellos no ocasiona una caída total del sistema. Este tipo de servicio es de gran valor para compañías que trabajan con grandes volúmenes de tráfico y trabajo en sus Web. Es de pensar que la imagen y el prestigio de una empresa que ofrece sus servicios por Internet se comprometen en la velocidad, la calidad y la disponibilidad de estos servicios.

Las características más relevantes de este tipo de clusters son:

- La independencia de datos entre las tareas individuales.
- El retardo entre nodos del cluster no es considerado un gran problema.
- La meta es el completar el mayor número de tareas en el tiempo mas corto posible.

2.4.3 ALTA DISPONIBILIDAD (High Availability)

Este tipo de clusters se conoce como "clusters de alta disponibilidad" o "clusters de redundancia".El objetivo aquí es la máxima disponibilidad de servicios y el rendimiento sostenido lo cual se puede lograr con el mantenimiento de servidores que actúen entre ellos

como respaldos de la información que sirven. La flexibilidad y robustez que proporcionan este tipo de clusters, los hace necesarios en ambientes de intercambio masivo de información, almacenamiento de datos sensibles y allí donde sea necesaria una disponibilidad continua del servicio ofrecido. Los clusters de alta disponibilidad permiten un fácil mantenimiento de servidores. Una maquina de un cluster de servidores se puede sacar de línea, apagarse y actualizarse o repararse sin comprometer los servicios que brinda el cluster. De esta forma cuando el servidor vuelva a estar listo, se reincorporara y volverá a formar parte del cluster.

2.5 COMPONENTES DE UN CLUSTER

Para que un cluster funcione como tal, no basta solo con conectar entre sí los ordenadores, sino que es necesario proveer un sistema de manejo del cluster, el cual se encargue de interactuar con el usuario y los procesos que corren en él para optimizar el funcionamiento. En general, un cluster necesita varios componentes de hardware y software para poder funcionar, los cuales se presentan en la figura 2.2 y listan a continuación:

- Nodos: Pueden ser simples computadores, sistemas multiprocesador o estaciones de trabajo.
- Sistema operativo: Debe ser de fácil uso y acceso y permitir además múltiples procesos y usuarios.
- Conexiones de red: los nodos de un cluster pueden conectarse mediante una simple red Ethernet con placas comunes (network adapters o NIC'S) o utilizarse tecnologías especiales de alta velocidad como: Fast Ethernet, Gigabit Ethernet, Myrinet, Infiniband, SCI, etc.
- Middleware: es un software de conectividad que permite ofrecer un conjunto de

servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas. Funciona como una capa de abstracción de software distribuida que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red).

- Protocolos de comunicación y servicios: Es la forma como se comunicará los diferentes nodos del cluster a través de las conexiones de red; comúnmente se utilizan dos: SSH (Security Shell) o RSH (Remote Shell).
- Aplicaciones: Es el conjunto de herramientas disponibles para el programador y el usuario final, por lo tanto pueden ser explícitamente paralelas o no.

2.5.1 EL MIDDLEWARE

El Middleware nos abstrae de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, proporcionando una API (Application Program Interface) para la fácil programación y manejo de aplicaciones distribuidas. Dependiendo del problema a resolver y de las funciones necesarias serán útiles diferentes tipo de servicios de middleware.

Por lo general, el middleware del lado cliente esta implementado por el sistema operativo subyacente, el cual posee las librerías que implementan todas las funcionalidades para la comunicación a través de la red.

El middleware es un software que generalmente actúa entre el sistema operativo y las aplicaciones con la finalidad de proveer a un cluster lo siguiente:

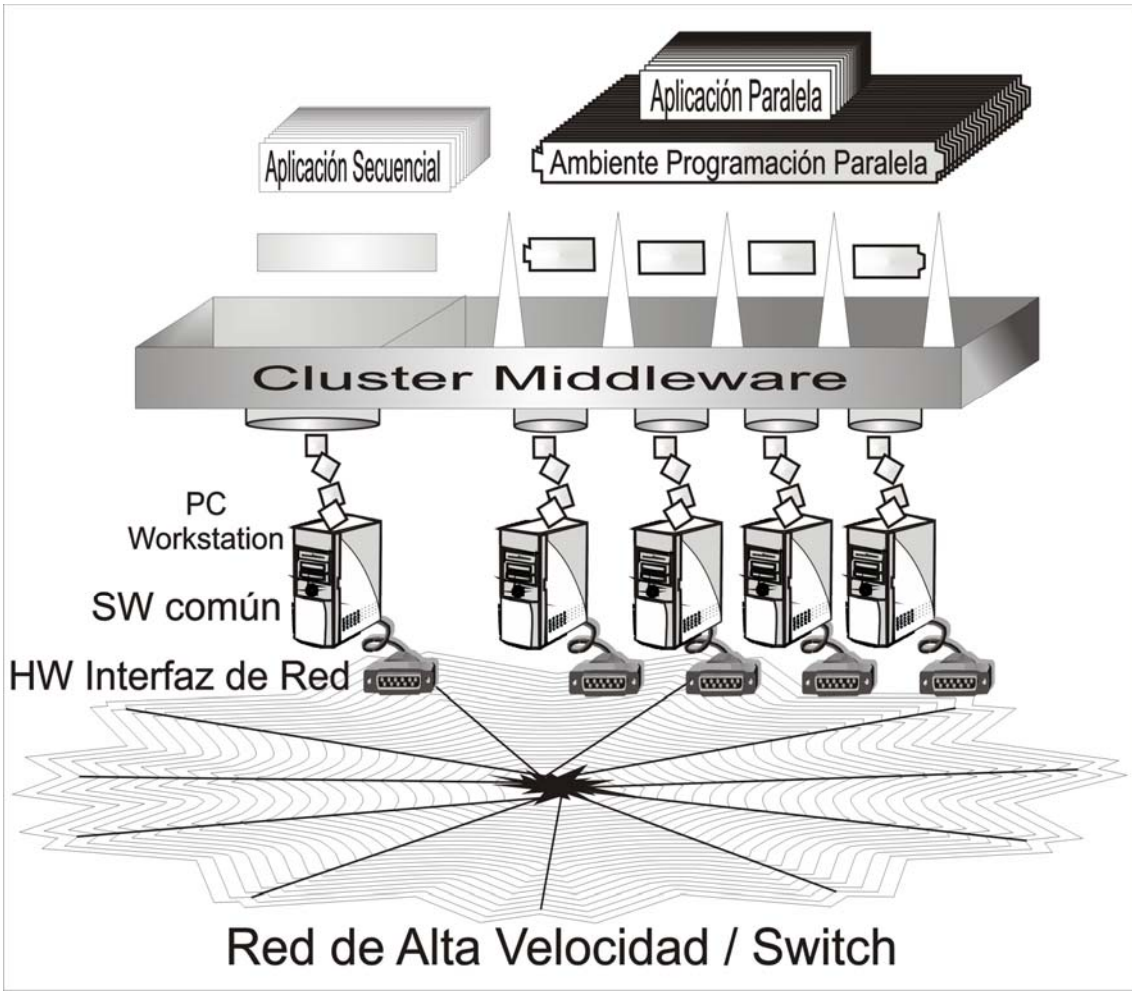

Figura 2.2. Componentes de un Cluster. (Fuente: Autor)

- Una interfaz única de acceso al sistema, denominada SSI (Single System Image), la cual genera la sensación al usuario de que utiliza un único ordenador muy potente.
- Herramientas para la optimización y mantenimiento del sistema, migración de procesos, checkpoint-restart[3], balanceo de carga, tolerancia a fallos, etc.
- Posibilidad de escalabilidad, debe poder detectar automáticamente nuevos

[3] Congelar uno o varios procesos, mudarlos de servidor y continuar su funcionamiento en el nuevo host

nodos conectados al cluster para proceder a su utilización.

Existen diversos tipos de middleware, como por ejemplo: MOSIX, OpenMOSIX, Condor, OpenSSI, etc.

El middleware recibe los trabajos entrantes al cluster y los redistribuye de manera que el proceso se ejecute más rápido y el sistema no sufra sobrecargas en un nodo. Esto se realiza mediante políticas definidas en el sistema (automáticamente o por un administrador) que le indican dónde y cómo debe distribuir los procesos con un sistema de monitorización, el cual controla la carga de cada CPU y la cantidad de procesos en él.

El middleware también debe poder migrar procesos entre nodos con distintas finalidades:

1. *Balancear la carga*: si un nodo está muy cargado de procesos y otro está inactivo, pueden transferirse procesos a este último para liberar de carga al primero y optimizar el funcionamiento;
2. *Mantenimiento de nodos*: si hay procesos corriendo en un nodo que necesita mantenimiento o una actualización, es posible migrar los procesos a otro nodo y proceder a desconectar del cluster al primero;
3. *Priorización de trabajos*: en caso de tener varios procesos corriendo en el cluster, pero uno de ellos de mayor importancia que los demás, puede migrarse este proceso a los nodos que posean más o mejores recursos para acelerar su procesamiento.

2.6 CONFIGURACION DE UN CLUSTER NO DEDICADO

Un cluster no dedicado o escasamente acoplado es una agrupación de computadores que aún siendo capaz de realizar procesamiento paralelo mediante librerías de paso de mensajes o de memoria compartida, no posee un sistema de instalación y gestión integrado que posibilite una recuperación rápida ante fallos y una gestión centralizada que ahorre tiempo al administrador. Sin embargo, permite la adaptación de nodos utilizados con diferentes propósitos aprovechando los ciclos no utilizados del procesador para realizar los cálculos asociados al cluster; así pues, al configurar una red determinada como un cluster no dedicado, no se afecta el uso independiente de cada nodo permitiendo utilizarlos para tareas fuera de las que realizará el cluster. Este hecho es muy importante si se consideran los equipos asociados a las áreas administrativas en un determinado campo universitario, ya que estos comúnmente no son utilizados a plenitud y podrían ser incorporados a un cluster no dedicado para aprovechar este poder de cómputo desperdiciado.

En la guía presentada en el anexo C se desarrollan los pasos necesarios para la configuración básica de un cluster no dedicado, el cual se utilizó para lograr los objetivos trazados dentro de esta tesis. Cabe aclarar que los pasos allí descritos son usualmente realizados por el administrador del cluster y fueron realizados como ejercicio académico, ya que existen paquetes de software que automatizan el proceso de instalación, de configuración y de administración de un *cluster*, denominados *toolkits*. Este conjunto de paquetes permite configurar un *cluster* completo en una fracción del tiempo que tomaría el hacerlo de forma manual. Estos *toolkits* utilizados para instalación automática de clusters pueden incluir una distribución de Linux; mientras que otros se instalan sobre una instalación existente de Linux. Sin embargo, incluso si primero se debe instalar Linux, los *toolkits* realizan la configuración e instalación de los paquetes requeridos por el *cluster* de forma automática. Del

conjunto de *toolkits* existentes se pueden mencionar a NPACI[4] *Rocks* y a OSCAR[5]. NPACI (*National Partnership for Advanced Computational Infrastructure*) *Rocks* es una colección de software de código abierto para crear un *cluster* sobre Red Hat Linux. *Rocks* instala tanto Linux como software para *clusters*. La instalación toma unos pocos minutos. OSCAR es una colección de software de código abierto que se instala sobre una instalación existente de Linux (Red Hat, Mandrake, Mandriva, Fedora).

2.7 HERRAMIENTAS PARA EL DESARROLLO DE APLICACIONES

Mediante la implementación de tareas paralelas es posible proveer solución a ciertos problemas computacionales de cálculos intensivos. El paralelismo se puede implementar mediante una aproximación del modelo cliente–servidor, llamado maestro–esclavo (*master–worker*). En el modelo maestro-esclavo se divide el problema computacional en tareas independientes, el maestro coordina la solución del problema computacional, asignando tareas independientes al resto de procesos (esclavos).

El maestro realiza la asignación inicial de las tareas a los esclavos, realiza sus tareas y espera por la finalización del procesamiento de las tareas en los esclavos, para proceder a recopilar los resultados desde los esclavos.

Para escribir programas paralelos, se puede hacer uso del modelo de paso de mensajes, con PVM o MPI, o se puede utilizar un modelo de memoria compartida, con OpenMP.

[4] http://www.rocksclusters.org/
[5] http://oscar.openclustergroup.org/

La diferencia entre el modelo de paso de mensajes y el modelo de memoria compartida está en el número de procesos o hilos activos. En un modelo de paso de mensajes, durante la ejecución del programa, todos los procesos se encuentran activos. Por el contrario, en un modelo de memoria compartida, sólo existe un hilo activo al iniciar y al finalizar el programa; sin embargo, durante la ejecución del programa, la cantidad de hilos activos puede variar de forma dinámica.

2.7.1 OpenMP

OpenMP es un conjunto de librerías para C y C++, regidas por las especificaciones ISO/IEC[6], basadas en el uso de directivas para ambientes paralelos. OpenMP tiene soporte para diferentes sistemas operativos como UNIX, Linux, y Windows.

Un programa escrito con OpenMP inicia su ejecución en un sólo hilo activo, llamado maestro, el cual ejecuta una región de código serial antes de que la primera construcción paralela se ejecute. Bajo el API de OpenMP dicha construcción se obtiene mediante directivas paralelas. Cuando se encuentra una región de código paralelo, el hilo maestro crea (*fork*) hilos adicionales, convirtiéndose en el líder del grupo. El hilo maestro y los nuevos hilos ejecutan de forma concurrente la sección paralela (realizan trabajo compartido). La unión al grupo (*join*) es el procedimiento donde al finalizar la ejecución de la región de código paralelo los hilos adicionales se suspenden o se liberan, y el hilo maestro retoma el control de la ejecución. Este método se conoce como *fork & join*.

En la figura 2.3 se indica como un hilo maestro encuentra una sección de código paralelo y crea hilos adicionales para ejecutar dicha sección. Una vez realizadas las tareas de ejecución, el hilo maestro retoma el control del programa.

[6] International Standard Organization - International Engineering Consortium

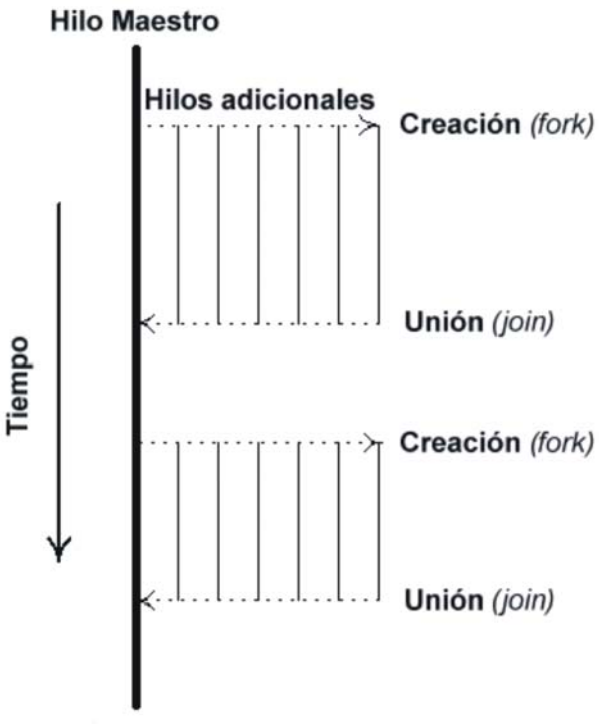

Figura 2.3. Modelo de ejecución fork & join

2.7.2 PVM

PVM es un conjunto de herramientas y librerías que fue desarrollado a principios de los 90s, por el Laboratorio Nacional Oak Ridge, en Estados Unidos. Se encuentra disponible para sistemas Linux o Windows NT/XP, y para los lenguajes C, C++ y Fortran.

Está compuesto de dos partes: un proceso demonio denominado pvmd3 y librerías basadas en rutinas. La interfaz de la librería PVM contiene las primitivas necesarias para la cooperación entre las tareas de una aplicación. Define las rutinas para paso de mensajes, sincronización de tareas y creación de procesos.

Las implementaciones de PVM para los lenguajes de programación C y C++ utilizan una interfaz con funciones basadas en las convenciones del lenguaje C, que permiten

acceder a sus diferentes librerías, mientras en el lenguaje Fortran, la funcionalidad de PVM se implementa como subrutinas en lugar de funciones.

2.7.3 MPI

MPI no es un lenguaje de programación, es un conjunto de funciones y macros que conforman una librería estándar de C y C++, y subrutinas en Fortran. La primera versión del estándar MPI aparece en Mayo de 1994 y a mediados de 1995, aparece la Versión 1.1, en la cual se agregaron algunas aclaraciones y refinamientos. Estas Versiones fueron diseñadas para los lenguajes C y Fortran 77. En Marzo de 1995, se extendió la versión original con la creación del estándar MPI Versión 2. La cual incluye la implementación para C++ y Fortran 90.

MPI ofrece un API, junto con especificaciones de sintaxis y semántica que explican como sus funcionalidades deben añadirse en cada implementación que se realice (tal como almacenamiento de mensajes o requerimientos para entrega de mensajes). MPI incluye operaciones punto a punto y colectivas, todas destinadas a un grupo específico de procesos. También realiza la conversión de datos heterogéneos como parte transparente de sus servicios, por medio de la definición de tipos de datos específicos para todas las operaciones de comunicación. Se pueden tener tipos de datos definidos por el usuario o primitivos.

Esta herramienta se describe con más detalle en el anexo D, ya que fue el protocolo base en la implementación del algoritmo desarrollado dentro de este trabajo.

Capítulo 3
DISEÑO DE ALGORITMOS PARALELOS Y DISTRIBUIDOS

3.1 INTRODUCCION

Un algoritmo paralelo básicamente, en oposición a los algoritmos clásicos o secuenciales, es un algoritmo que puede ser ejecutado por partes en el mismo instante de tiempo por varias unidades de procesamiento, para finalmente unir todas las partes y obtener el resultado correcto. Los algoritmos paralelos son importantes porque en general es más rápido tratar grandes tareas de computación mediante la paralelización que mediante técnicas secuenciales. Esta es la forma en que se trabaja en el desarrollo de los procesadores modernos, ya que es más difícil incrementar la capacidad de procesamiento con un único procesador que aumentar su capacidad de cómputo mediante la inclusión de unidades en paralelo, logrando así la ejecución de varios flujos de instrucciones dentro del procesador. Sin embargo, hay que tener cuidado con la excesiva paralelización de los algoritmos ya que cada algoritmo paralelo tiene una parte secuencial y debido a esto, los algoritmos paralelos pueden llegar a un punto de saturación, esta conclusión puede ser deducida de la ley de Amdahl[7] que en términos simples expone que es el algoritmo y no el número de procesadores el que define la mejora en la velocidad de ejecución. Debido a esto, a partir de cierto nivel de paralelismo, añadir más unidades de procesamiento puede sólo incrementar el

[7] Usada para averiguar la mejora máxima de un sistema cuando solo una parte de este es mejorado. Véase sección 3.4.2

coste y la disipación de calor.

3.2 PARADIGMAS DE LA COMPUTACION PARALELA

La computación paralela proporciona alternativas para el mejoramiento del rendimiento, se aplica a todos los niveles del diseño de un sistema de cómputo y aunque es una perspectiva desafiante, es necesaria tanto para el cómputo científico especializado como para el cómputo de propósito general. Desarrollar un algoritmo paralelo implica tendencias, paradigmas o métodos de programación de acuerdo al tipo de problema y maquina paralela disponible o hacia la que va dirigida la aplicación.

Desde el punto de vista de la arquitectura de la maquina paralela, los algoritmos paralelos deben buscar la forma de optimizar la comunicación entre las diferentes unidades de procesamiento. Esto se puede conseguir mediante la aplicación de dos paradigmas de programación y diseño de procesadores: memoria compartida y paso de mensajes.

La técnica memoria compartida necesita del uso de cerrojos en los datos para impedir que sean modificados simultáneamente por dos procesadores, por lo que se produce un coste extra en ciclos de CPU desperdiciados y ciclos de bus. La técnica paso de mensajes usa canales y mensajes pero esta comunicación añade un coste al bus, memoria adicional para las colas y los mensajes, además de la latencia en el mensaje. Los diseñadores de procesadores paralelos usan buses especiales para que el coste de la comunicación sea pequeño pero siendo el algoritmo paralelo el que decide el volumen del tráfico.

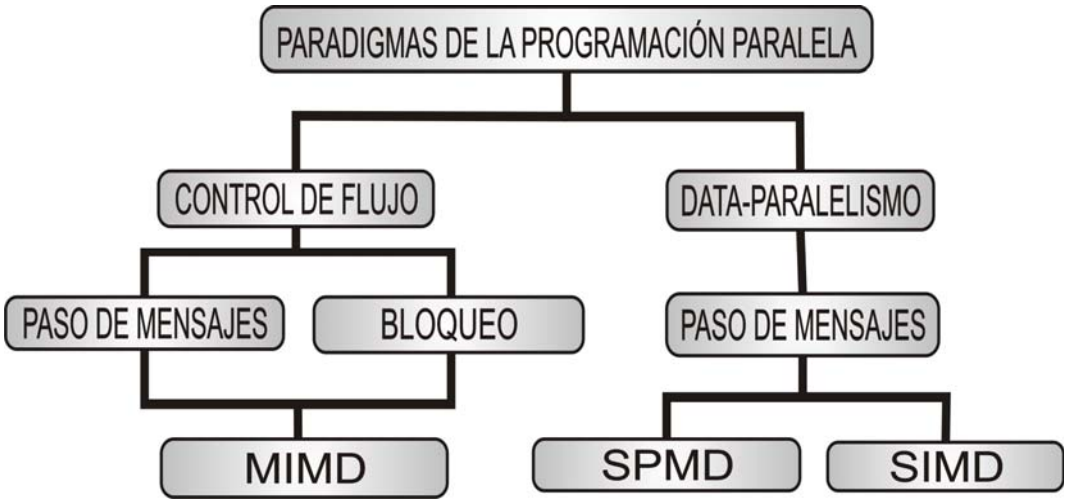

Figura 3.1. Paradigmas de la programación paralela. (Fuente: [MANT25])

Por otro lado, sin tener en cuenta la arquitectura del sistema, un algoritmo paralelo tiene que coordinar armoniosamente dos o más procesos para asegurar la exactitud así como también una alta velocidad en su ejecución. Este es el desafió de la programación paralela. Exactamente, cuanto paralelismo es controlado y usado para ganar alto desempeño, es ampliamente determinado por el paradigma particular usado por el diseñador del algoritmo. De acuerdo a la forma de generar y sincronizar apropiadamente las partes de un determinado problema, un algoritmo paralelo puede considerar cualquiera de los dos estilos generales presentados a continuación:

- Flujo de control (control flow): es logrado aplicando diferentes operaciones a distintos elementos del conjunto de datos simultáneamente; el flujo de datos entre estos procesos puede ser arbitrariamente complejo.

- Paralelismo de datos (data parallel): es el uso de múltiples unidades funcionales para aplicar las mismas operaciones simultáneamente a elementos del mismo conjunto de datos.

La figura 3.1 presenta un esquema que resume las ideas anteriormente expuestas, el

diseño de un determinado algoritmo paralelo debe considerar la combinación de todos estos enfoques.

3.3 METODOLOGIA PARA LA CREACION DE ALGORITMOS PARALELOS [GAR23] [MANT25] [HOE23]

El diseño de algoritmos paralelos no es tarea fácil y es un proceso altamente creativo. La transformación de un determinado problema en un algoritmo requiere especial cuidado en lo referente a concurrencia, escalabilidad, localidad y modularidad. La adopción de un enfoque metódico que maximice el rango de opciones y provea de mecanismos para evaluar alternativas, puede reducir el costo que implica tomar caminos inadecuados. La metodología aquí expuesta se compone de una serie de fases o etapas presentadas secuencialmente en la figura 3.2 pero que en la práctica no lo son, dependerá del diseñador combinar o retomar cada etapa para lograr un algoritmo óptimo. En las fases iniciales se debe explorar la complejidad del problema dejando de lado los aspectos independientes de la arquitectura de la maquina, los cuales serán considerados en las fases finales. La metodología PCAM (Particion-Communicate-Agglomerate-Map) involucra cuatro etapas, sin embargo debido a la importancia de analizar el problema en este libro se consideran cinco, las cuales se describen a continuación:

- ***Análisis del problema*:** Observar el problema para deducir si una solución paralela es viable o no.

- ***Partición*:** El computo y los datos sobre los cuales se opera se descomponen en tareas. Se ignoran aspectos sobre el numero de procesadores de la maquina a usar y se centra la atención en explorar oportunidades de paralelismo.

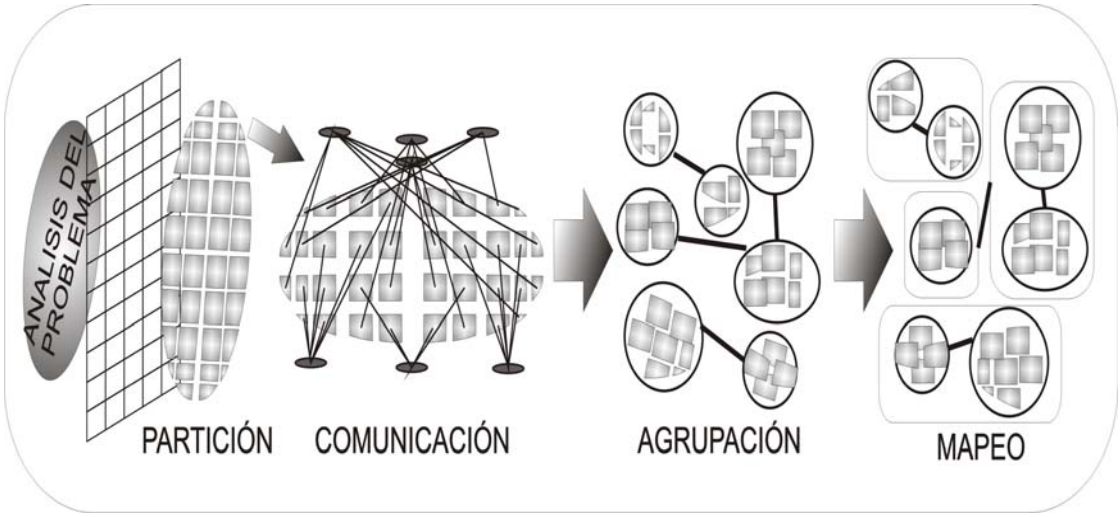

Figura 3.2 Pasos Básicos en la creación de un algoritmo paralelo. (Fuente: Autor)

- ***Comunicación:*** Se determina la comunicación requerida para coordinar las tareas, definiendo estructuras y algoritmos de comunicación.

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

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

3.3.1 ANALISIS DEL PROBLEMA

En esta etapa se analiza el problema y se observa si una solución paralela es buena o no, ya que en ocasiones, la mejor solución secuencial es la peor solución paralela y viceversa. Además siempre se debe explorar a fondo si la mejor solución secuencial no sirve.

En este análisis se identifican las variables y relaciones entre ellas, así mismo se determinan ciclos y causalidades. También se analiza la exigencia computacional: capacidad de almacenamiento, intensidad de procesamiento, características de datos y procesos y tiempos de respuesta exigidos.

Algunos autores afirman que el 80% de la eficiencia de un programa está dada por la correcta comprensión y su "buen" algoritmo (independiente del lenguaje de programación utilizado).

3.3.2 PARTICION

El objetivo de esta fase es detectar oportunidades de ejecución paralela y se trata de subdividir el problema en tareas pequeñas en orden de rendimiento buscando la granularidad[8] fina. Las tareas deben ser las suficientes para mantener ocupados los procesos todo el tiempo, sin embargo, no deben ser demasiadas, ya que el número de tareas disponibles a la vez es el límite superior en aceleración que se puede lograr. Evaluaciones futuras podrán llevar a aglomerar tareas y descartar ciertas posibilidades de paralelismo. Se debe considerar que las tareas pueden estar disponibles dinámicamente y que el numero de tareas disponibles puede variar con el tiempo. Una buena partición divide tanto los cómputos como los datos. Hay dos

[8] Un grano es una medida del trabajo computacional a realizar.

formas de proceder con la descomposición:

- ***Descomposición del dominio***: Primero se centra en los datos asociados con el problema, se determina una descomposición apropiada de los mismos y finalmente asocia computación a cada porción de los datos. Como resultado se obtienen tareas, donde cada tarea es un conjunto disyunto de datos con un conjunto de operaciones asociadas. Se deben tener en cuenta tanto los datos de entrada y salida como los datos intermedios que se generen, por lo tanto se pueden dar diferentes descomposiciones. Hay que centrarse primero sobre la estructura más grande o la accedida más frecuentemente. También es de resaltar la existencia de diferentes fases de computo lo cual puede generar diferentes descomposiciones de la misma o de diferentes estructuras de datos, por lo tanto se debe tratar cada fase separadamente y luego determinar como encajar todas juntas. En la figura 3.3 se presenta un ejemplo de descomposición de dominio de una matriz tridimensional, los cálculos son efectuados en cada punto de la matriz, cada tarea mantiene el estado de varios valores asociados con sus respectivos puntos y es responsable de los cálculos requeridos para actualizar el estado.

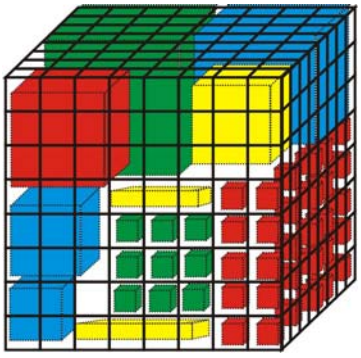

Figura 3.3. Descomposición de dominio de un problema de matriz 3D.

(Fuente: Autor)

- ***Descomposición funcional***: Inicialmente se descomponen los cómputos a realizar y luego se ocupa de los datos. El procesamiento se divide en tareas disyuntas, después se procede a examinar los datos que serán utilizados por esas tareas, si los datos son disyuntos el particionamiento es completo, sino se requiere replicar los datos o comunicarlos entre tareas diferentes lo cual determina si la técnica es inadecuada. Por lo general esto requiere una manera diferente de pensar, pero el centrarse en los cálculos podría revelar estructuración en un problema generando oportunidades de optimización. Como ejemplo se presenta en la figura 3.4 la descomposición jerárquica de un modelo climático en el cual lo más importante es modelar cada factor que afecta el clima y su interacción con los demás factores.

Estas técnicas son complementarias y pueden ser aplicadas a diferentes componentes de un problema e inclusive al mismo problema para obtener algoritmos paralelos alternativos.

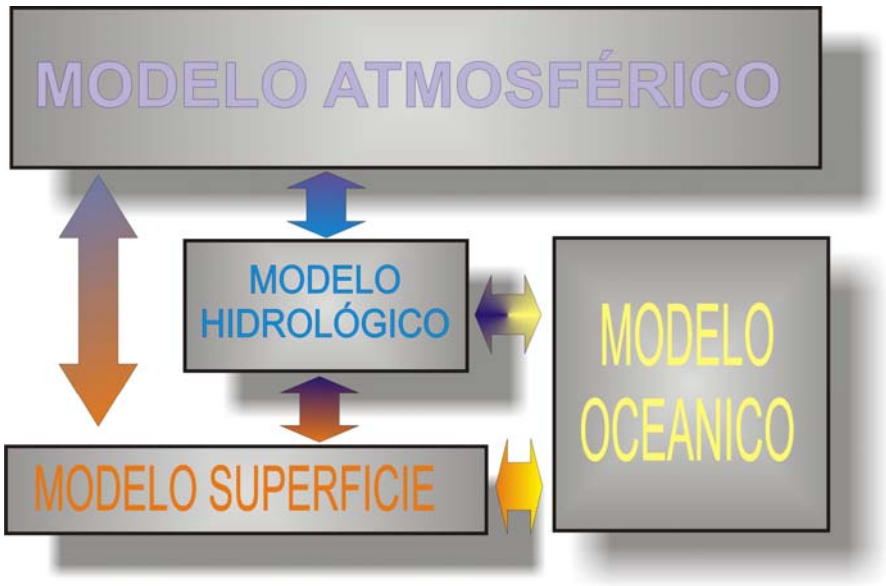

Figura 3.4 Descomposición funcional de un modelo climático (Fuente: [MANT25])

Al particionar se deben tener en cuenta los siguientes aspectos:

- ✓ El número de tareas debe ser por lo menos un orden de magnitud superior al número de procesadores disponibles para tener flexibilidad en las etapas siguientes.

- ✓ Hay que evitar cómputos y almacenamientos redundantes, de lo contrario el algoritmo puede no ser extensible a problemas más grandes

- ✓ Se debe tratar que las tareas sean de tamaños equivalentes, ya que esto facilita el balanceo de la carga de los procesadores

- ✓ El número de tareas debe ser proporcional al tamaño del problema. De esta forma el algoritmo será capaz de resolver problemas más grandes cuando se tenga mayor disponibilidad de procesadores. En otras palabras, se debe tratar que el algoritmo sea escalable, que el grado de paralelismo aumente al menos con el tamaño del problema.

- ✓ Considerar alternativas de paralelismo ya que puede flexibilizar etapas subsecuentes.

Si se obtienen respuestas negativas se debe replantear la descomposición e incluso revisar la especificación del problema.

3.3.3 COMUNICACIÓN

En esta etapa es fundamental tener en cuenta la arquitectura disponible, el modelo y el lenguaje de programación. Las tareas definidas en la parte anterior, en general, pueden correr concurrentemente pero no independientemente. Datos deben ser compartidos o transferidos entre tareas por lo tanto en esta fase se debe atacar la

reducción de costos de comunicación y sincronización, para lo cual se tienen como objetivos: el promover la localidad de referencia de datos, la organización de tareas adecuadas para evitar tiempos latentes y la reducción del trabajo adicional para controlar el paralelismo.

La comunicación requerida por un algoritmo puede ser definida en dos fases. Primero se definen los canales de comunicación que conectan directa o indirectamente las tareas que requieren datos (clientes) con las que los proporcionan (productores). Segundo, se especifica la información o mensajes que deben ser enviados y recibidos en estos canales. Dependiendo del tipo de maquina donde se implementara el algoritmo, memoria distribuida o memoria compartida, la forma de atacar la comunicación entre tareas varia.

En ambientes de memoria distribuida, cada tarea tiene una identificación única y las tareas interactúan enviando y recibiendo mensajes hacia y desde tareas especificas. Las librerías más conocidas para implementar el pase de mensajes en ambientes de memoria distribuida son: MPI (Message Passing Interface) y PVM (Parallel Virtual Machine).

En ambientes de memoria compartida no existe la noción de pertenencia y el envió de los datos no se da como tal. Todas las tareas comparten la misma memoria. Semáforos, semáforos binarios, barreras y otros mecanismos de sincronización son usados para controlar el acceso a la memoria compartida y coordinar las tareas. Por ejemplo, un semáforo es un tipo de variable que puede tomar valores enteros mayores o iguales a cero. El valor del semáforo es cambiado mediante dos operaciones: UP (incrementa) y DOWN (decremento). La operación DOWN chequea si el valor es mayor que cero, y de ser así, decrementa el valor del semáforo en uno y continua, cuando el valor sea cero el proceso es bloqueado. Chequear el valor del semáforo, cambiarlo y posiblemente ser bloqueado es hecho en una acción elemental única e indivisible. Se garantiza que una vez que una operación haya comenzado, ningún otro sujeto puede acceder el semáforo hasta que la acción haya sido completada. Esta

atomicidad es esencial para lograr la sincronización entre procesos. La operación UP incrementa en uno el semáforo y en caso de haber procesos bloqueados por no haber podido completar un DOWN en ese semáforo, uno de ellos es seleccionado por el sistema y se le permite completar el DOWN. Nunca un proceso es bloqueado efectuando una operación UP.

Esta etapa implica la identificación de requerimientos de comunicación y la introducción de estructuras de canales (Topología) y operaciones de comunicación de acuerdo a esos requerimientos. Existen diversos tipos de comunicación, sin embargo se resaltaran dos en particular por su generalidad, estos son: la comunicación local y la comunicación global.

Figura 3.5 Esquema comunicación método de diferencias finitas de Jacobi (Fuente: [MANT25])

Comunicación Local: Se emplea cuando una operación requiere datos de un pequeño numero de tareas; Primero se definen los canales que enlacen los productores con los consumidores y luego se introduce las operaciones de envió y recepción adecuadas. Como ejemplo se presenta en la ecuación 3.1 la formula iterativa del método de

diferencias finitas de Jacobi el cual se esquematiza en la figura 3.5; nótese que cada nuevo dato depende del dato de sus vecinos horizontales y verticales.

$$X_{i,j}^{(t+1)} = \frac{\left(4X_{i,j}^{(t+1)} + X_{i-1,j}^{(t+1)} + X_{i+1,j}^{(t+1)} + X_{i,j-1}^{(t+1)} + X_{i,j-1}^{(t+1)}\right)}{8} \quad (3.1)$$

Comunicación Global: En un operación de comunicación global participan muchas tareas y es ineficiente solo identificar pares productor/consumidor, ya que por lo general se enfoca el problema centralizadamente y sugiere poca concurrencia. Como ejemplo se tiene la sumatoria de N números en la cual todos los datos se centran en el resultado lo cual no permite ver la concurrencia del problema, sin embargo si se distribuye la computación y la comunicación en N-1 pasos se podrá tener una ejecución con un cauce concurrente para múltiples operaciones de suma, con lo cual, aplicando la técnica de divide y vencerás, se descompone recursivamente modificando el orden de las operaciones para obtener una estructura de comunicación regular y local en logaritmo de N pasos. En la figura 3.6 se presenta la idea principal de este problema y su solución altamente paralela.

En resumen, en esta etapa es importante tener en cuenta los siguientes aspectos:

- ✓ Todas las tareas deben efectuar aproximadamente el mismo número de operaciones de comunicación. Si esto no se da, es muy probable que el algoritmo no sea extensible a problemas mayores ya que habrá cuellos de botella

- ✓ La comunicación entre tareas debe ser tan pequeña como sea posible.

- ✓ Las operaciones de comunicación deben poder proceder concurrentemente.

- ✓ Los cómputos de diferentes tareas deben poder proceder concurrentemente.

Si estos enunciados no se cumplen posiblemente el algoritmo será ineficiente y no escalable. Se recomienda en todo caso usar una descomposición recursiva para descubrir concurrencia, en caso de fallar habrá que reordenar las operaciones de comunicación y computación e incluso replantear las especificaciones del problema.

3.3.4 AGRUPACION

En esta etapa se va de lo abstracto a lo concreto y se revisa el algoritmo obtenido tratando de producir un algoritmo que sea ejecutado eficientemente sobre cierta clase de computadores. En particular se considera si es útil agrupar tareas y si vale la pena replicar datos y/o cómputos.

En la fase de partición se trató de establecer el mayor número posible de tareas con la intención de explorar al máximo las oportunidades de paralelismo. Esto no necesariamente produce un algoritmo eficiente ya que el costo de comunicación puede ser significativo. En la mayoría de los computadores paralelos, la comunicación se realiza mediante el paso de mensajes y frecuentemente hay que parar los cómputos para enviar o recibir mensajes. Mediante la agrupación de tareas de puede reducir la cantidad de datos a enviar y así reducir el numero de mensajes y el costo de comunicación. En la figura 3.6 se muestran dos enfoques para el método de diferencias finitas en dos dimensiones y el efecto superficie-volumen, el cual se refiere a que los requisitos de comunicación son proporcionales a la superficie del subdominio mientras los requisitos de cómputo son proporcionales al volumen del subdominio. Para este ejemplo, en el caso A se tienen 64 (8x8) tareas cada una con 4 mensajes por tarea para un total de 256 comunicaciones con 256 valores a transferir. En el caso B se obtiene 4 (2x2) tareas con 4 mensajes por tarea con lo cual se necesitan solo 16 comunicaciones de 64 valores a transferir. En consecuencia se observa que las descomposiciones de dominio con mayores dimensiones son generalmente las más eficientes.

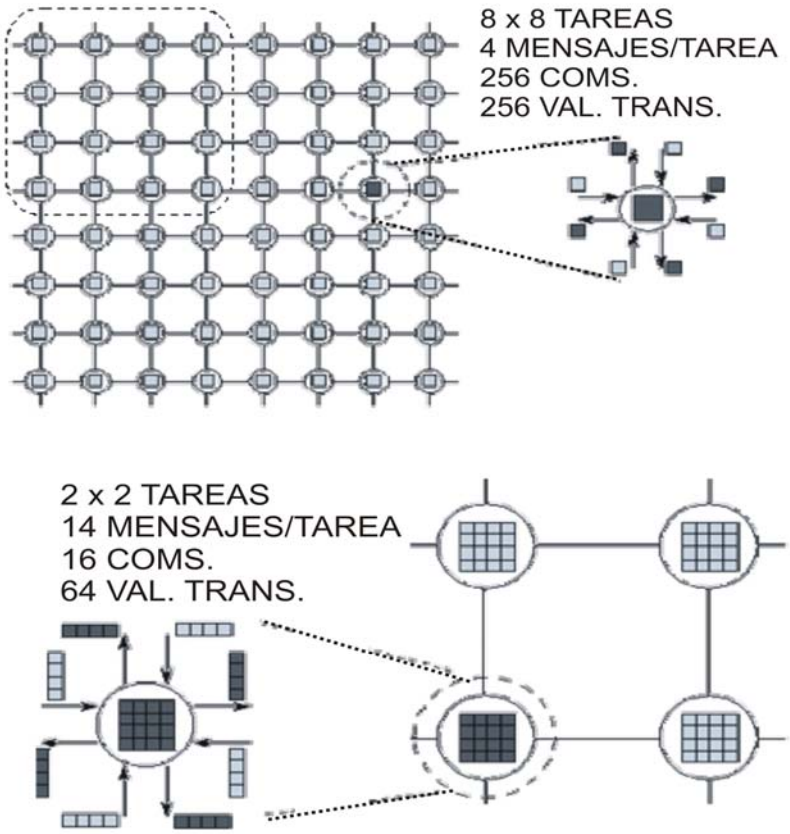

Figura 3.6 Método de diferencias finitas: dos maneras de descomposición. (Fuente: [LLO22])

Los objetivos generales de esta etapa se pueden resumir en tres: reducir costos de comunicación, flexibilidad (escalabilidad y asignación) y reducir costos en ingeniería del software.

- ✓ **Reducir costos de comunicación**: La comunicación no solo depende de la cantidad de información enviada. Cada comunicación tiene un costo fijo de arranque. Reduciendo el número de mensajes, a pesar de que se envié la misma cantidad de información, puede ser útil. Así mismo se puede intentar replicar

cómputos y/o datos para reducir el costo de creación de tareas y el costo de cambio de contexto (context switch) en caso de que se asignen varias tareas a un mismo procesador.

En caso de tener distintas tareas corriendo en diferentes computadores con memorias privadas, se debe tratar de que la granularidad sea gruesa: exista una cantidad de cómputo significativa antes de tener necesidades de comunicación. Se puede tener granularidad media si la aplicación correrá sobre una maquina de memoria compartida. En estas maquinas el costo de comunicación es menor que en las anteriores siempre y cuando en número de tareas y procesadores se mantenga dentro de cierto rango. A medida que la granularidad decrece y el número de procesadores se incrementa, se intensifica las necesidades de altas velocidades de comunicación entre los nodos. Esto hace que los sistemas de grano fino por lo general requieran maquinas de propósito especifico.

Se pueden usar maquinas de memoria compartida si el numero de tareas es reducido, pero por lo general se requieren maquinas masivamente paralelas conectadas mediante una red de alta velocidad (arreglos de procesadores o maquinas vectoriales).

- ✓ **Preservando flexibilidad**: En ningún momento la portabilidad y escalabilidad del algoritmo deben comprometerse, se debe tratar de utilizar un número de tareas variable pero siempre mayor al número de procesadores, ya que facilita el ajuste a un computador concreto (solapamiento computación/comunicación) y permite el equilibrado de carga al tener más tareas que procesadores. Sin embargo la flexibilidad no implica un gran número de tareas, se debe tratar de no limitar el número de tareas sin una necesidad real, lo ideal es parametrizar la granularidad.

- ✓ **Reducción de costo en ingeniería del software**: Este es un aspecto importante cuando se paraleliza un código secuencial existente. Se deben escoger estrategias de descomposición que minimicen los cambios del código secuencial sin perdida

excesiva de eficiencia. Esto mejora la reutilización de código y minimiza los costos de desarrollo.

En resumen los puntos resaltantes que se deben considerar en esta etapa son:

- Chequear si la agrupación redujo los costos de comunicación por el incremento de la localidad.

- Si se han replicado cómputos y/o datos, se debe verificar que los beneficios son superiores a los costos.

- Se deben verificar que las tareas resultantes tengan costos de cómputo y comunicación similares.

- Hay que revisar si el número de tareas es extensible con el tamaño del problema.

- Si el agrupamiento ha reducido las oportunidades de ejecución concurrente, se debe verificar que aun hay suficiente concurrencia y considerar diseños alternativos.

- Analizar si es posible reducir aun más el número de tareas sin introducir desbalances de cargas, incrementar los costos en la ingeniería de software o reducir la escalabilidad.

3.3.5 MAPEO

En esta última etapa se determina en qué procesador se ejecutará cada tarea. Por el momento no hay mecanismo de asignación de tareas para maquinas distribuidas. Esto continúa siendo un problema difícil que debe ser atacado explícitamente a la hora de diseñar algoritmos paralelos. El objetivo aquí no es otro sino reducir el tiempo de

ejecución total; para tal fin, se pueden plantear dos estrategias: ejecutar tareas concurrentemente sobre diferentes procesadores (concurrencia) y localizar tareas con interacción frecuente sobre el mismo computador (localidad). Aquí la asignación de tareas debe mezclarse con una estrategia para ordenar la ejecución lo que se resume en la planificación de tareas.

La asignación de tareas puede ser estática o dinámica. En la asignación estática, las tareas son asignadas a un procesador al comienzo de la ejecución de un algoritmo paralelo y corren ahí hasta el final, lo cual en ciertos casos puede resultar en un tiempo de ejecución menor respecto a asignaciones dinámicas y también puede reducir el costo de creación de procesos, sincronización y terminación.

En la asignación dinámica se hacen cambios en la distribución de las tareas entre procesadores en tiempo de ejecución, o sea, hay migración de tareas en tiempo de ejecución. Esto con el fin de balancear la carga del sistema y reducir el tiempo de ejecución. Sin embargo, el costo de balanceo puede ser significativo y por ende incrementar el tiempo de ejecución. Entre los algoritmos de balanceo de carga están los siguientes:

- Balanceo centralizado: un nodo ejecuta el algoritmo y mantiene el estado global del sistema. Este método no es extensible a problemas más grandes ya que el nodo encargado del balanceo se convierte en un cuello de botella.

- Balanceo completamente distribuido: cada procesador mantiene su propia visión del sistema intercambiando información con sus vecinos y así hacer cambios locales. El costo de balanceo se reduce pero no es óptimo debido a que solo se posee información parcial.

- Balanceo semi-distribuido: divide los procesos en regiones, cada una con un algoritmo centralizado local. Otro algoritmo balancea la carga entre las regiones.

En la figura 3.7 se presenta un ejemplo de mapeo con balanceo de carga dinámico. Allí los procesadores organizados en malla periódicamente comparan carga con los procesadores vecinos transfiriendo si la diferencia de carga es muy alta.

El balanceo puede ser iniciado por envió o recibimiento. Si es balanceo iniciado por envió, un procesador con mucha carga envía trabajos a otros. Si es balanceo iniciado por recibimiento, un procesador con poca carga solicita trabajo de otros. Si la carga por procesador es baja o mediana, es mejor el balanceo iniciado por envió. Si la carga es alta se debe utilizar balanceo por recibimiento. De lo contrario, en ambos casos, se puede producir una fuerte migración innecesaria de tareas.

Entre los puntos que hay que revisar en esta etapa encontramos:

- ¿Se ha considerado algoritmos con un número estático de tareas y algoritmos de creación dinámica de tareas?

- Si se usan algoritmos centralizados de balanceo hay que asegurarse que no sea un cuello de botella

- hay que evaluar los costos de las diferentes alternativas de balanceo dinámico, en caso de que se usen, y que su costo no sea mayor que sus beneficios.

- Hay que tener en cuenta que aunque el programador especifique algunos aspectos del mapeo. En el momento de la ejecución el sistema operativo los puede ignorar.

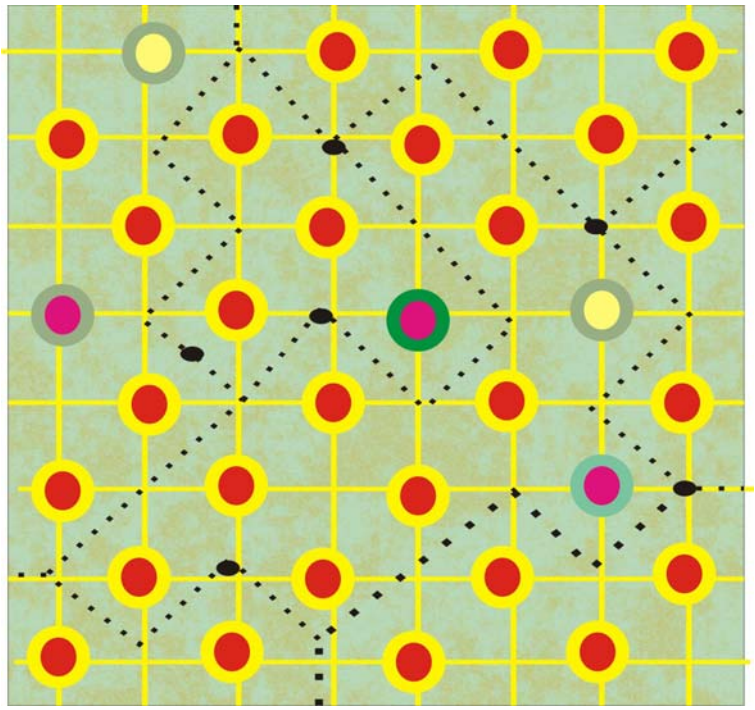

Figura 3.7. Procesadores organizados en malla con algoritmo local de equilibrio de carga. (Fuente: Autor)

3.4 ANALISIS DE RENDIMIENTO

El rendimiento de un algoritmo paralelo es una cuestión compleja que depende de muchos factores como el tiempo de ejecución, escalabilidad, generación, almacenamiento y comunicación de datos. Hay diversidad de métricas para el rendimiento entre las que se pueden mencionar: el tiempo de ejecución, requerimientos de memoria, throughput, latencia, rango de entrada/salida, uso de red, costos de implementación, requerimientos de hardware, portabilidad, escalabilidad, etc.

Dependiendo de la aplicación una métrica toma más importancia que otra, como ejemplo se tienen los siguientes casos:

- Predicción de tiempo: las métricas más importantes son el tiempo de ejecución, los costos de hardware, los costos de implementación, la exactitud, la confiabilidad y la escalabilidad.

- Bases de datos paralelas: se destacan el throughput, los costos de implementación y mantenimiento.

- Sistemas de visión: por lo general el tiempo de imágenes procesadas por segundo (throughput) y el tiempo de espera (latencia) son los factores a evaluar.

- Aplicaciones bancarias: se resalta el rango de tiempo de ejecución contra el costo.

Aquí se definirán las más comúnmente usadas para el análisis de un algoritmo paralelo, estas son la aceleración y la eficiencia.

3.4.1 ACELERACION Y EFICIENCIA [KAR24]

La definición formal de aceleración de un algoritmo paralelo se denota:

Sea P un problema computacional dado y n su tamaño de entrada. Denote la complejidad secuencial de P como $T^*(n)$. Sea A un algoritmo paralelo que resuelve P en $T_p(n)$ sobre una computadora paralela con p procesadores. Entonces, la aceleración obtenida por A es definida como:

$$S_p(n) = \frac{T^*_n(n)}{T_S(n)} \quad (3.2)$$

Idealmente, nos gustaría desarrollar algoritmos que obtuvieran $S_p(n)$ aproximadamente igual a p. En realidad, hay muchos factores que restringen este

incremento lineal como los retardos introducidos por comunicación y gastos generales por la sincronización de las actividades de varios procesadores y el sistema de control.

La aceleración es una medida relativa al mejor algoritmo secuencial posible. En la práctica, es común remplazar $T^*(n)$ por el tiempo limite del mejor algoritmo secuencial conocido en los que la complejidad del problema es desconocida.

Otra medida de rendimiento de un algoritmo paralelo (A) es la eficiencia $E_p(n)$, definida como:

$$E_p(n) = \frac{T_1(n)}{P \bullet T_p(n)} \qquad (3.3)$$

Un valor de E(n) aproximadamente igual a uno, para algún p, indica que el algoritmo A se ejecuta aproximadamente p veces mas rápido usando p procesadores que usando un solo procesador.

Existe un limitante sobre el tiempo de ejecución, denotado T_∞, más allá del cual el algoritmo no puede ser más rápido, sin importar el número de procesadores. Así, $T_P \geq T_\infty$, para cualquier valor de p, de este modo la eficiencia satisface

$$E_p(n) \geq \frac{T_1(n)}{P \bullet T_\infty(n)} \qquad (3.4)$$

Por lo tanto la eficiencia de un algoritmo se atenúa rápidamente conforme p crece a razón de T_1/T_∞.

3.4.2 LA LEY DE AMDAHL. [HOE23]

Sea B la fracción de operaciones en un cómputo que debe ser efectuado secuencialmente. El mejor programa paralelo usando p procesadores obtendrá un tiempo de ejecución:

$$T_p = B \cdot T_1 + \frac{T_1(1-B)}{p} \qquad (3.5)$$

Así, la aceleración está superiormente limitada por:

$$S_p = \frac{p}{B \cdot p + (1-B)} \quad (3.6)$$

Se observa que si B tiende a cero entonces la aceleración es p. excepto casos muy particulares, este límite nunca es obtenible debido a las siguientes razones:

- Inevitablemente parte del algoritmo es secuencial,
- Hay conflictos de datos y memoria,
- La comunicación y sincronización de procesos consume tiempo,
- Y es muy difícil lograr un balanceo de carga perfecto entre los procesadores.

La Ley de Amdahl se puede interpretar de manera más técnica, pero en términos simples, significa que es el algoritmo el que decide la mejora de velocidad, no el número de procesadores. Finalmente se llega a un momento que no se puede paralelizar más el algoritmo.

3.4.3 COSTO DE COMUNICACIÓN. [KAR24]

Para analizar el costo de comunicación en el algoritmo paralelo se desprecia el tiempo de comunicación entre procesadores y se asume que el código es 100% paralelizable, esto implica que la aceleración es

$$S = p. \qquad (3.7)$$

Ahora; si se considera el tiempo de comunicación o latencia T_C, la aceleración

decrecerá aproximadamente a:

$$S_p = \frac{T}{\frac{T}{p} + T_C} < p \qquad (3.8)$$

Y para que la aceleración no sea afectada por la latencia de comunicación se necesita que:

$$\frac{T}{p} >> T_C \rightarrow p << \frac{T}{T_C} \qquad (3.9)$$

Esto significa que a medida que se divide el problema en partes más y más pequeñas para poder correr el problema en más procesadores, llega un momento en que el costo de comunicación T_C se hace muy significativo y desacelera el cómputo.

Capítulo 4
ALGORITMO PARALELO DEL NODO SA

En este capitulo se describen los lineamientos básicos para el desarrollo del algoritmo paralelo del nodo SA propuesto dentro de este trabajo de grado; la metodología PCAM expuesta en la sección 3.3 es utilizada para dar un enfoque metódico al desarrollo del algoritmo, sin embargo el orden de ideas aquí expuesto es el resultado de un proceso de combinación de las diferentes etapas, retomando cada una a lo largo del desarrollo del algoritmo. Pese que el punto de partida del algoritmo paralelo resultante es el algoritmo secuencial propuesto por [DASIERRA23] el enfoque es diferente; [DASIERRA23] centra la atención en el potencial de membrana en el centro y la periferia desarrollando una simulación para estos dos puntos, el algoritmo aquí desarrollado obtiene el potencial de membrana para un punto x desde el centro del nodo SA hasta el final del músculo atrial pasando por la periferia del nodo SA, hecho por el cual se aplicaron modificaciones al algoritmo base para desarrollar un algoritmo secuencial que arrojara el potencial todo el nodo SA y el músculo atrial.

4.1 ANALISIS DEL MODELO DEL NODO SA COMPLETO

Como se expuso en la sección 1.3.4 el modelo del nodo SA completo posee una serie de características que derivan el conjunto total de ecuaciones involucradas para el cálculo de cada valor de potencial de membrana en el espacio-tiempo. Este conjunto de ecuaciones esta compuesto por:

- ✓ 2 Ecuaciones diferenciales parciales unidimensionales
- ✓ 15 Ecuaciones diferenciales ordinarias de primer orden

- ✓ 58 Ecuaciones algebraicas auxiliares

Las ecuaciones algebraicas son utilizadas en su mayoría para determinar alguna variable correspondiente con un punto en el espacio-tiempo, dadas sus características denotan una independencia en el espacio, es decir, para un determinado tiempo se puede calcular el valor de la variable asociada en un punto del espacio sin tener que conocerla para algún otro valor de la variable espacial (x) en ese instante.

Por otro lado, la existencia de ecuaciones diferenciales ordinarias de primer orden implica la dependencia temporal del modelo completo, ya que para poder calcular el valor de una variable a una distancia x_i del centro del nodo SA y para un instante de tiempo t_i dado se hace necesario conocer el valor de dicha variable en x_i y t_{i-1}, es decir, el instante anterior, este hecho limita la paralelización pues enmarca la simulación en un proceso secuencial para la variable temporal (t) haciendo difícil la paralelización. Sin embargo, dado que para un instante de tiempo se deben realizar una gran cantidad de cálculos, se debe centrar la atención en la variable espacial (x) si se quiere lograr una paralelización efectiva dentro de estas limitaciones.

La forma de las ecuaciones diferenciales parciales unidimensionales descritas en la tabla 1.10 genera una dependencia en la variable espacial (x) ya que incluye una derivada de segundo grado de la variable x, en la ecuación 4.1 se relaciona el método de Euler explicito con una aproximación de tres nodos, utilizado para resolver la ecuación diferencial parcial unidimensional del modelo, dicho método evidencia la dependencia con sus valores vecinos (x_{i-1} - x_{i+1}), es decir, para calcular el potencial de membrana en x_i es necesario conocer el potencial en x_{i+1} y x_{i-1} este hecho es de especial interés a la hora de paralelizar pues puede incurrir en una aumento en la comunicación entre procesos.

$$\left.\frac{\partial^2 V}{\partial X^2}\right|_{Xi} \approx \frac{V_{i+1} - 2 \cdot V_i + V_{i-1}}{\Delta X^2} \qquad (4.1)$$

Otro hecho a considerar en la paralelización del modelo es la existencia de varias corrientes iónicas cada una de las cuales agrupa una serie de ecuaciones únicas que deben ser resueltas para cada punto en el espacio-tiempo. El cómputo de las corrientes iónicas tiene como fin calcular una corriente iónica total que es utilizada en la ecuación diferencial parcial unidimensional para encontrar el valor del potencial de membrana. Cada corriente iónica es independiente, lo cual genera un posible punto de partida para la descomposición funcional del problema.

El autor del modelo aconseja utilizar un paso espacial (Δx) de 0.1 mm para el nodo SA y 0.32 mm para el músculo atrial los cuales son suficientemente pequeños para una solución estable y exacta. El hecho de que existan dos pasos diferentes para la variable espacial puede generar conflictos en la descomposición espacial ya que se debe especificar el paso espacial dentro de cada proceso ejecutado distinguiendo el músculo atrial del nodo SA.

Las pautas dadas anteriormente sugieren tanto una descomposición de dominio como una posible descomposición funcional, lo cual genera dos algoritmos diferentes. A continuación se expondrán las dos alternativas para luego acoplarlas en la descomposición efectuada para el algoritmo desarrollado.

4.2 ALGORITMO BASADO EN DESCOMPOSICION ESPACIAL

Debido a que el modelo incluye ecuaciones diferenciales en el dominio del tiempo se descarta una descomposición en este dominio y se centra la atención en la variable espacial (x). Esta descomposición del dominio espacial (x) es propuesta a partir del análisis de cada una de las ecuaciones involucradas para la solución del potencial de membrana. El modelo se basa en la variación de la capacidad eléctrica de las células a lo largo del nodo, explicando así las diferencias regionales dentro del nodo SA, esta variación esta asociada con la variable espacial x, lo cual es el punto de partida para esta descomposición.

La longitud total de la variable espacial es de 12.6 mm, considerando el punto x=0 como el centro del nodo, el punto x = 3 mm como la periferia y x>3 el músculo atrial. Para la longitud del nodo SA (3 mm) se considera un paso espacial (Δx) de 0.1 mm con lo cual se obtienen 30 valores de la variable x en este rango, mientras que para el músculo atrial (longitud 9.6 mm) se considera un paso espacial (Δx) de 0.32 mm generando también 30 valores de la variable x. En total se trabajan 60 (30+30) valores para la variable x, lo cual significa que en un instante t de simulación cada variable tendrá 60 valores que corresponden con su respectiva variación espacial, estos valores pueden ser calculados independientemente de los valores en otro punto del espacio (x_i) para todas la variables asociadas al modelo excepto para el potencial de membrana, el cual según la ecuación 4.1 necesita del valor de sus vecinos espaciales ($V_{(Xi-1)}$ $V_{(Xi+1)}$).

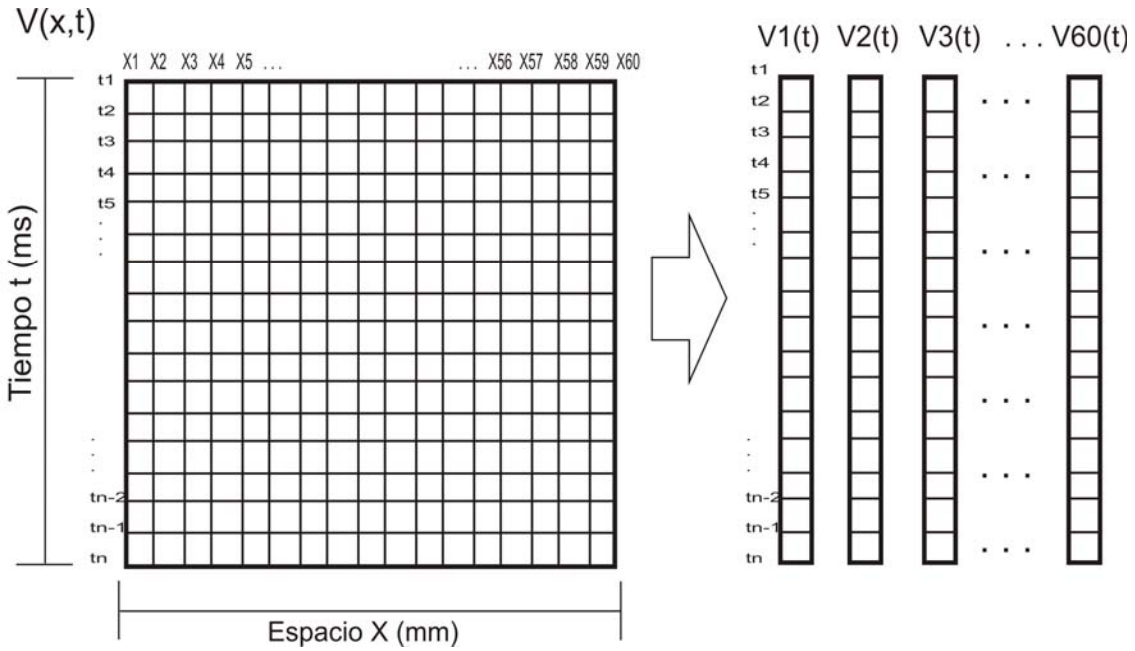

Figura 4.1. Descomposición Espacial de la variable V - Potencial de Membrana

(Fuente: Autor)

En la figura 4.1 se muestra la descomposición espacial para el potencial de membrana, allí la variable V representa el potencial de membrana en el modelo para cada punto de muestra (x_i,

t_i) y las variable V_1, V_2, V_3,…, V_{60} representan el potencial de membrana asociado con el punto x_i en todo el intervalo de tiempo, por lo tanto se pueden considerar 60 problemas cada uno con valores iniciales diferentes y asociado con una variable V_i (i=1,2,..,60) en todo el intervalo de tiempo. Así cada una de las 60 tareas calculará las corrientes iónicas en un determinado punto $x_{i,}$ determinando la corriente total en ese punto sin necesidad de conocer la corriente total en otro punto $x_{j.}$

$$\frac{V\langle t_{i+1}, x_i\rangle - V\langle t_i, x_i\rangle}{\Delta t} = \frac{-1}{C_m\langle x_i\rangle}\cdot I_T\langle t_i, x_i\rangle + D_s\cdot\left(\frac{V\langle t_i, x_{i+1}\rangle - 2\cdot V\langle t_i, x_i\rangle + V\langle t_i, x_{i-1}\rangle}{\Delta x^2}\right) \qquad (4.2)$$

Esta descomposición sugiere 60 tareas totalmente independientes, sin embargo la existencia de ecuaciones diferenciales parciales unidimensionales, generan una dependencia de las tareas con sus vecinas al necesitarse el valor del potencial de membrana de sus vecinos espaciales ($V_{(Xi-1)}$ $V_{(Xi+1)}$), para computar el valor de $V_{(Xi)}$ en el siguiente instante de tiempo. Esto se explica al observar la ecuación 4.2 la cual corresponde con la ecuación iterativa usada para resolver la ecuación diferencial parcial del potencial de membrana descrita en la ecuación 1.77.

La figura 4.2 presenta esta dependencia, observe que para poder calcular el potencial en cada instante t de la simulación es necesario que la tarea P_i conozca el valor del potencial de membrana de sus tareas vecinas P_{i-1} y P_{i+1} en el instante t-Δt. Este hecho sugiere una comunicación de todas las tareas con sus vecinas para recibir el valor necesario en cada paso de tiempo lo cual genera un costo alto en comunicaciones.

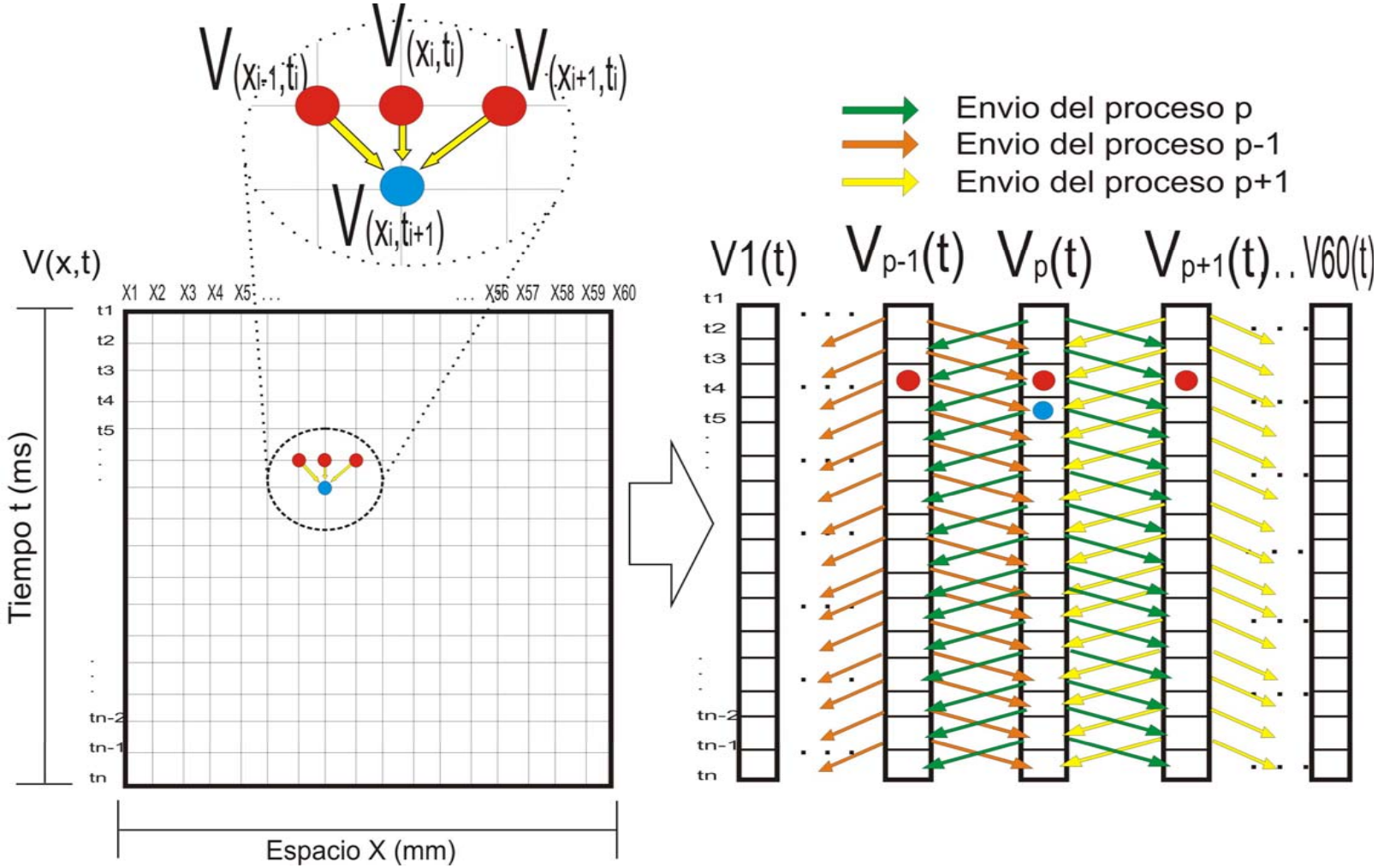

Figura 4.2 Dependencia Espacial del Potencial de Membrana (Fuente: Autor)

4.3 ALGORITMO BASADO EN DESCOMPOSICION SEGÚN CORRIENTES IONICAS

El modelo del nodo SA envuelve el cálculo de varias corrientes iónicas, cada una de las cuales implica la solución de ecuaciones diferenciales ordinarias asociadas a las variables de activación y desactivación, además del cálculo de ecuaciones algebraicas dentro del modelo de cada corriente. El cálculo de cada corriente es independiente de las demás, sin embargo para computar cada corriente en un instante dado se necesita el potencial de membrana en ese mismo instante, lo cual genera una dependencia en segundo grado, ya que para calcular el potencial de membrana se necesita la corriente iónica total, la cual se obtiene al sumar cada una de las corrientes iónicas.

Basado en lo anterior se plantea un algoritmo paralelo centralizado, en el cual exista un proceso maestro encargado de calcular el potencial de membrana y la corriente iónica total a partir del cálculo de las corrientes iónicas efectuado por cada uno de los procesos esclavos. Una vez calculado el potencial de membrana, el proceso maestro se encargará de actualizar a cada proceso esclavo con el nuevo valor del potencial de membrana para que cada proceso esclavo genere un nuevo valor de corriente el cual será recolectado por el maestro para obtener un nuevo valor de la corriente iónica total. Este esquema es presentado en la figura 4.3, allí cada variable es considerada como un vector que contiene todos los valores para la variable espacial (x) en un determinado instante de tiempo.

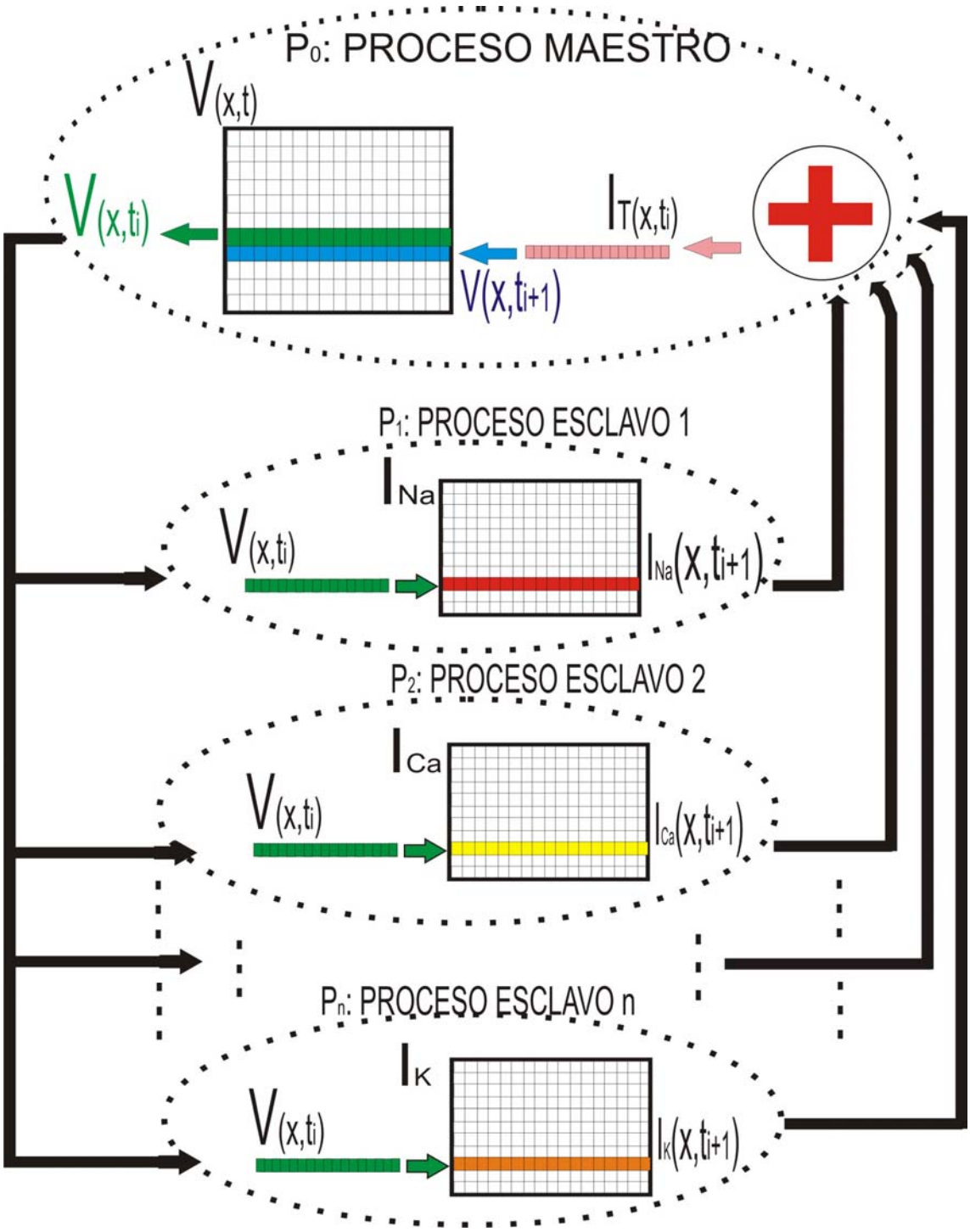

Figura 4.3 Descomposición según corrientes iónicas.

(Fuente: Autor)

4.4 ALGORITMO PARALELO PROPUESTO PARA EL NODO SA

Los algoritmos citados anteriormente tienen un único objetivo el cual corresponde a una frase: muy conocida en computación paralela: "Divide y Vencerás", sin embargo cada algoritmo posee un enfoque diferente que corresponde según la descomposición efectuada, el aprovechar las ventajas que ofrece cada solución planteada para fusionarlas en una única solución generó el algoritmo propuesto en este trabajo de grado. En la tabla 4.1 se presentan las principales ventajas y desventajas de cada enfoque, de allí se observa que las desventajas de uno son las ventajas del otro, y este hecho fue el que impulsó a fusionar los dos métodos en uno solo.

Descomposición Espacial (Dominio)	Descomposición Iónica (Funcional)
✓ Partición hasta en 60 procesos, correspondiente con cada punto $<x_i,t_i>$	✗ Partición limitada, máximo 8 corrientes iónicas independientes
✓ Balanceo de carga garantizado, cada proceso hace lo mismo para diferente punto en el espacio, calcular corriente y obtener el potencial.	✗ El balanceo de carga no esta garantizado ya que el cálculo de cada corriente varía según sus propias ecuaciones.
✗ Dependencia espacial en el cálculo del potencial de membrana. Cada proceso necesita datos de los procesos vecinos	✓ Independencia Total entre procesos esclavos. Cada proceso esclavo realiza cálculos independientemente.
✗ Ocupación de canal para envió de datos pequeños, latencia elevada.	✓ El canal se ocupa para enviar toda la corriente y no secciones de esta.

Tabla 4.1. Comparación de las posibles descomposiciones

El algoritmo propuesto consta de dos niveles o capas de paralelización jerárquica, en la primera capa se realiza una descomposición funcional según las corrientes iónicas en p procesos, luego en la segunda capa cada uno de los p procesos realiza una

descomposición de dominio espacial en q procesos esclavos. Con esto el particionamiento total del problema será de p*q procesos lo cual no limita la descomposición funcional y genera una mayor descomposición de dominio. En la figura 4.4 se presenta el diagrama jerárquico para el algoritmo propuesto, allí el proceso maestro es el encargado de computar el potencial de membrana para luego enviar este a los procesos de segundo nivel que se denotarán procesos jefes los cuales están relacionados con las corrientes iónicas, estos procesos dividen su trabajo en los procesos esclavos, los cuales computarán un determinado sector espacial de la corriente de su proceso jefe. Una vez calculado el respectivo sector los esclavos envía su parte al proceso jefe el cual organiza la información de los esclavos y la envía al maestro para que este obtenga la corriente iónica total y así pueda calcular el siguiente potencial de membrana, este proceso continua hasta un tiempo t con lo cual concluye la simulación.

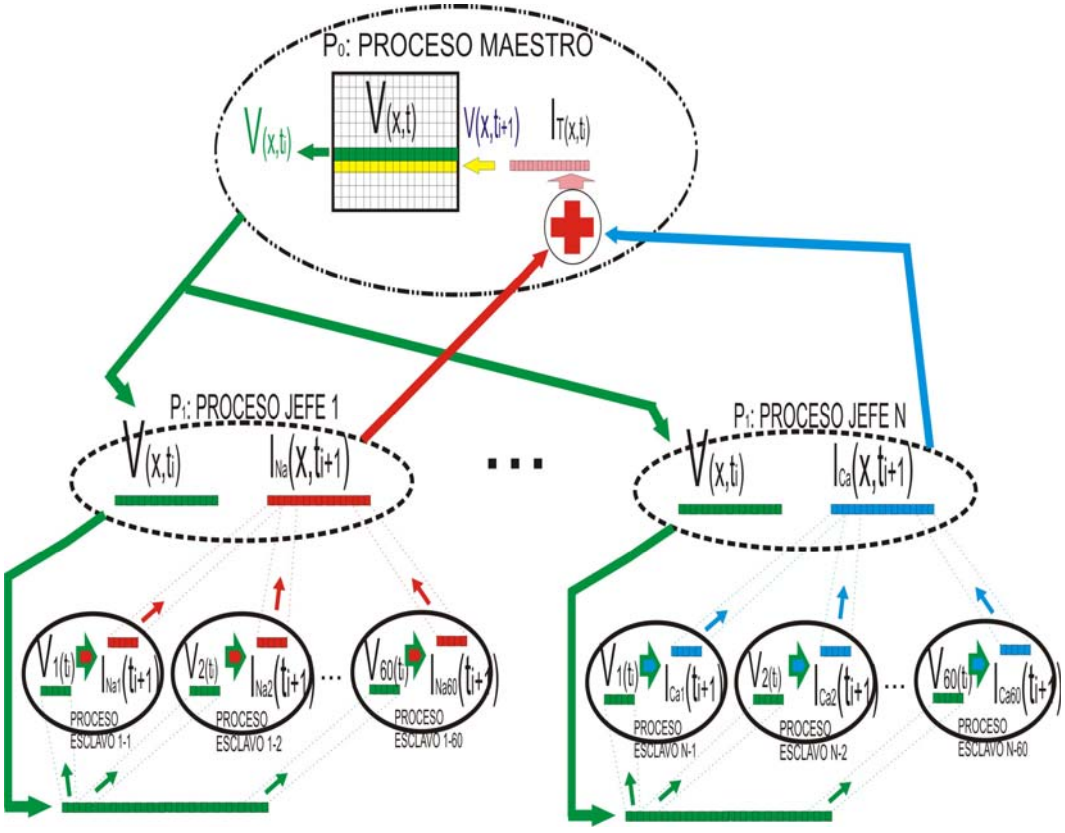

Figura 4.4. Diagrama Jerárquico para el algoritmo paralelo del NODO SA

(Fuente: Autor)

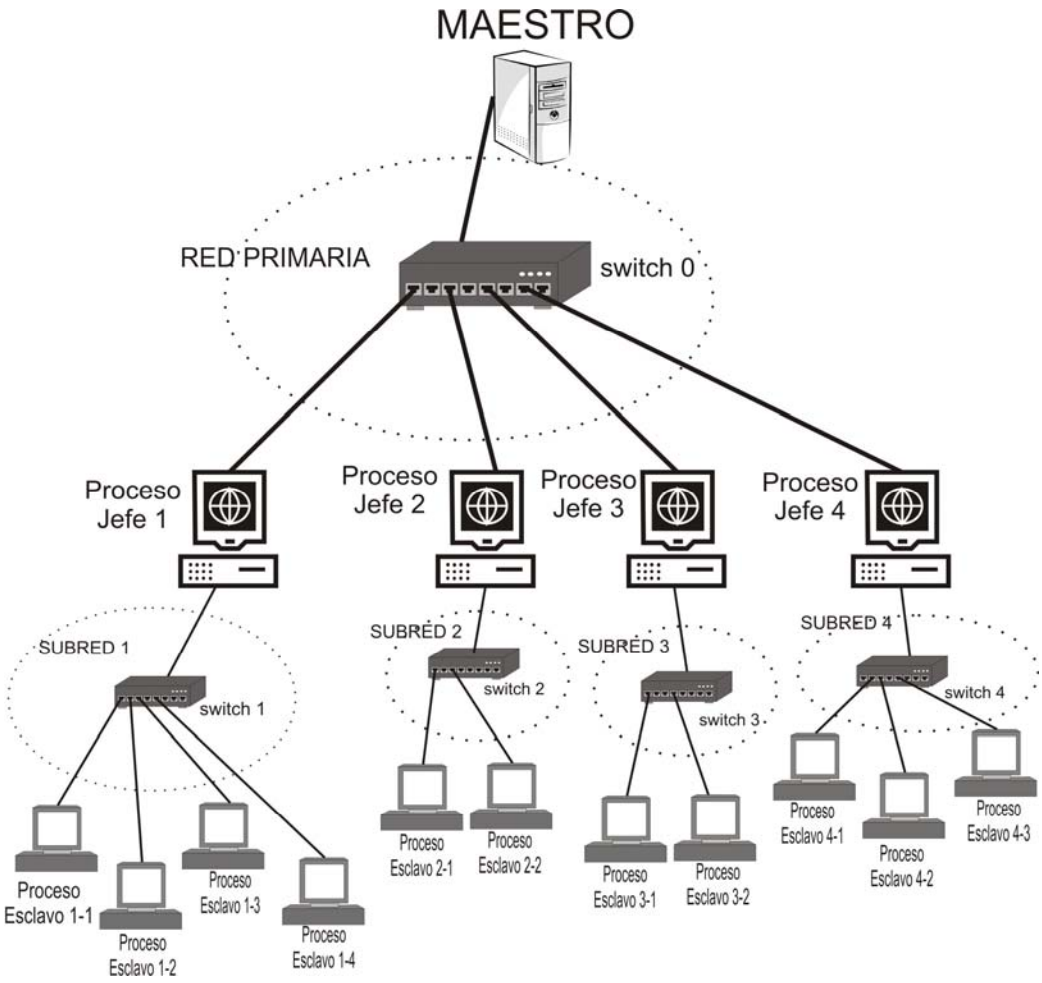

Figura 4.5. Ejemplo del esquema físico para el diagrama jerárquico del algoritmo paralelo del nodo SA. Se especifican 4 subredes asociadas al cálculo de las corrientes iónicas. (Fuente: Autor)

El balanceo de carga de este algoritmo se puede garantizar al asignar un número de esclavos proporcional a la complejidad del cálculo de la respectiva corriente, por lo tanto el proceso jefe asociado con el calculo de la corriente mas compleja tendrá a su cargo un numero mayor de procesos esclavos.

La dependencia espacial del potencial de membrana es evitada al realizar el calculo de forma centralizada, ya que el potencial de membrana lo calcula el proceso maestro y este aprovechando la independencia funcional de las corrientes iónicas lo envía a los procesos jefe del segundo nivel que corresponden con un determinado grupo de

corrientes iónicas, estos procesos jefes aprovechan la independencia espacial en el calculo de las corrientes iónicas y dividen este calculo entre sus procesos esclavo, por lo tanto el algoritmo resultante aprovecha la independencia que ofrece la descomposición funcional en un primer nivel evitando la dependencia de dominio que posee el potencial de membrana y una vez evitada esta dependencia aplicar en un segundo nivel la descomposición de dominio al calculo de las corrientes iónicas, evitando una mayor comunicación entre todos los procesos.

Por otro lado el esquema de dos niveles propuesto secciona el canal de comunicación entre los procesos permitiendo un canal especializado para cada subred de proceso jefe, así mismo concede un canal único entre los procesos jefe y el proceso maestro evitando el trafico de los procesos esclavo con paquetes relativamente pequeños. En la figura 4.5 se aclara este concepto con un ejemplo, observe que los procesos jefes son mapeados en servidores de subredes y cada una de estas subredes posee un determinado numero de maquinas en las cuales son mapeados los procesos esclavo. Por lo tanto el canal físico entre el proceso maestro y los procesos jefes es independiente del canal de los procesos esclavos, además cada subgrupo de procesos esclavo relacionados con una determinada corriente iónica posee un canal físico diferente con lo cual se logra independizar el tráfico de los procesos esclavos al pertenecer a una subred diferente.

La estructura jerárquica del algoritmo paralelo propuesto específica tres tipos de procesos: en primer lugar, el proceso maestro el cual divide las corrientes iónicas entre los procesos jefes, en segundo lugar, el proceso jefe encargado de realizar una descomposición espacial para una determinada corriente iónica, y el proceso esclavo que realiza los cálculos para un sector espacial de la corriente iónica dada. En la siguiente sección se presenta para cada uno de los tres procesos el diagrama de flujo paralelo, el cual es la base a la hora de realizar el respectivo programa de simulación.

4.5 DIAGRAMAS DE FLUJO PARA EL ALGORITMO PARALELO

El algoritmo paralelo desarrollado dentro de este trabajo es enfocado hacia la técnica de programación distribuida de paso de mensajes, por lo tanto los diagramas de flujo presentados en las figura 4.6, 4.7 y 4.8 utilizan las herramientas básicas de programación de paso de mensaje: SEND (enviar) y RECV (recibir). La palabra SEND hace referencia a un envió de datos hacia algún otro proceso, mientras que RECV es utilizada para denotar una orden de recepción de datos con algún origen previamente conocido. Es importante resaltar que cada orden de envío en un proceso origen debe corresponder con una orden de recibido en otro proceso destino, de lo contrario puede generarse información "basura" que retardará el trafico de la red.

4.5.1 PROCESO MAESTRO

En el algoritmo paralelo del nodo SA el proceso maestro es el encargado de computar el potencial de membrana a partir del cálculo de las corrientes iónicas presentes en el nodo SA. Sin embargo esta no es la única tarea que debe realizar, pues debe distribuir el cálculo entre todos los procesos de primer nivel (procesos jefes), las tareas más importantes que realiza el proceso maestro dentro del algoritmo paralelo son:

- Repartir el cálculo de las corrientes iónicas entre los procesos Jefes y proveerlos de los datos necesarios para dicho cálculo.
- Recolectar las corrientes iónicas de los procesos jefes para realizar el calculo de la corriente iónica total
- Computar el potencial de membrana para todo el nodo SA.
- Almacenar el potencial de membrana y la corriente iónica total.

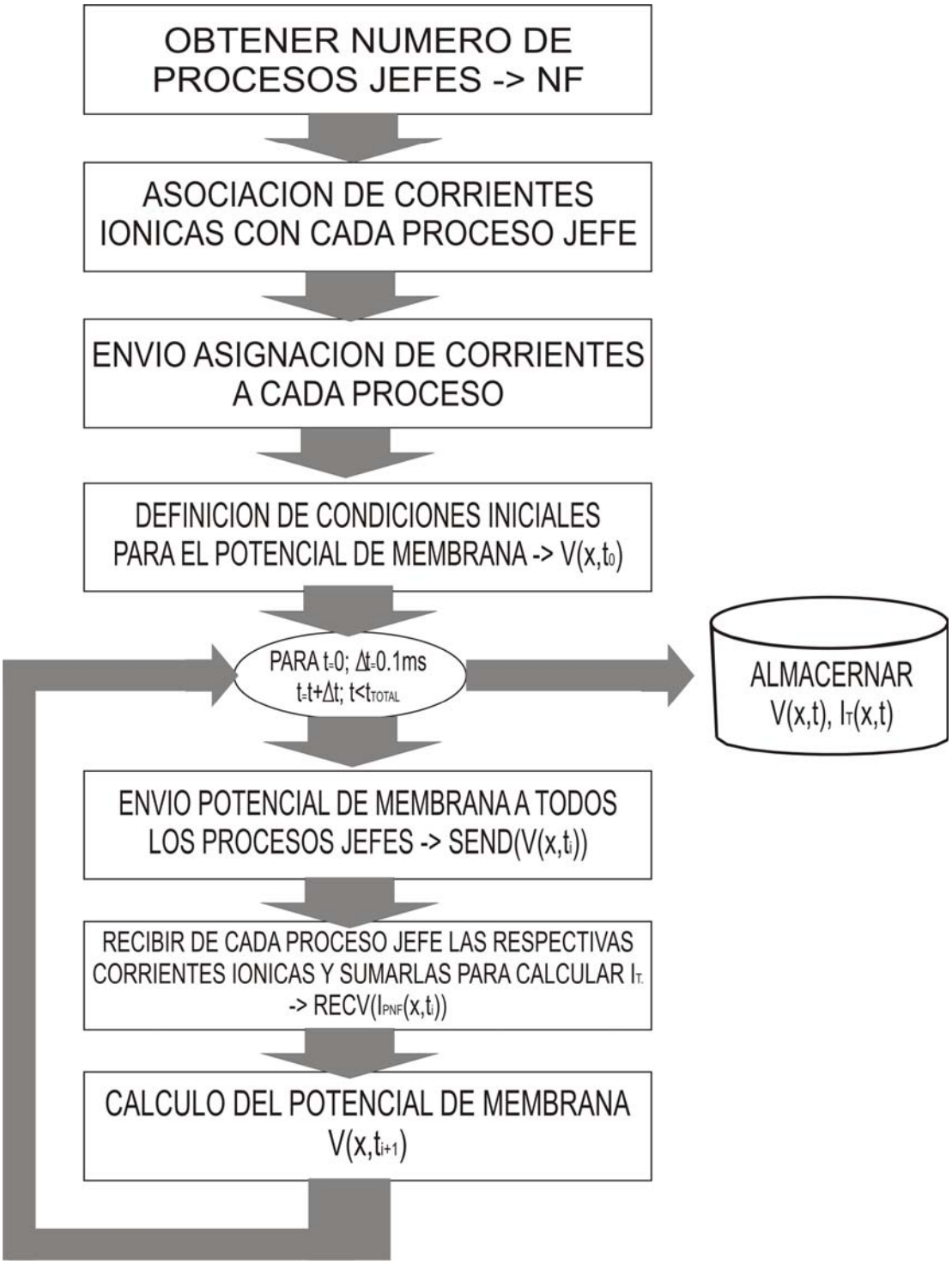

Figura 4.6 Diagrama de Flujo Proceso Maestro. (Fuente: Autor)

Teniendo en cuenta las características descritas anteriormente es posible plantear el diagrama de flujo para el proceso maestro, el cual se presenta en la figura 4.6. El diagrama consta de ocho tareas básicas de las cuales tres son iterativas. La primera tarea consiste en reconocer el entorno de trabajo, es decir, conocer cuantos procesos jefe tiene a su cargo y así poder realizar una partición del problema. El segundo paso es la asociación de cada corriente iónica a un proceso jefe para que este se encargue de su respectivo cómputo. En total el modelo considera 14 corrientes iónicas las cuales se encuentran especificadas en el anexo A (tabla A.1), sin embargo debido a la complejidad en los cálculos de unas corrientes respecto a otras, se ha optado por agrupar los cálculos de algunas corrientes con el propósito de balancear la carga entre los procesos, por lo tanto se consideran 8 tareas asociadas con el cómputo de las corrientes iónicas, agrupando en una sola tarea el cálculo de las componentes transitoria y permanente de la corriente sensitiva a la 4AP ($i_{to,}$ i_{sus}) y en otra tarea todos los cómputos relacionados con las corrientes de respaldo ($i_{b.Na,}$ $i_{b.K,}$ $i_{b.Ca}$), la corriente de intercambio ($i_{.NaCa}$) y las corrientes de bomba ($i_{p,}$ i_{CaP}).

En tercer lugar se tiene que informar a cada proceso que corrientes debe computar, este paso es importante ya que permite configurar dentro de cada subred los cálculos previos al proceso iterativo. Luego se tiene como cuarto paso la definición de las condiciones iniciales para el potencial de membrana, que corresponden con los valores dados en el anexo B (tabla B.1), sin embargo en esa tabla sólo aparecen los valores para el centro y la periferia por lo cual se debe interpolar estos valores para el resto de punto en el espacio. [ZHANG00]

Los pasos 5, 6 y 7 corresponden al proceso iterativo, para un tiempo t_i de simulación primero se envía el potencial de membrana a los procesos jefe, los cuales devolverán el valor de las corrientes iónicas para ese instante de tiempo, luego el proceso maestro debe sumar las corrientes iónicas para calcular la corriente iónica total y así poder calcular el potencial de membrana para un tiempo t_{i+1} con lo cual se vuelve a iniciar el proceso iterativo, dicho proceso seguirá hasta que el tiempo de simulación se complete. Una vez concluido el tiempo de simulación, el proceso maestro tendrá que almacenar el potencial de membrana y la corriente iónica total para poder ser accedidos por el usuario una vez finalizado el programa.

4.5.2 PROCESO JEFE

Dentro del algoritmo paralelo el proceso jefe es el más difícil de coordinar pues debe recibir órdenes del proceso maestro y emitir órdenes a los procesos esclavos que estén a su cargo. En la figura 4.7 se presenta el diagrama de flujo para el proceso jefe, allí se pueden distinguir 9 pasos los cuales son descritos con más detalle a continuación:

- Paso 1: Obtener el numero de procesos esclavos. Este paso es importante pues se obtienen el número de particiones en el dominio espacial que se deben realizar para las corrientes asignadas.
- Paso 2: Realizar la descomposición asociada con el numero de procesos esclavo, es decir, dividir la longitud total de puntos en el espacio entre todos los procesos esclavo, se recuerda que según se analizó anteriormente (sección 4.2) en total se tienen 60 puntos en el espacio acorde con los valores de barrido de la variable x (mm).
- Paso 3: Recibir del maestro la asignación del cálculo de las corrientes, en otras palabras saber qué corrientes debo calcular.
- Paso 4: Envío de la información a los procesos esclavo acerca de las corrientes que deben calcular para una determinada sección de espacio.
- Paso 5: Este es el primer paso dentro del proceso iterativo, el proceso maestro le envió el potencial de membrana para un tiempo t_i, por lo tanto el proceso jefe debe recibir estos datos ya que son necesarios para el calculo de cualquier corriente iónica.
- Paso 6: Una vez recibido el potencial de membrana, se debe repartir a los procesos esclavo según la partición de dominio realizada.
- Paso 7: Cada esclavo calculará una sección de las corrientes iónicas, por lo tanto el proceso jefe debe recibir y organizar esta información par poder ser enviada al proceso maestro.
- Paso 8: Se envía al maestro las corrientes iónicas que se calculan, con lo cual el proceso maestro obtiene el potencial de membrana para un tiempo t_{i+1} y por lo tanto se debe volver al paso 5.

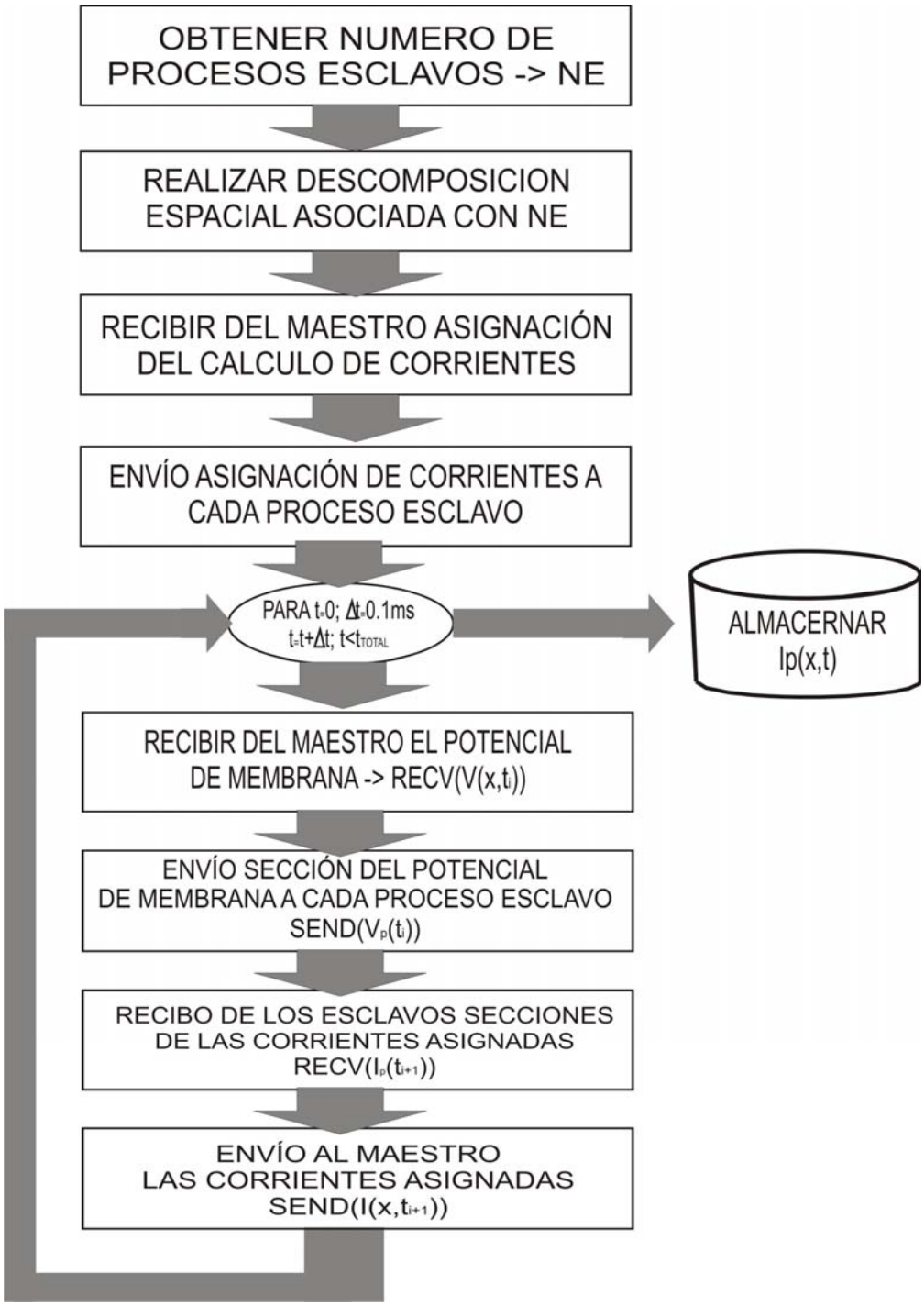

Figura 4.7. Diagrama de Flujo Proceso Jefe (Fuente: Autor)

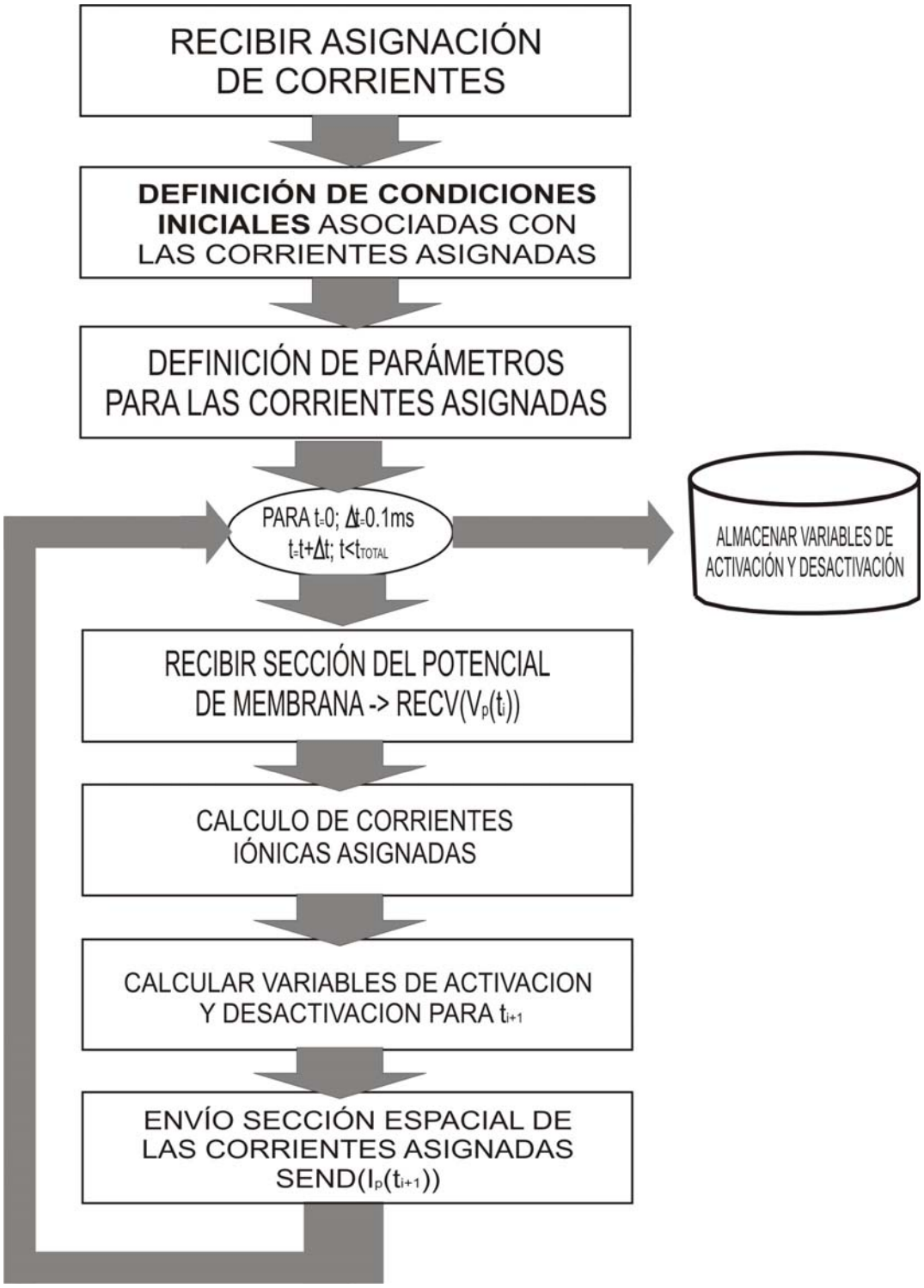

Figura 4.8. Diagrama de Flujo Proceso Esclavo (Fuente: Autor)

- Paso 9: Finalmente el proceso jefe almacenara las corrientes iónicas asignadas, para que estos datos puedan ser revisados por el usuario una vez terminada la simulación.

4.5.3 PROCESO ESCLAVO

El proceso esclavo es el encargado de realizar todos los cálculos correspondientes con una sección de las corrientes iónicas. En la figura 4.8 se muestra el diagrama de flujo para un proceso esclavo, este diagrama consta de 8 pasos básicos que son coherentes con el diagrama de flujo descrito para el proceso jefe.

En primer lugar el proceso esclavo debe ser informado acerca de las corrientes que debe calcular, por lo tanto, dado que el proceso jefe le envió esa información el debe recibirla para poder iniciar el computo. Una vez conocidas las corrientes que debe calcular, se definen para las corrientes asignadas las condiciones iniciales para cada variable de activación y desactivación en el intervalo espacial designado por el proceso jefe.
En tercer lugar se debe cargar e interpolar el valor de los parámetros asociados a las corrientes que se van a calcular, dichos parámetros se encuentran en el anexo B (tabla B.2) para el centro y la periferia, por lo tanto se deben interpolar para las demás zonas a lo largo del nodo SA. Una vez realizados estos tres primeros pasos, el proceso maestro esta listo para iniciar el proceso iterativo, el cual esta coordinado por todos los procesos en todos los niveles.

Los pasos del 4 al 7 corresponden al proceso de cálculo de las corrientes iónicas, para tal fin se reciben el sector del potencial de membrana que es necesario en el calculo de las corrientes iónicas. Una vez recibido el potencial se procede a calcular el valor de la corriente iónica para ese instante de tiempo y además se obtienen las variable de

activación y desactivación para un instante después ya que implícitamente se esta realizando el desarrollo de las respectivas ecuaciones diferenciales de primer orden. Con la corriente iónica calculada solo queda enviársela al proceso jefe para que este vuelva a enviarle el potencial de membrana para el siguiente instante y por lo tanto repetir el paso 4.

Finalmente se tiene la opción de almacenar las variables de activación y desactivación por si se desea analizar los resultados.

Capítulo 5
RESULTADOS

En este capitulo se presentan los resultados de la implementación del algoritmo paralelo del nodo sinusal, así como también se determina la efectividad del algoritmo, esto se logra comparando el tiempo de respuesta obtenido del algoritmo paralelo con el obtenido para el algoritmo secuencial. Tanto el algoritmo secuencial como el paralelo fueron implementados en MATLAB bajo el sistema operativo Linux, este hecho hizo necesario la utilización de una toolbox para el procesamiento paralelo llamada MPITB la cual es descrita en el anexo E.

5.1 REPRODUCIBIDAD DEL MODELO

Al realizar la simulación con los valores de condiciones iniciales y parámetros iguales a los sugeridos en [DASIERRA23] y relacionados en las tablas del anexo B se logro reproducir las características del potencial de membrana y las corrientes iónicas presentadas en el articulo del autor del modelo [ZHANG00]. En la figura 5.1 y la figura 5.2 se presentan las formas de onda obtenidas de la simulación paralela en el centro y la periferia del nodo SA, respectivamente.

Una característica importante de la simulación realizada es la obtención de las formas de onda para todos los puntos a lo largo del nodo SA y el músculo atrial, a diferencia del modelo base que solo obtiene las ondas para el centro y la periferia. En la figura 5.3 muestra la onda del potencial de membrana en el tiempo a lo largo de todo el nodo SA y el músculo atrial, esta grafica bidimensional permite apreciar la variación del

potencial de acción desde el centro del nodo SA hasta la periferia y como permanece constante a lo largo del músculo atrial, además se observa como los potenciales de acción primero se inician en el centro del nodo SA y se propagan a lo largo de la periferia y el músculo atrial.

La figura 5.4 presenta un diagrama tipo malla del nodo SA para un segundo de simulación, allí mediante el uso de colores se puede apreciar desde varias perspectivas la distribución del potencial en el tiempo, lo cual corrobora que la propagación del potencial inicia del centro hacia la periferia generando una frecuencia mayor a medida que se acerca al músculo atrial.

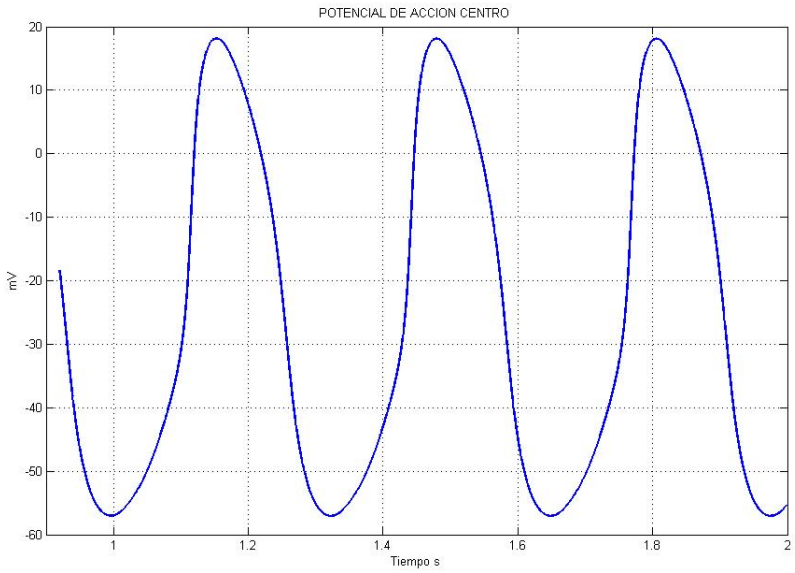

Figura 5.1. Potencial de Acción en el centro del nodo SA, obtenido de la simulación paralela

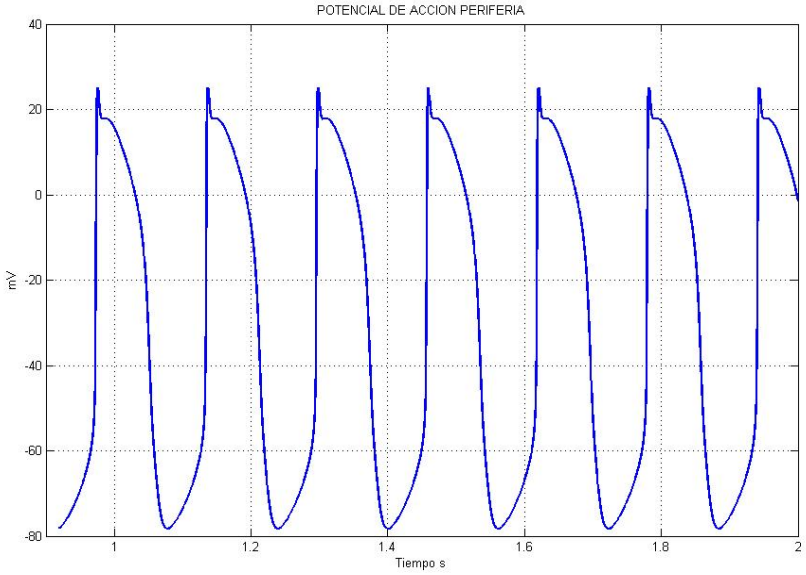

Figura 5.2. Potencial de Acción en la periferia del nodo SA, obtenido de la simulación paralela

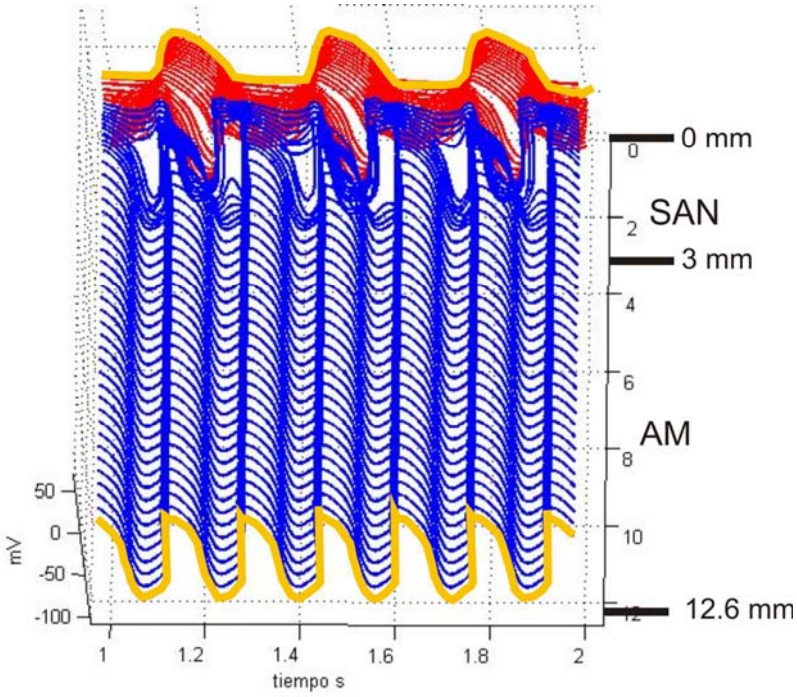

Figura 5.3. Potenciales de acción a lo largo del nodo SA completo

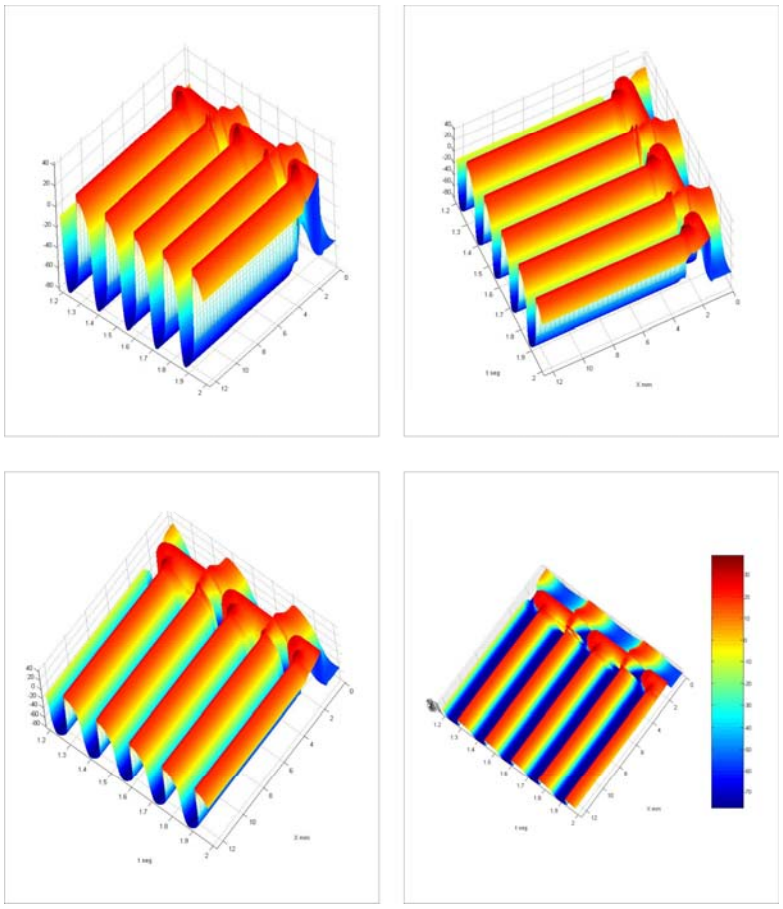

Figura 5.4. Distribución bidimensional del potencial de acción en el nodo SA completo

5.2 RESPUESTA DEL ALGORITMO SECUENCIAL.

El punto de comparación para comprobar la efectividad del algoritmo paralelo es el algoritmo secuencial propuesto por [DASIERRA23], sin embargo el código secuencial fue modificado para permitir obtener el potencial en el nodo SA completo. En la figura 5.5 se muestra la relación del tiempo de respuesta del código secuencial para diferentes valores de tiempo de simulación del potencial, de esta figura se puede observar que el tiempo de respuesta aumenta drásticamente si se incrementa de 2 a 3 segundos el tiempo de simulación, este se explica basándose en que para un tiempo de simulación mayor la cantidad de datos a manipular es mayor pese a que la longitud de espacial (x) se mantenga constante.

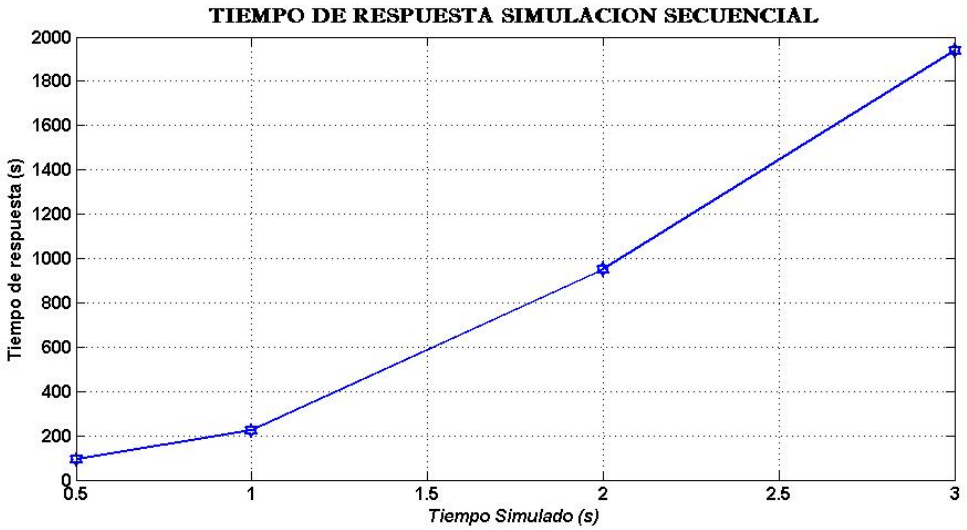

Figura 5.5 Tiempo de Respuesta para el algoritmo secuencial

En la tabla 5.1 se relacionan los tiempos de respuesta del programa secuencial con el tiempo de simulación obtenido, de la información relacionada en esa tabla se puede observar que la eficiencia disminuye conforme se pretende simular un mayor tiempo de funcionamiento del nodo SA, este hecho es desalentador si se quisiera hacer una simulación en tiempo real pues el retardo es de mas del 90 % del tiempo simulado, es decir para obtener la forma de onda a los 0.5 segundos de simulación se deben esperar aproximadamente 93 segundos, luego para observar la onda a los 2 segundos de simulación se tendrá que esperar 949 segundo por lo cual se debe descarta una simulación en tiempo real, sin embargo este retardo esta muy relaciona con la estructura que maneja el algoritmo para almacenar los datos, ya que el programa solo almacena la información al final de la simulación por lo cual los procesos deben manejar un gran volumen de información el cual aumenta en la medida que se extiende la simulación, por lo tanto si se deseara un algoritmo para simulación en tiempo real se debe pensar en arrojar la información a la par que se avanza en la simulación.

TIEMPO SIMULADO Ts (s)	DURACION DE SIMULACION (Td) (s)	EFICIENCIA (%) (Ts/Td)*100
0.5	**93.515**	**0.53**
1	**225.468**	**0.44**
2	**949.766**	**0.21**
3	**1938.432**	**0.15**

Tabla 5.1. Tiempo de respuesta para la simulación secuencial

Debido a que para un segundo de simulación la respuesta transitoria del modelo ha desaparecido, se considero conveniente realizar todas las simulaciones de prueba para dos segundo de simulación, por lo tanto se tomara de aquí en adelante el tiempo de respuesta en la simulación secuencial como 949.766 segundos que corresponde según la tabla 5.1 con la duración de la simulación para obtener las formas de onda del nodo SA entre cero y dos segundos.

5.3 RESPUESTA DEL ALGORITMO PARALELO.

El algoritmo paralelo descrito en el capitulo anterior posee una gran variedad de configuraciones, pues hay un grado de libertad bastante amplio a la hora de especificar la topología jerárquica. Aquí se acortan las posibilidades considerando constantes el numero de procesos esclavo para cada proceso jefe, además pese a que el algoritmo planteado permite hasta 60 esclavos por proceso jefe aquí se limita el análisis a una red de 17 computadores incluido el proceso maestro. Otro factor limitado en las pruebas realizadas para el algoritmo paralelo fue la existencia de un solo canal de conexión físico para todos los procesos involucrados en la simulación. En la tabla 5.2 se presentan la especificaciones técnicas de la red empleada para desarrollar las simulaciones del algoritmo paralelo, además cabe aclarar que los resultados para el algoritmo secuencial presentados anteriormente se obtuvieron al ejecutar el código en uno de los computadores de la red, esto con el fin de conservar las características de simulación lo mas homogéneas posibles.

ESPECIFICACIONES TECNICAS DE LA RED UTILIZADA	
COMPUTADORES	
	PROCESADOR AMD SEMPRON 2800+
	MAINBOARD MSI K8MM
	MEMORIA RAM DDR 512 MB
	DISCO DURO 80 GB
	TARJETA RED 10 BASE 100
RED	
	SWITCH 8 PUERTOS 10/100
	SWITCH 16 PUERTOS 10/100
	CABLEADO UTP
	TOPOLOGIA = BUS

Tabla 5.2. Especificaciones técnicas de la red utilizada para la simulación paralela

La metodología utilizada para obtener las diferentes combinaciones entre procesos jefe y esclavos se baso en las siguientes consideraciones:

- El número máximo de procesos jefe no puede superar las 8 corrientes consideradas en el algoritmo, ya que por objeto de balanceo de carga se opto por agrupar los cálculos correspondientes con algunas corrientes en uno solo. Además los 8 grupos de corrientes iónicas consideradas limitan aun mas las opciones para los procesos jefe, considerando solo los divisores de 8, pues de lo contrario se desbalanceara la carga computacional.
- Cada proceso jefe tendrá a su cargos igual numero de procesos esclavos. Esto con el fin de acortar las opciones, ya que el algoritmo permite diversidad en la distribución de los procesos esclavo (ver figura 4.5).
- El numero total de procesos jefe y esclavo no puede superar el número de equipos disponibles en la red (16).

- No se consideran los casos para los cuales se transfiera la totalidad del trabajo a un proceso esclavo, es decir, no se considera el número de procesos esclavo igual a uno.

En la tabla 5.3 se relacionan los casos realizados dentro de las simulaciones efectuadas, además se especifica las condiciones de trabajo para el determinado caso tratado, por ejemplo para un numero de procesos jefe igual a 4 con 2 procesos esclavo cada uno, se entenderá que cada proceso jefe debe calcular 2 grupos de corrientes y cada proceso esclavo manipulara vectores de datos de longitud 30 para el calculo de las respectivas corrientes, se recuerda que la longitud de puntos considerados para la variable espacial es fija e igual a 60.

NE	NF → 2	4	8
0	Cada Proceso Jefe realiza todos los cálculos para 4 grupos de corrientes	Cada Proceso Jefe realiza todos los cálculos para 2 grupos de corrientes	Cada Proceso Jefe realiza todos los cálculos para 1 grupo de corrientes
2	Proceso jefe coordina 4Is Proceso Esclavo computa vectores de tamaño 30	Proceso jefe coordina 2 Is Proceso Esclavo computa vectores de tamaño 30	NA
3	Proceso jefe coordina 4 Is Proceso Esclavo computa vectores de tamaño 20	Proceso jefe coordina 2 Is Proceso Esclavo computa vectores de tamaño 20	NA
4	Proceso jefe coordina 4 Is Proceso Esclavo computa vectores de tamaño 15	NA	NA
5	Proceso jefe coordina 4 Is Proceso Esclavo computa vectores de tamaño 12	NA	NA
6	Proceso jefe coordina 4 Is Proceso Esclavo computa vectores de tamaño 10	NA	NA
7	Proceso jefe coordina 4 Is Proceso Esclavo computa vectores de tamaño 8	NA	NA

Tabla 5.3. Casos Realizados en las simulaciones paralelas.
NA: NO APLICA, EXCEDE EL NUMERO DE EQUIPOS DISPONIBLES
NE: Numero de procesos Esclavo, NF: Numero de procesos Jefe, Is: Corrientes Ionicas

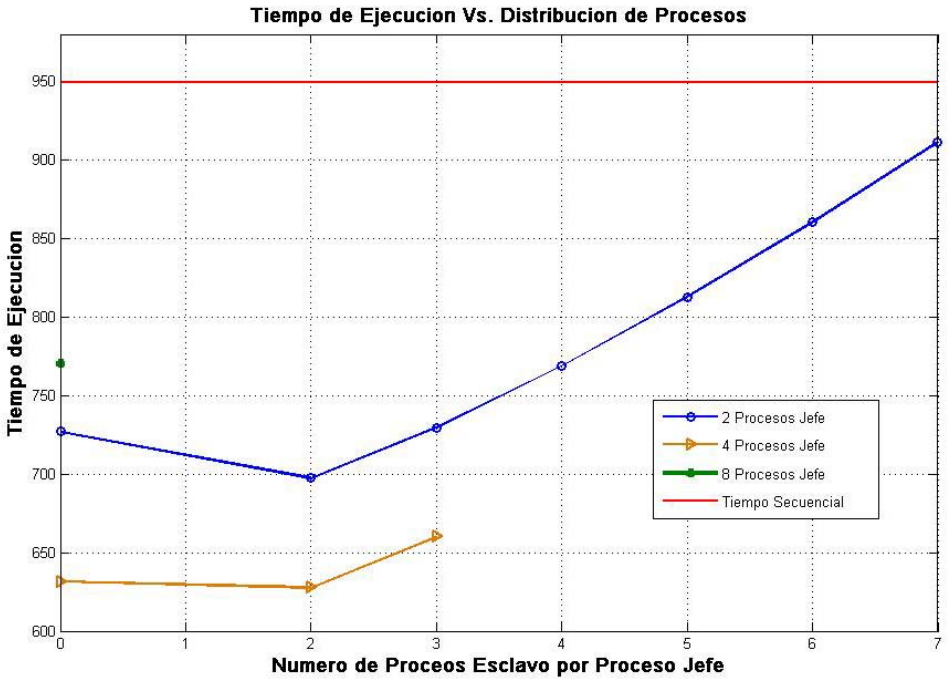

Figura 5.6. Tiempo de ejecución para el algoritmo paralelo con diferentes configuraciones

Una vez definidas las combinaciones posibles de simulación se procedió a realizar las respectivas simulaciones con lo cual se obtuvo la grafica de distribución de tiempos presentada en la figura 5.6. En esta grafica se relacionan los tiempos obtenidos para cada configuración con el valor del tiempo de respuesta para la simulación secuencial, los resultados son satisfactorios, ya para la gran mayoría de los casos se disminuye en mas de un 20% (aprox.190 segundos) con relación al tiempo de respuesta base tomado del algoritmo secuencial.

De esta grafica también se observa que el tiempo de respuesta tiende a disminuir según se utilicen menos procesos esclavo, sin embargo la no utilización de procesos esclavo desmejora la respuesta en relación a utilizar dos procesos excepto para el caso de 8 procesos jefe, en el cual la mejor opción es no utilizar procesos esclavo. De la grafica 5.6 se puede concluir que para un número de procesos esclavo mayor a 2, el tiempo de respuesta tiende a aumentar casi de manera lineal. Este hecho se explica considerando que para un número elevado de procesos esclavo, el canal de comunicación, que en

este caso fue compartido, tiende a ser accesado más frecuentemente y con una menor cantidad de datos (mayor fraccionamiento de dominio), lo cual genera un mayor trafico y por ende un retardo por transmisión mas alto. De hay la importancia del canal de comunicación, ya que para este tipo de simulaciones que necesitan una gran cantidades de comunicaciones el mas mínimo retardo produce un retraso considerable en la salida pues cada comunicación se encuentra dentro de un proceso iterativo y el retardo en una tarea posiblemente retardara a la siguiente.

Otro hecho a resaltar es que los mejores tiempos de respuesta se obtuvieron para la configuración de 4 procesos jefe, sin embargo la figura 5.6 no clarifica cual es la distribución más efectiva, por lo tanto se procede a realizar una distribución de contorno superficial de los datos relacionados en la figura 5.6 con lo que se obtiene el diagrama de contorno presentado en la figura 5.7, la interpretación de este diagrama permite concluir que efectivamente los mejores resultados se tienen para un numero de procesos jefe no superior a cuatro, ya que la distribución de color es centrada en el eje de 4 procesos jefe, además se observa que el vértice de esta distribución se encuentra localizado aproximadamente en el punto (4,2) que corresponde al caso de 4 procesos jefe cada uno con 2 procesos esclavo para el cual se obtuvo un tiempo de respuesta promedio de 627,7 segundos. Este resultado puede ser explicado si se considera que con esta configuración se esta aprovechando las mejores características en la partición funcional y de dominio para este determinado caso; cuatro procesos parten los cálculos de manera que cada proceso debe coordinar el computo de dos corrientes y debido a que solo poseen dos procesos esclavo, la partición de dominio es la mínima haciendo operaciones con vectores de longitud 30, lo que permite evitar el retardo en la transmisión debido al trafico ineficaz del canal ya que el numero de proceso esclavo es bajo.

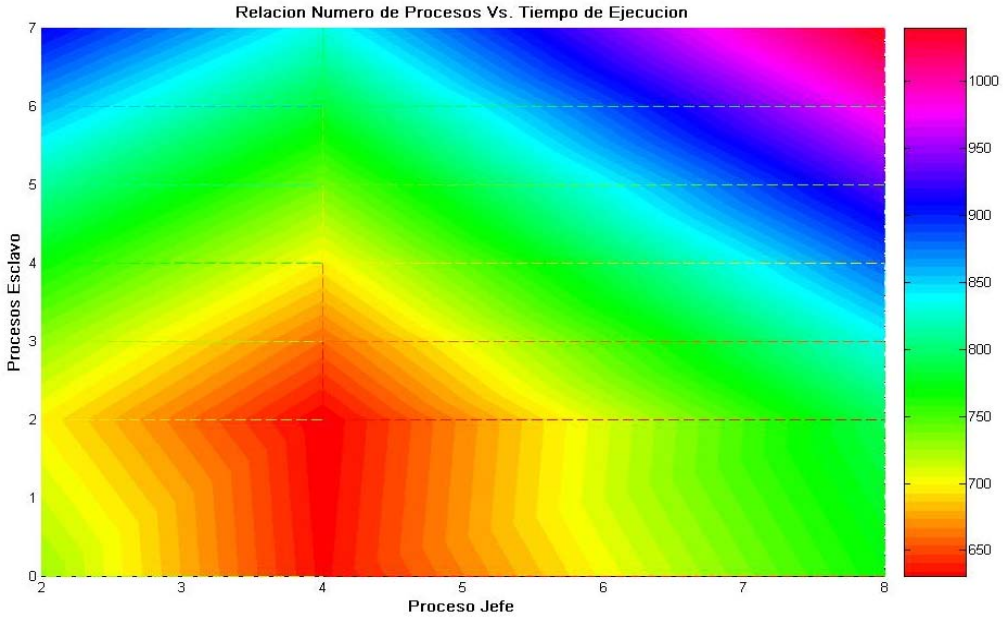

Figura 5.7. Diagrama de contorno, distribución del algoritmo paralelo

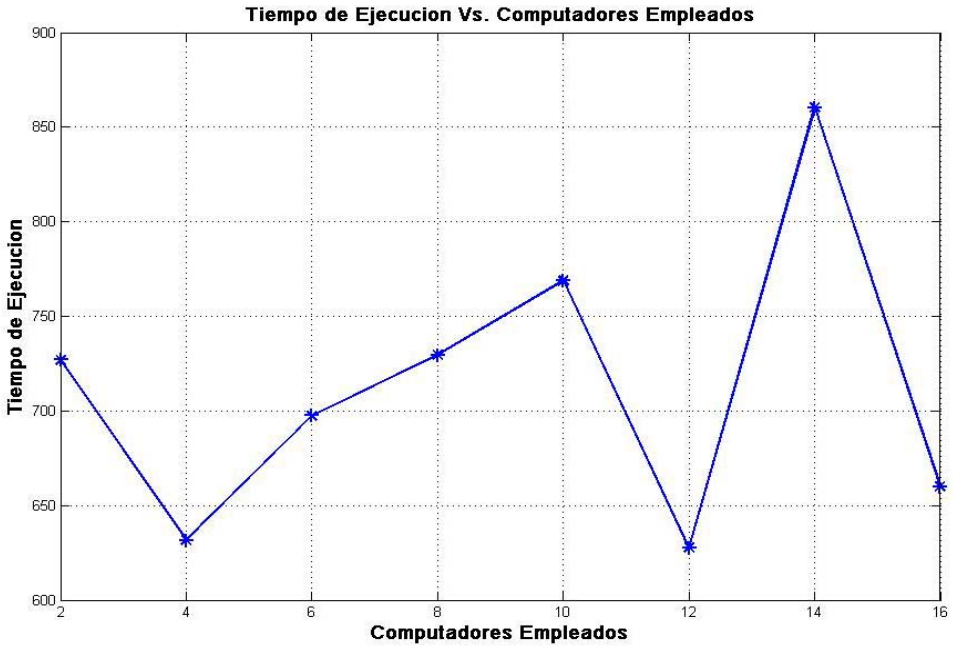

Figura 5.8. Relación Tiempo de duración de la simulación vs. Numero de computadores usados

Numero de Maquinas Empleadas	Tiempo total de simulación (s)	Eficiencia del código paralelo	Aceleración
4	631,64	0,38	1,5
8	729.57	0,16	1,3
12	627.70	0,13	1,51
16	659.99	0,09	1,44

Tabla 5.4. Eficiencia y aceleración del algoritmo paralelo

Por ultimo la figura 5.8 muestra una relación entre el tiempo de retardo y el número de computadores empleados para la simulación del nodo SA. De allí se observa que se obtiene un mejor tiempo de respuesta para un número de computadores igual a 12 (combinación 4 procesos jefe y dos esclavo por proceso jefe) lo cual es acorde con los resultados obtenidos de las graficas anteriores, además se observa que la cota inferior de tiempo pertenece a los resultados obtenidos para un número de procesos jefe igual a 4, además a la par con esta gráfica se realizo la tabla 4 que refiere el numero de computadores usado con la aceleración y eficiencia del algoritmo paralelo según las ecuaciones 3.2 y 3.3 presentadas en el capitulo 3. De la tabla 5.4 se observa que pese a que el menor tiempo de respuesta (627,7 s) se obtiene para 12 computadores la eficiencia (0,13) en este caso es mas baja que para el caso de cuatro computadores (0,38), este hecho se debe a que la aceleración para 4 y 12 computadores es aproximadamente igual, causando una ineficiente utilización de recursos para el caso de 12 computadores ya que se obtiene una mejora poco significativa en el tiempo de respuesta.

Capítulo 6

CONCLUSIONES, APORTES Y TRABAJOS FUTUROS

CONCLUSIONES

- Las características propias del modelo matemático del nodo SA completo derivadas del conjunto total de ecuaciones involucradas en el calculo del potencial de acción permitieron sugerir dos posibles esquemas de paralelización por medio de la técnica de memoria distribuida; el primero sugiere una descomposición de dominio en la variable espacial, mientras el segundo se basa en la descomposición funcional asociada con el calculo de las corrientes iónicas.

- El análisis del problema es sino por mucho el paso mas importante dentro de la metodología de desarrollo de algoritmos paralelos, ya que para el algoritmo paralelo del nodo SA se logró identificar dos esquemas de descomposición diferentes, con lo cual se genero una nueva perspectiva del problema que fusiona las características mas favorables de la descomposición de dominio espacial y la descomposición funcional de las corrientes iónicas.

- Se lograron reproducir las características del potencial de acción de membrana y las corrientes iónicas presentadas por el autor del modelo del nodo sinoatrial en [ZHANG00] y además se modificó el algoritmo secuencial base para obtener las formas de onda a lo largo de todo el nodo SA y el músculo atrial, ya que este se limitaba a encontrar las ondas para el centro y la periferia del nodo SA.

- La simulación basada en el algoritmo paralelo logró, para el mejor caso, disminuir el tiempo de repuesta en un 34% con relación a la simulación fundamentada en el algoritmo secuencial, lo cual demuestra la potencialidad del algoritmo paralelo.

- A pesar de la disminución en el tiempo de respuesta utilizando el algoritmo paralelo, se observa que aun este tiempo es muy elevado, ya que para simular 2 segundos de funcionamiento del nodo SA se hace necesario un retardo de aproximadamente 627 segundos lo cual es un precio alto relativo al tiempo de simulación calculado. Esto se refleja en la eficiencia del algoritmo paralelo calculada en la tabla 5.4, la cual para el mejor de los casos fue de tan solo 0.38. Sin embargo esto no significa que el algoritmo paralelo sea inapropiado para el modelado del nodo SA, ya que los resultados obtenidos son el reflejo de la implementación del algoritmo en una plataforma basada en el paso de mensajes y existen otras alternativas que podrían adecuarse a las características del modelo planteado por [ZHANG00]

- Para la simulación del nodo SA basada en el algoritmo secuencial, se observó un aumento exponencial en el tiempo de respuesta ante una ampliación en el tiempo de simulación, por lo cual se descarta este esquema para una simulación en tiempo real. Sin embargo se notó que este aumento se debe a que el programa debe manipular y almacenar matrices de gran tamaño y este tamaño es proporcional al tiempo de simulación, por lo tanto se plantea para una simulación en tiempo real un esquema que permita el manejo de matrices de dimensión reducida mediante la opción de imprimir los resultados a medida que se van calculando dentro de la simulación. Esto evita el manejo de matrices de enormes dimensiones pues un archivo completo de simulación típicamente sobrepasa los 200 MB lo cual afecta el desempeño de la plataforma de simulación.

APORTES

- Se realizó una investigación y recopilación bibliográfica sobre la configuración de un cluster no dedicado orientado al procesamiento paralelo mediante la técnica del paso de mensajes, dicha información esta referenciada en los capitulos y anexos 2, 3, C, D y E, respectivamente, y puede ser utilizada con el fin de emprender nuevas investigaciones relacionadas con la paralelización de un determinado problema.

- Se muestra que es posible mejorar el tiempo de respuesta para la simulación del modelo matemático del nodo SA mediante el uso de técnicas de programación paralelo.

TRABAJOS FUTUROS

- Dada la gran dependencia temporal presentada en el modelo del nodo SA, se debe enfocar la implementación del algoritmo paralelo a ambientes de memoria compartida en donde el tiempo de acceso a datos es menor que en el esquema de memoria distribuida, siendo este último es la base de la plataforma escogida para la implementación de este trabajo de grado. Con esto se muestra un camino alentador en la búsqueda de nuevas mejoras en la simulación del nodo SA para poder obtener una simulación satisfactoria en tiempo real.

- Implementar algoritmos que simulen el funcionamiento de cada una de las partes que componen el sistema eléctrico del corazón y realizar el respectivo acople para obtener una simulación del sistema eléctrico completo del corazón.

BIBLIOGRAFIA

[ALON97] ALONSO JOSE MIGUEL. "Programación de aplicaciones paralelas con MPI (*Message Passing Interface*)". Facultad de Informática UPV/EHU. Enero 1997

[BAR22] BARRIOS HERNANDEZ CARLOS J., CASALLAS BOLIVAR JUAN C. "Proyecto Cúmulos: Desarrollo de aplicaciones para el procesamiento paralelo en una red de pc's" Universidad Industrial de Santander. Colombia 2002.

[BREW97] BREWER E. "Clustering: Multiply and Conquer" Data Communications 1997

[DASIERRA23] SIERRA BUENO DANIEL ALFONSO. "Modelado Matemático del Nodo Sinusal: Estudio de la Sensibilidad Paramétrica". Universidad Industrial de Santander, Proyecto de Investigación de Maestría, Colombia 2003.

[*DEM94*] DEMIR, S.S., J. W. CLARK, C. R. MURPHY, and W. R GILES "A Mathematical Model of a rabbit sinoatrial node cell" Am. J. Physiol. 266 (Cell Physiol. 35) C832-C852, 1994

[GAR23] GARCIA LEIVA RAFAEL A. "Instalación y Configuración de un Cluster de Alta Disponibilidad Bajo Linux". Universidad Autónoma de Madrid. Abril 2003

[HOE23] HOEGER HERBERT. "Introducción a la Programación Paralela". Centro Nacional de Cálculo Científico. Universidad de Los Andes. *CeCalCULA.* Mérida - Venezuela

[KAR24] KARNIADAKIS GEORGE, KIRBY II ROBERT. "Parallel Scientific Computing in C++ and MPI". Cambridge University Press. 2003.

[LLO22] LLORENTE IGNACIO MARTIN. "Tipos De Aplicaciones Paralelas" Dpto. Arquitectura de Computadores y Automática, Universidad Complutense, Madrid España. 2002

[MANT25] MANTAS RUIZ JOSE MIGUEL. "Diseño Sistemático de Algoritmo Paralelos". Programación Distribuida y Paralela, Departamento. De Lenguajes y Sistemas Informáticos. Universidad de Granada. España 2005.

[QUINN94] QUINN M. "Parallel Computing: Theory and Practice" McGraw-Hill, New York 1994

[RUSS20] RUSSELL RUSTY. "Linux Networking-concepts HOWTO". Volumen 1.0.1 May 2000.

[STALL20] STALLINGS WILLIAM. “Procesamiento Paralelo” Capitulo 16 del libro “Organización y Arquitectura de Computadores” Quinta edición, Prentice Hall Iberia, Madrid, España 2000

[YAÑ00] YAÑES GOMEZ ROSA MARIA. “Introducción a las Tecnologías de Clustering en GNU/Linux”. Version 1.0. June 2000.

[ZHANG00] ZHANG H, HOLDEN AV, KODAMA I, HONJO H, LEI M, ARGHESE T, AND BOYETT MR. “Mathematical models of action potentials in the periphery and center of the rabbit sinoatrial node.” American Journal Physiology Heart And Circulatory Physiology, vol 279: H397-H421, July 2000

DOCUMENTACION WEB

1. ***FireWire***

   <http://www.apple.com/firewire/>

2. ***FSMLabs Inc. creadores de RTLinux***

   <http://www.rtlinux.org/>

3. ***Internet Parallel Computing Archive***

   <http://wotug.ukc.ac.uk/parallel/>

4. ***LAM/MPI Parallel Computing***

   <http://www.lam-mpi.org/>

5. ***MPI Forum***

   <http://www.mpi-forum.org/>

6. ***MPICH - A Portable Implementation of MPI***

   <http://www.mcs.anl.gov/mpi/mpich/>

7. ***MPICH Downloads***

   <http://www.mcs.anl.gov/mpi/mpich/download.html>

8. ***MPICH Parches***

   <http://www.mcs.anl.gov/mpi/mpich/buglist-tbl.html>

9. ***OpenSSH - Versión libre de SSH***

   <http://www.openssh.com/>

10. ***Packet Engines Gigabit Ethernet with Linux***

    <http://www.scyld.com/network/yellowfin.html>

11. ***PVM - Parallel Virtual Machine***

    <http://www.epm.ornl.gov/pvm/pvm_home.html>

12. ***The Beowulf Underground***

    <http://www.beowulf-underground.org/>

13. ***The Berkeley NOW Project***

    <http://now.cs.berkeley.edu/>

14. ***The Globus Project***

    <http://www.globus.org/>

15. **CeCalCULA:**

<http://www.cecalc.ula.ve/>

16. ***MPITB – TOOLBOX MATLAB***

<http://atc.ugr.es/~javier/investigacion/mpitb/>

17. ***High Performance Computing and Communications Glossary:***

<http://nhse.npac.syr.edu/hpccgloss/hpccgloss.html>

18. ***3Com Glossary of Networking Terms:***

<http://www.3com.com/>

19. ***Compilador GCC***

<http://gcc.gnu.org>

20. ***MATLAB***

<http://www.mathworks.com/products/distribtb/>

21. ***LINIX REDHAT***

http://www.redhat.com

ANEXO A
CONVENCIONES PARA EL MODELO DEL NODO SA[*]

Tabla A.1.

Convención	*Unidades*	*Descripción*
$V^a(x)$, $V^s(x)$, V	mV	Potencial de Membrana de las células del músculo atrial o del nodo sinusal a una distancia x del centro del nodo.
$C_m{}^a(x)$ $C_m{}^s(x)$	μF	Capacidad de la célula del músculo atrial o del nodo sinusal a una distancia x del centro del nodo.
x	mm	Distancia desde el centro del nodo
L^s	mm	Longitud total desde el centro a la periferia del nodo SA
L^a	mm	Longitud total del músculo atrial
$i_{tot}{}^a(x)$ $i_{tot}{}^s(x)$	nA	Corriente iónica total en las células del músculo atrial o del nodo SA a una distancia x del centro del nodo SA.
i_{to} , i_{sus}	nA	Componentes transitoria y permanente de la corriente sensitiva a la 4AP.
i_{Na}	nA	Corriente de sodio Na^+, sensitiva al TTX
$i_{Ca\,L}$	nA	Corriente de calcio de tipo lenta
$i_{Ca\,T}$	nA	Corriente de calcio de tipo rápida
$i_{K,\,r}$	nA	Corriente de potasio de rectificación retardada rápida
$i_{K,\,s}$	nA	Corriente de potasio de rectificación retardada lenta
i_f	nA	Corriente activada por hiperpolarización
$i_{f.Na}$ e $i_{f.K}$	nA	Componentes de sodio y potasio de la corriente i_f
$i_{b.Na}$ $i_{b.K}$ $i_{b.Ca}$	nA	Corrientes de respaldo de Na, K, y Ca
$i_{.NaCa}$	nA	Corriente de intercambio de Na^+/Ca^{++}
i_p	nA	Corriente de bomba Na^+/K^+
$\bar{i}_p$	nA	Valor máximo de i_p
i_{CaP}	nA	Corriente de bomba de Ca^{++}
E_{Na} , E_K E_{Ca}	mV	Potenciales de equilibrio para los iones de Na^+, K^+ y Ca^{++}
$E_{Ca,L}$	mV	Potencial de inversión para $i_{Ca,\,L}$
$E_{Ca,T}$	mV	Potencial de inversión para $i_{Ca,\,T}$
$E_{K,s}$	mV	Potencial de inversión para $i_{K,\,s}$
$[Na^+]_0$ $[Na^+]_i$	mM	Concentraciones extracelular e intracelular de Na^+
$[Ca^{++}]_0$ $[Ca^{++}]_i$	mM	Concentraciones extracelular e intracelular de Ca^{++}
$[K^+]_0$ $[K^+]_i$	mM	Concentraciones extracelular e intracelular de K^+
F	C/mol	Constante de Faraday
R	J/(mol.K)	Constante Universal de los Gases
T	K	Temperatura Absoluta
z	-	Valencia del ión
$g_{to,}$ g_{sus}	μS	Conductancia de i_{to} e i_{sus}
g_{Na}	μS	Conductancia de i_{Na}
$g_{Ca,\,L}$	μS	Conductancia de $i_{Ca,\,L}$

[*] Tabla tomada de [DASIERRA]

Tabla A.2. (Cont.)

Convención	***Unidades***	***Descripción***
$g_{Ca,\,T}$	μS	Conductancia de $i_{Ca,\,T}$
$g_{K,\,r}$	μS	Conductancia de $i_{K,\,r}$
$g_{K,\,s}$	μS	Conductancia de $i_{K,\,s}$
$g_{f,\,Na,}\, g_{f,\,K}$	μS	Conductancia de las componentes de Na y K de i_f
$g_{b,\,Na,}\, g_{b,\,K}\, g_{b,\,Ca}$	μS	Conductancia de $i_{b.Na}$ $i_{b.K}$ e $i_{b.Ca}$
q	-	Variable de desactivación de i_{to}
r	-	Variable de activación de i_{to}
q_∞ , r_∞	-	Valores estacionarios (finales) de q y r
τ_q , τ_r	s	Constantes de tiempo de q y r
h_1, h_2	-	Variables de desactivación rápida y lenta de i_{Na}
m	-	Variable de activación de i_{Na}
$h_{1\infty}$,$h_{2\infty,}$ m_∞	-	Valores estacionarios (finales) de h_1, h_2, m
τ_{h1} , $\tau_{h2,}$ τ_m	s	Constantes de tiempo de h_1, h_2, m
F_{Na}	-	Fracción de desactivación de i_{Na} que ocurre lentamente
$f_{L,}$	-	Variable de desactivación de $i_{Ca,\,L}$
$d_{L,}$	-	Variable de activación de $i_{Ca,\,L}$
$\alpha_{\mathbf{dL}}$ $\alpha_{\mathbf{fL,}}$	s^{-1}	Constantes de rapidez de apertura dependientes del voltaje
$\beta_{\mathbf{dL}}$, $\beta_{\mathbf{fL}}$	s^{-1}	Constantes de rapidez de cierre dependientes del voltaje
$d_{L\infty}$,$f_{L\infty,}$	-	Valores estacionarios (finales) de d_L , f_L
$\tau_{\mathbf{dL}}$, $\tau_{\mathbf{fL}}$	s	Constantes de tiempo de d_L , f_L
f_T	-	Variable de desactivación de $i_{Ca,\,T}$
d_T	-	Variable de activación de $i_{Ca,\,T}$
$\alpha_{\mathbf{dT.}}$ $\alpha_{\mathbf{fT}}$	s^{-1}	Constantes de rapidez de apertura dependientes del voltaje
$\beta_{\mathbf{dT}}$, $\beta_{\mathbf{fT}}$	s^{-1}	Constantes de rapidez de cierre dependientes del voltaje
$d_{T\infty}$,$f_{T\infty}$	-	Valores estacionarios (finales) de d_T , f_T
$\tau_{\mathbf{dT}}$, $\tau_{\mathbf{fT}}$	s	Constantes de tiempo de d_T , f_T
p_i	-	Variable de desactivación para $i_{K,\,r}$
p_a	-	Variable de activación general para $i_{K,\,r}$
$p_{a\,f}$, $p_{a,s}$	-	Variables de activación rápida y lenta para $i_{K,\,r}$
$F_{K,r}$	-	Fracción de activación de $i_{K,\,r}$ que ocurre lentamente
$p_{a,s\infty}$, $p_{a,f\infty}$ $p_{i\infty}$	-	Valores estacionarios (finales) de $p_{a,s}$, $p_{a,f}$ y p_i
$\tau_{\ pa,s}$, $\tau_{\ pa,f,}$ $\tau_{\ pi}$	s	Constantes de tiempo de $p_{a,s}$, $p_{a,f}$ y p_i
x_s	-	Variable de activación para $i_{K,\,s}$
$\alpha_{\mathbf{xs}}$	-	Constante de rapidez de apertura dependiente del voltaje
$\beta_{\mathbf{xs}}$	-	Constante de rapidez de cierre dependiente del voltaje
$x_{s\,\infty}$	-	Valor estacionario (final) de x_s
$\tau_{\ xs}$	s	Constante de tiempo de x_s
y	-	Variable de activación para i_f
$\alpha_{\mathbf{y}}$	-	Constante de rapidez de apertura dependiente del voltaje
$\beta_{\mathbf{y}}$	-	Constante de rapidez de cierre dependiente del voltaje
y_∞	-	Valor estacionario (final) de y
$\tau_{\ y}$	s	Constante de tiempo de y
$k_{.NaCa}$	nA	Factor de escala para $i_{.NaCa}$
$\gamma_{\ NaCa}$	-	Posición de la Barrera de Energía de la teoría de cambio Erying controlando la dependencia de voltaje de $i_{.NaCa}$
$d_{\ NaCa}$	mM^{-4}	Constante de denominador para $i_{.NaCa}$
$K_{m,\,Na}$ $K_{m,\,K}$	mM	Constantes de disociación para activación de i_p por Na y K

ANEXO B

VALORES INICIALES Y PARAMETROS PARA EL MODELO DEL NODO SA

Tabla B.1. Valores Iniciales del Modelo Original del Nodo Sinusal[*].

	Modelo Celular de la Periferia del Nodo SA	**Modelo Celular del Centro del Nodo SA**
V, mV	-64.35	-51.44
m	0.124	0.124
h_1	0.595	0.595
h_2	$5.250*10^{-2}$	$5.250*10^{-2}$
d_L	$8.450*10^{-4}$	$5.912*10^{-2}$
f_L	0.987	0.825
d_T	$1.725*10^{-2}$	0.106
f_T	0.436	0.119
y	$5.280*10^{-2}$	$3.775*10^{-2}$
r	$1.970*10^{-2}$	$3.924*10^{-2}$
q	0.663	0.358
x	$7.670*10^{-2}$	$5.700*10^{-2}$
$p_{a,f}$	0.400	0.470
$p_{a,s}$	0.327	0.637
p_i	0.991	0.965

[*] Tabla tomada de [DASIERRA].

Tabla B.2.Valores de Parámetros en el Modelo del Nodo Sinusal.[*]

	Células Periféricas del nodo SA		**Células Centrales del nodo SA**		**Relación**
	Valor Absoluto	Valor Normalizado	Valor Absoluto	Valor Normalizado	
C_m	65 pF	--	20 pF	--	--
d_{NaCa}	0.0001	--	0.0001	--	--
$E_{Ca,L}$	46.4 mV	--	46.4 mV	--	--
$E_{Ca,T}$	45 mV	--	45 mV	--	--
g_{Na}	$1.2*10^{-6}$ μS	$1.85*10^{-8}$ μS/pF	0 S	0 μS/pF	∞
$g_{Ca,\ L}$	$6.59*10^{-2}$ μS	$1*10^{-3}$ μS/pF	$0.58*10^{-2}$ μS	$2.90*10^{-4}$ μS/pF	3.45
$g_{Ca,\ T}$	$1.39*10^{-2}$ μS	$2.14*10^{-4}$ μS/pF	$0.43*10^{-2}$ μS	$2.14*10^{-4}$ μS/pF	1
g_{to}	$36.49*10^{-3}$ μS	$5.6*10^{-4}$ μS/pF	$4.91*10^{-3}$ μS	$2.5*10^{-4}$ μS/pF	2.33
g_{sus}	$1.14*10^{-2}$ μS	$1.8*10^{-4}$ μS/pF	$6.65*10^{-5}$ μS	$3.3*10^{-6}$ μS/pF	54.55
$g_{K,\ r}$	$1.60*10^{-2}$ μS	$2.46*10^{-4}$ μS/pF	$7.97*10^{-4}$ μS	$3.99*10^{-5}$ μS/pF	6.17
$g_{K,\ s}$	$1.04*10^{-2}$ μS	$1.6*10^{-4}$ μS/pF	$5.18*10^{-4}$ μS	$2.59*10^{-5}$ μS/pF	6.17
$g_{f,\ Na}$	$0.69*10^{-2}$ μS	$1.05*10^{-4}$ μS/pF	$5.48*10^{-4}$ μS	$0.27*10^{-4}$ μS/pF	3.93
$g_{f,\ K}$	$0.69*10^{-2}$ μS	$1.05*10^{-4}$ μS/pF	$5.48*10^{-4}$ μS	$0.27*10^{-4}$ μS/pF	3.93
$g_{b,\ Na}$	$1.89*10^{-4}$ μS	$2.9*10^{-6}$ μS/pF	$5.8*10^{-5}$ μS	$2.91*10^{-6}$ μS/pF	1
$g_{b,\ Ca}$	$4.3*10^{-5}$ μS	$6.61*10^{-7}$ μS/pF	$1.32*10^{-5}$ μS	$6.62*10^{-7}$ μS/pF	1
$g_{b,\ K}$	$8.19*10^{-5}$ μS	$1.3*10^{-6}$ μS/pF	$2.52*10^{-5}$ μS	$1.3*10^{-6}$ μS/pF	1
$\overline{i_p}$	0.16 nA	$2.46*10^{-3}$ nA	$4.78*10^{-2}$ nA	$2.46*10^{-3}$ nA	1
k_{NaCa}	$0.88*10^{-5}$ nA	$1.36*10^{-7}$ nA	$0.27*10^{-5}$ nA	$1.36*10^{-7}$ nA	1
$[Na^+]_o$	140 mM	--	140 mM	--	--
$[Na^+]_i$	8 mM	--	8 mM	--	--
$[Ca^{++}]_o$	2 mM	--	2 mM	--	--
$[Ca^{++}]_i$	0.0001 mM	--	0.0001 mM	--	--
$[K^+]_o$	5.4 mM	--	5.4 mM	--	--
$[K^+]_i$	140 mM	--	140 mM	--	--
$K_{m,\ K}$	0.621	--	0.621	--	--
$K_{m,\ Na}$	5.64	--	5.64	--	--
γ_{NaCa}	0.5	--	0.5	--	--

[*] Tabla tomada de [DASIERRA]

ANEXO C

CONFIGURACION DE UN CLUSTER NO DEDICADO

REQUISITOS

- Conocimientos básicos en Linux, OpenSSH, redes y lenguaje C.
- Red de PC's.
- Sistema operativo Linux instalado previamente en cada PC. Se debe contar con los compiladores de C y C++.
- Tener habilitados los servicios de SSH y RSH
- Tener instalados nfs-utils y portmap.
- Contar con el archivo de instalación LAM-MPI, que se puede descargar de http://www.lam-mpi.org

INTRODUCCION

El contenido de este anexo describe las acciones a realizar para la creación de un cluster de máquinas Linux. No se supone que cada uno de los nodos sea una máquina dedicada; de hecho cada máquina del cluster continuará realizando las tareas asignadas además de pertenecer al cluster. Este cluster permite la

ejecución de programas paralelos; esto significa que su codificación contiene instrucciones especiales para ser ejecutados de forma paralela. Existen diferentes protocolos para la creación de este tipo de entornos por ejemplo PVM o MPI. El cluster que se describe utiliza el protocolo MPI a través de la implementación LAM-MPI.

El cluster descrito consiste en un conjunto de máquinas con sistema operativo Linux a las que se les define un usuario (lam). El directorio de trabajo de este usuario es un directorio remoto que contiene los archivos binarios necesarios para la activación del nodo. No es necesaria la ejecución de un daemon de sistema. Cuando el cluster se activa, el nodo maestro se pone en contacto con cada uno de los nodos clientes a través del usuario lam y ejecuta el daemon lamd. Mientras está activo se pueden ejecutar los programas paralelos dentro del entorno LAM. Cuando el cluster ya no es necesario, se cierra el daemon lamd y no queda ningún proceso ni usuario del cluster en ejecución.

Todos los pasos descritos a continuación se deben realizar sobre cada maquina en modo consola con el usuario root, ya que es la manera mas general e independiente de la distribución Linux instalada.

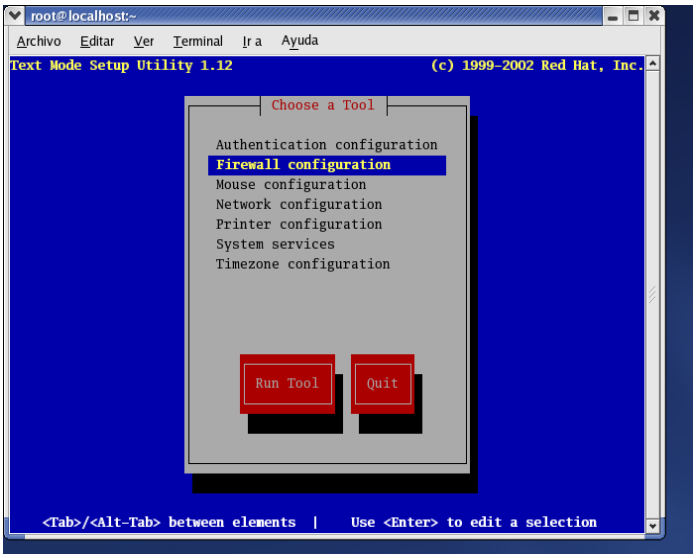

Figura C.1 Setup del sistema operativo Linux

CONFIGURACION DE LA RED

Lo primero que se debe garantizar es la correcta comunicación entre las maquinas que conformarán el cluster. Para tal fin se debe configurar cada una de ellas, quitando las barreras de seguridad y modificando los archivos de red para que los computadores se reconozcan entre si. Los pasos aquí descritos suponen la no existencia de un DNS, si la red que utilizara para el cluster posee un DNS configurado puede omitir estos pasos ya que la red ya se encuentra configurada.

Ingresar al Setup: El setup es una utilidad que permite entre otras cosas configurar el Firewall y el dispositivo de red, siendo estos aspectos los que interesan para la configuración de la red. En una consola de comando digite ***setup***, lo cual abrirá una interfaz como la de la figura C.1. La interfaz no permite el uso del mouse, por lo tanto debe desplazarse usando las flechas del teclado, la tecla TAB y ENTER.

Deshabilitar el Firewall: Una vez en el setup ingrese a la sección ***firewall configuration***. Elija la opción ***No firewall*** y luego ***ok***. Con este paso se esta permitiendo acceso libre al equipo.

Configuración de la red interna: Ingrese a la sección del setup ***Network configuration*** en donde se le pedirán los datos apropiados para la creación de una red interna tales como: dirección IP, mascara de subred, puerta de enlace y servidor de nombres de dominio primario. Para el resto de esta guía se tendrá una red de las siguientes característica:

RED IP: 192.168.45.0

Mascara de subred: 255.255.255.0

Puerta de enlace: 192.168.45.254

Servidor DNS: ---------. Se supone que no hay.

Con esto la dirección IP de cada computador de la red será: 192.168.45.xx donde el xx varía para cada computador. Sin embargo esta configuración es libre de ser cambiada de acuerdo a la disposición del usuario de esta guía.

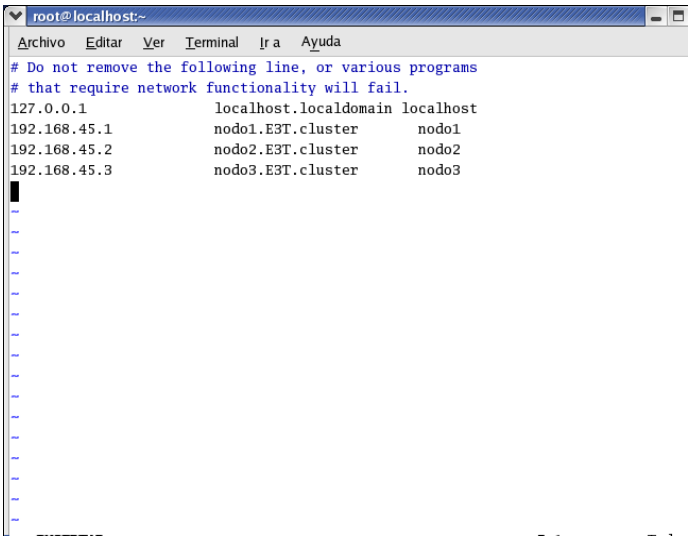

Figura C.2. Fichero *hosts* para tres nodos

Modificar Archivos de configuración de red: Una vez ha salido del setup (opción ***Quit***) la interfaz de red estará configurada, sin embargo se deben modificar algunos archivos para que los nodos se reconozcan entre si.

- **Fichero *letc/hosts***: El primer archivo a modificar es *hosts* en el cual se encuentran las definiciones de todos los nodos de la red asociados con un nombre de dominio y un alias. Para ingresar al fichero *hosts* en la consola de comandos teclee: ***vi /etc/hosts***

  Una vez este dentro del archivo *hosts* ingrese la dirección IP de cada nodo (anteriormente digitada) con su respectivo nombre.dominio y un alias (cualquiera)[*]. En la figura C.2 se presenta un ejemplo del archivo hosts para tres nodos.

[*]Recuerde que para poder editar el fichero debe presionar la tecla i, una vez modificado presione Esc y luego digite :wq para guardar los cambios y salir

- **Fichero /etc/*hosts.equiv***: Este fichero contiene una lista de computadores y usuarios a los cuales se les a concedido acceso "confiable" a su sistema mediante r-ordenes (ordenes remotas, **rlogin**, **rsh** o **rcp**) sin proporcionar una contraseña. El fichero usa el siguiente formato:

  [+/-] [Computador] [+/-] [usuario]

  Computador: es el nombre de un computador que es lógicamente equivalente al computador local. A los usuarios conectados a ese computador se les permite acceder a las cuentas de los usuarios con igual nombre en el computador local sin proporcionar una contraseña. ***Computador*** puede ir (opcionalmente) precedido por un signo más (+) o un signo meno (-) para permitir o negar el acceso de los computadores al sistema. Los usuarios procedentes de ese ***computador*** siempre deberán proporcionar una contraseña. Por razones de seguridad, siempre debe usar el "nombre de dominio totalmente cualificado" (FQDN) del computador y no su nombre corto

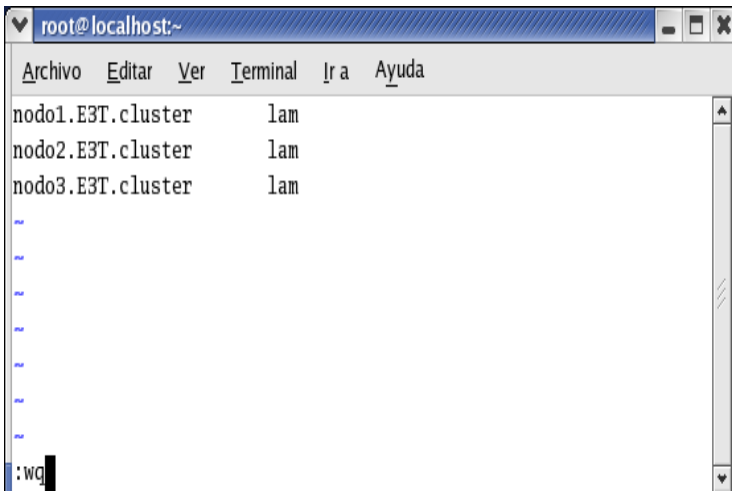

Figura C.3. Fichero /etc/hosts.equiv para tres nodos

La entrada ***usuario*** concede a un usuario específico el acceso a todas las cuentas de usuario (excepto la del root) sin proporcionar una contraseña. Eso significa que el usuario NO está restringido a las cuentas que tengan el mismo nombre. ***Usuario*** puede ir (opcionalmente) precedido por un signo más (+) o un signo meno (-) para permitir o negar el acceso a un usuario específico.

Para ingresar al fichero teclee: ***vi /etc/hosts.equiv*** en la consola de comandos. En la figura C.3 se muestra el fichero *hosts.equiv* para el ejemplo de tres nodos.

- **Ficheros /etc/*hosts.allow y /etc/hosts.deny***: Estos ficheros son consultados por el software de control de acceso en la maquina Linux. Para esta configuración bastara con garantizar que ambos ficheros este vacios[*].

CREACION DEL USUARIO LAM EN CADA NODO

El usuario que utilizara el cluster será el usuario lam. Las características del usuario serán las siguientes:

Nombre: lam

Home: /home/lam

Grupo: cluster (puede ser otro)

[*] Recuerde que las líneas que inician con el símbolo # son comentarios.

shell: bash

password: lo decide el administrador de la maquina y es local al nodo.

Descripción: usuario cluster

Antes de crear el usuario debe comprobar que exista el grupo donde será asignado. Para crear el grupo y el usuario con las características descritas se ejecuta como root:

groupadd *cluster* *# crea el grupo cluster (puede ser otro)*

useradd -c "usuario cluster" -d /home/lam -g cluster -n -s /bin/bash *lam #crea el usuario lam*

passwd *lam* *# crea un password para el usuario lam, no es necesario que sea conocido por el resto del cluster.*

COMPARTIR EL SISTEMA DE FICHEROS DE LAM

Es de interés que el directorio de trabajo del usuario *lam* sea único en todo el cluster para poder garantizar lo siguiente:

Ejecutables LAM-MPI únicos. La versión de los ejecutables debe ser la misma para que el cluster funcione correctamente. Además permitirá la actualización centralizada de la herramienta.

Simplificación en la distribución de programas a ejecutar. Dentro de la carpeta compartida existe un directorio para albergar los programas que ejecutará el cluster, lo cual evita el distribuir el programa antes de

ejecutarlo.

Distribución automática de las claves públicas para SSH.

Para compartir el sistema de ficheros se usara NFS (Network File System). Primero se **verifica si esta instalados nfs-utils y portmap**, para ello se ejecuta en la consola:

rpm -q nfs-utils portmap

Lo cual debe regresar algo como:

nfs-utils-*** // los *** dependerán de la versión Linux instalada.

portmap-***

En caso de que falte alguno de los paquetes, inserte el cd de instalación y ejecute lo siguiente:

mount /dev/cdrom /mnt/cdrom

rpm -Uvh /mnt/cdrom/Distribucion_Linux/PRMS/paquete faltante

Una vez hecho esto se disponen de los daemons necesarios para utilizar nfs.

Configurar el servidor de ficheros: Esto se debe hacer solo desde un nodo el cual será establecido como el servidor de ficheros para los demás. El archivo ***/etc/exports*** controla cuáles sistemas de archivos son exportados a las máquinas remotas y especifica opciones. Las líneas en blanco son ignoradas, se pueden comentar líneas con el símbolo # y las líneas largas pueden ser divididas con una barra invertida (\). Cada sistema de archivos exportado debe tener su propia línea y cualquier lista de hosts autorizadas colocada después de un sistema de

archivos exportado, debe estar separada por un espacio. Las opciones para cada uno de los hosts deben ser colocadas entre paréntesis directamente detrás del identificador del host, sin ningún espacio de separación entre el host y el primer paréntesis.

Una línea para un sistema de archivos exportado tiene la estructura siguiente:

<Directorio a exportar> <host1>(<opciones>) <hostN>(<opciones>)

Para este caso el archivo *exports* para los nodos del cluster tendrá la siguiente línea:

/home/lam *.E3T.cluster(rw)

Note que el uso del comodín * en el nombre del host permite la simplificación para todos los nodos del cluster. Además se da la opción (rw) que permiten montar el fichero con permiso de lectura/escritura. Para mas información refiérase al manual de exports (***man exports***) en la consola de comandos.

Una vez editado el fichero */etc/exports* se deben reiniciar los servicios de nfs, por lo tanto ejecute el siguiente comando: ***/sbin/service nfs restart***

Por ultimo a fin de asegurar que la próxima vez que se encienda el equipo este habilitado el servicio nfs se debe ejecutar lo siguiente: ***/sbin/chkconfig –level 345 nfs on***

Configurar los clientes: Una vez establecido el nodo servidor se debe configurar el resto de nodos para que monten el sistema de ficheros que será compartido por el cluster. Esto se logra simplemente editando el fichero ***/etc/fstab*** agregando la siguiente línea:

nodo1.E3T.cluster:/home/lam /home/lam nfs exec,rw 0 0

Aquí se supone que nodo1 es el servidor de ficheros, y monta el sistema de

ficheros en modo lectura/escritura (rw) permitiendo la ejecución de binarios (exec). Con esto cada vez que se encienda la maquina se montara automáticamente los archivos para el usuario lam, sin embargo debe estar primero encendido el servidor de ficheros.

ACCESO LIBRE A LOS NODOS: GESTION DE CLAVES

Conexión de prueba: Una vez configurarla la red ya se puede acceder a cualquier nodo del cluster de forma remota. Se supone que todas las máquinas del cluster tienen el cliente y servidor openSSH. Para probar que se puede realizar la conexión vía SSH intente conectarse como usuario lam a otro nodo del cluster, para ello ejecute los siguientes comandos:

su lam # Para ingresar al usuario lam

ssh 192.168.45.2 # Por ejemplo para conectarse al nodo2

Al tratar de hacer conexión por primera vez el sistema preguntara si esta seguro que desea conectarse, por lo tanto se debe responder yes (*si)* y dar ENTER. Después pedirá el password del equipo asociado al usuario que se trata de conectar (en este caso lam), por lo tanto hay que ingresarlo y de nuevo ENTER.

Si la conexión fue exitosa debe mostrar el prom del otro equipo, lo cual significa que se puede manipular el nodo accedido remotamente como si se estuviera sentado en ese equipo, pudiéndose reiniciar, listar ficheros y todo aquello a lo cual el usuario tenga permiso. Para probar que se esta en el otro nodo ejecute un comando como ***eject*** que sacara la bandeja de cd del equipo donde esta conectado remotamente y luego para evitar pararse a introducir la bandeja de nuevo ejecute:

eject -t.

Finalmente, para terminar una conexión remota se utiliza el comando ***exit***.

Evitar password: De lo anterior es claro que cada vez que se desee acceder a un nodo remoto se debe ingresar un password para comprobar la identidad, lo cual resulta demasiado engorroso para la ejecución de un programa paralelo, ya que cada vez que se quiera lanzar procesos en los demás nodos se debe ingresar el respectivo password, es decir, si lanza un programa en diez nodos se tendrá que ingresar nueve passwords, ahora si fuesen cien o tal vez mil nodos, eso resultaría demasiado molesto y engorroso. Para evitar este problema lo que comúnmente se hace es crear una llave de acceso y dársela a quien se desee permitir el ingreso. Esto supone que cada nodo cree una llave y le entregue una copia a los demás, sin embargo, como en este caso el directorio de trabajo del usuario lam es compartido, todo este trabajo se puede evitar ya que con solo crear una llave todos los nodos tendrán acceso a esta.

Las siguientes instrucciones las debe realizar el administrador del cluster como usuario lam una sola vez. Para generar las claves ingrese en el nodo que hace de servidor de ficheros como usuario lam y compruebe que existe el directorio ***$HOME/.ssh.*** En caso de no existir, conéctese vía ssh usando password a otra máquina, con lo cual se creará el directorio automáticamente con los permisos adecuados. Para tal fin ejecute los siguientes comandos:

cd *$HOME/.ssh* #compruebar si existe el directorio e ingresar en este

ssh *lam@nodo2.E3T.cluster* #ingresa a otra maquina, en caso que no exista $HOME/.ssh

exit #salir de la otra maquina.

ssh-keygen -t rsa #crear las llaves de acceso.

Con ***ssh-keygen*** se crean dos llaves, una privada (de uso propio) y otra publica (para dársela a los demás), la opción ***-t rsa*** se utiliza para especificar el tipo de encriptamiento de la llave, que en este caso es el algoritmo rsa. Al dar este comando le preguntara donde se guardaran las llaves, el directorio por defecto es ***$HOME/.ssh,*** así que simplemente hay que dar ENTER. Además pide un passphrase (frase contraseña), pero como se desea libre acceso no se debe ingresar nada.

Una vez realizado esto, compruebe que existen las llaves en el directorio $HOME/.ssh, para ello liste los archivos con el siguiente comando:

ls *$HOME/.ssh*

Con lo cual debe listar algo como:

id_rsa id_rsa.pub known_hosts

id_rsa e *id_rsa.pub* son la clave privada y publica, respectivamente. Por lo tanto *id_rsa.pub* debe ser conocida por todos los nodos, para tal efecto, hay que mover la clave pública al fichero ***$HOME/.ssh/authorized_keys***. Ejecutando la siguiente instrucción:

cat id_rsa.pub >> $HOME/.ssh/authorized_keys

Finalmente, se comprueba si se puede acceder vía ssh al usuario lam sin necesidad de claves. Conéctese a otro nodo, si no le solicita password significa que se abrió una sesión ssh utilizando las claves publica y privada; en caso contrario, algo no funciona correctamente. (Revise los pasos anteriores)

INSTALACION Y CONFIGURACION DEL MIDDLEWARE: LAM-MPI

Cuando se ejecuta un programa en paralelo, es necesario de algo que dirija las acciones que se realizaran dentro del cluster, ese algo es el middleware, el cual actúa como una capa sobre el sistema operativo que ordena la actividades asociadas con los programas, para este caso es LAM-MPI que es una interfaz del protocolo de paso de mensajes para el correcto funcionamiento y comunicación de las maquinas del cluster.

La instalación de LAM se realizará de forma centralizada, en el directorio $HOME del usuario lam del servidor de ficheros. De esta forma se evita el actualizar la instalación de los binarios lam cada vez que haya un cambio de versión. Además la instalación aquí provista es elemental, para una instalación con características mas avanzadas por favor consulte el manual de instalación que puede descargar de la página principal de LAM[*].

Descomprimir: Se supondrá que la versión descargada de LAM[**] es **lam-7.1.1.tar.gz** y que se encuentra en el directorio $HOME del usuario lam. Por lo general cuando se descargan programas Linux, estos están compresos y corresponden a los ejecutables binarios, por lo tanto hay que descomprimirlos y compilarlos. Para descomprimir el fichero de los ejecutables de lam ejecute las siguientes instrucciones:

cd $HOME #ingresar al directorio home de lam.

[*] http://www.lam-mpi.org

[*] Esta es la versión más reciente en el momento de elaborar esta guía y además es la utilizada para el desarrollo del trabajo de grado.

gunzip -c lam-7.1.1.tar.gz **| tar xf -** # descomprimir y desarchivar los ejecutables.

rm lam-7.1.1.tar.gz #borrar lo que no necesito

Configurar y Compilar: una vez descompreso el archivo se debe crear un directorio lam-7.1.1, se debe ingresar y compilar los ejecutables de lam para poder realizar la instalación. Para tal fin, ejecute lo siguiente:

cd lam-7.1.1 #ingresa al directorio

./configure –prefix=/home/lam #configuración de lam[@]

make #compilar

make install #instalar

Las opciones de configuración de lam son muy variadas[#] y dependerán de las características especiales del cluster, sin embargo con solo especificar el directorio de instalación se configurara con las opciones estándar, las cuales se acomodan muy bien en este caso. Una vez realizada la configuración se procede a construir y compilar (make) los binarios LAM y las librerías incluidas con la distribución, para luego pasar a instalar (make install). Cada uno de esto pasos tomara un largo tiempo y debe arrojar muchas salidas, si termina todo satisfactoriamente no se mostrara ningún mensaje de error.

[@] Para poder utilizar la toolbox de MATLAB (MPITB), es necesario especificar otras opciones de compilación, las cuales se especifican en el anexo E.

[#] Para mas información consulte el manual de instalación en www.lam-mpi.org

Modificar $HOME/.bashrc: Como la instalación de LAM no se hizo en el directorio por defecto de Linux (/usr/local/), se debe indicar al shell donde encontrar los comandos relacionados con LAM. El fichero /.bashrc es propio de cada usuario y allí se definen una serie opciones para el shell del usuario, por lo tanto se tendrá que editar el fichero $HOME/.bashrc para el usuario lam especificando donde encontrar los respectivos binarios, las paginas asociadas a man y una serie de variables que el entorno LAM necesita para su correcto funcionamiento. Las siguientes líneas deben ser añadidas al fichero $HOME/.bashrc [##]:

```
PATH=$HOME/bin:$PATH
MANPATH=$MANPATH:$HOME/man
LAMHOME=/home/lam
LAMRSH="ssh -x "
export PATH MANPATH LAMHOME LAMRSH
export LD_LIBRARY_PATH=$HOME/lib:$LD_LIBRARY_PATH
export LIBRARY_PATH=$HOME/lib:$LIBRARY_PATH
export C_INCLUDE_PATH=$HOME/include:$C_INCLUDE_PATH
export
CPLUS_INCLUDE_PATH=$HOME/include:CPLUS_INCLUDE_PATH
```

Configuración del cluster: La definición del cluster para LAM se encuentra en el fichero ***/home/lam/etc/lam-bhost.def***. Este fichero debe contener todos los nodos del cluster, usuario de conexión (en este caso, lam) y número de

[##] Recuerde: para editar este fichero puede utilizar el comando: vi $HOME/.bashrc

CPU's. El contenido para este caso debe ser parecido al siguiente:

```
#
============================================================
=====
#                          lam-bhost.def                          file
#          contains          node          list          of          cluster
#        for        more        info        execute:        man        bhost
#
============================================================
=====
nodo1.E3T.cluster cpu=2  user=lam
nodo2.E3T.cluster cpu=1  user=lam
nodo3.E3T.cluster cpu=1  user=lam
```

El parámetro cpu indica a LAM el número de procesos que debe lanzar en cada máquina. Es el sistema operativo quien de forma automática distribuye los diferentes procesos en cada CPU's disponible. Normalmente se indica el número de CPU's de la máquina para aprovechar todos los recursos existentes. Es recomendable en el nodo maestro poner un CPU's mas del que tiene ya que al codificar un programa paralelo es usual que el nodo maestro se encargue de repartir las tareas a realizar y recoger los resultados; entonces, el nivel de carga es inferior al resto de nodos. Por lo tanto, en el nodo maestro existirán dos procesos corriendo en una misma CPU, el maestro (con poca carga de CPU) y el esclavo.

Con esto, el cluster debe quedar listo para su utilización. Recuerde que esta es una guía muy general y por lo tanto las especificaciones propias de algunos equipos o configuraciones Linux pueden no concordar con las especificaciones básicas aquí descritas, por lo tanto se sugiere consultar la paginas de referencia del Linux y lam respectivamente.

COMANDOS BASICOS DEL CLUSTER

Una sesión de trabajo estándar en el cluster comprende la ejecución por parte del usuario lam de los siguientes comandos:

recon: Comando para comprobar si el cluster está preparado para ser arrancado. Utiliza las variables de entorno LAMHOME para determinar donde se encuentra la instalación de LAM y la variable LAMRSH para indicar que la conexión a los nodos se realice usando SSH.

lamboot: Comando usado para arrancar el cluster. La configuración del cluster la obtiene del fichero $HOME/etc/lam-bhost.def. Utiliza las variables de entorno LAMHOME y la variable LAMRSH para determinar donde se encuentra la instalación de LAM y para indicar que la conexión a los nodos se realice usando SSH, respectivamente.

mpicc | mpiCC | mpif77 : Comandos utilizados para compilar los programas a ejecutar en el cluster.

mpirun: Comando para lanzar los programas en el cluster.

lamexec [C|N] command: Comando similar a mpirun para lanzar programas estándar unix (programas no MPI, por ejemplo: date). La opción C, indica que debe ejecutarse en todas las CPU's; la opción N indica que solamente debe ejecutarse una instancia en cada nodo.

lamclean: Este comando solamente debe utilizarse cuando un programa del cluster finaliza de forma incorrecta o inesperada y mantiene recursos del

cluster alocados. Detiene todos los programas que se estén ejecutando en aquel momento en el cluster.

mpitask | laminfo | lamnodes | tping: Comandos informativos. mpitask lista el conjunto de tareas que está ejecutando el cluster en este momento. Es el equivalente al comando unix *ps*. laminfo proporciona información del entorno LAM. Tiene multitud de opciones, para más información consulte las páginas de man. lamnodes lista el conjunto de nodos que forman el cluster. Proporciona una información muy parecida al contenido del fichero lam-bhost.def. tping comprueba que todos los nodos están activados.

lamhalt: Comando utilizado para deshabilitar el cluster. En caso de retorno de error o falta de respuesta utilice *wipe -v $HOME/etc/lam-bhost.def.*

CORRIENDO UN EJEMPLO

Para finalizar se muestra como ejecutar un programa con los comandos antes descritos. La distribución LAM viene con una serie de ejemplos entre los cuales se encuentra 'hello' que corresponde al típico "HOLA MUNDO" que todas las guías de programación manejan. Los siguientes pasos se deben realizar para ejecutar el ejemplo y en general para ejecutar cualquier programa.

- Iniciar el cluster:

  lamboot

- Ingresar al directorio donde esta el programa:

  cd $HOME/lam-7.1.1/examples/hello

- Compilar el programa. Se utilizará el ejemplo en código c. sin embargo también

vienen en C++ y Fortran.

mpicc hello.c -o hello [*]

- Ejecuta el programa de ejemplo hello en el cluster.

mpirun C hello

La opción C, indica que debe ejecutarse en todas las CPU's

- Finalizar el cluster si no se utilizara más.

lamhalt

[*] la opción -o indica el nombre del ejecutable, ya que por defecto es a.out

ANEXO D

INTRODUCCION AL ESTÁNDAR MPI
(*MESSAGE PASSING INTERFACE*)

INTRODUCCION

MPI ("Message Passing Interface", Interfaz de Paso de Mensajes) es un estándar que define una infraestructura común y una semántica específica de interfaz de comunicación contenidas en una librería de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores. Dicha librería puede ser utilizada por una amplia variedad de usuarios para implementar programas que utilicen paso de mensajes. Desde la finalización de su primera versión en Junio de 1994, MPI ha sido ampliamente aceptado y usado. Por ser un estándar los productores de software de procesamiento paralelo pueden implementar su propia versión de MPI siguiendo sus especificaciones, con lo cual múltiples implementaciones de MPI cambiarían solamente en factores tales como la eficiencia de su implementación y herramientas de desarrollo, pero no en sus principios básicos.

Existen actualmente diferentes implementaciones de MPI, algunas son comerciales y otras son de dominio público, además están disponibles para múltiples arquitecturas. Gracias a ello MPI ha alcanzado uno de sus principales objetivos: darle credibilidad al procesamiento paralelo. En el desarrollo del presente trabajo de grado se trabajó con LAM (Local Area Multicomputer) que es un ambiente de desarrollo y programación de MPI orientado a arquitecturas

heterogéneas de ordenadores distribuidos en una red local, con LAM tanto un cluster dedicado como un una simple red local de ordenadores puede actuar como un ordenador multiprocesador.

La finalidad de MPI, de manera concisa, es desarrollar un estándar para escribir programas de paso de mensajes que sea ampliamente utilizado. Para ello la interfaz debe establecer un estándar práctico, portable, eficiente, escalable, formal y flexible. Según esto MPI cumple con los siguientes objetivos:

- Diseño de una interfaz para la programación de aplicaciones.
- Interfaz diseñada para que pueda ser implementada en la mayoría de las plataformas, sin necesidad de cambios importantes en la comunicación o el software del sistema.
- Permite implementaciones que puedan ser utilizadas en entornos heterogéneos.
- Permite una comunicación eficiente.
- Asume una interfaz de comunicación fiable: el usuario no debe preocuparse por los fallos en la comunicación.
- La semántica de la interfaz es independiente del lenguaje.
- La interfaz permite la utilización de hilos.
- Proporciona extensiones que añaden más flexibilidad.

CONCEPTOS BASICOS

MPI está especialmente diseñado para desarrollar aplicaciones SPMD. Por lo general, al arrancar una aplicación se lanzan en paralelo N copias del mismo programa (procesos). Estos procesos no avanzan sincronizados instrucción a instrucción sino que

la sincronización tiene que ser explícita. Los procesos tienen un espacio de memoria completamente separado. El intercambio de información, así como la sincronización, se hacen mediante paso de mensajes. En MPI los procesos implicados en la ejecución de un programa paralelo se identifican por una secuencia de enteros no negativos. Si hay p procesos ejecutando un programa, éstos tendrán los identificadores 0, 1, 2,…, p-1.

Todos los programas MPI comparten una serie de características, la inicialización y finalización del entorno de ejecución se llevan a cabo mediante funciones así como también se dispone de funciones de comunicación punto a punto (que involucran sólo a dos procesos), y de funciones u operaciones colectivas (que involucran a múltiples procesos). Los procesos pueden agruparse y formar comunicadores, lo que permite una definición del ámbito de las operaciones colectivas, así como un diseño modular.

Es así como la forma básica de todo programa MPI en lenguaje C tendrá la siguiente estructura:

#include "mpi.h"
...
main(int argc, char argv**){
    ...
    Iniciar Ambiente Paralelo → **MPI_Init**.
    ...
    Desarrollo de la parte del programa paralelo
    …
    Finalizar Ambiente Paralelo → **MPI_Finalize**.
    ...
    }

El fichero **'mpi.h'** contiene las definiciones, macros y prototipos de función necesarios para compilar los programas MPI.

FUNCIONES BASICAS

MPI_Init	Inicializar el entorno de MPI
MPI_Finalize	Terminar el entorno de MPI
MPI_Comm_size	Determina el tamaño del grupo asociado con un comunicador
MPI_Comm_rank	Determina el rango del proceso actual dentro del comunicador
MPI_Send	Envía datos en un mensaje
MPI_Recv	Recibe datos de un mensaje

Tabla D.1 Funciones Básicas de MPI

Típicamente, las implementaciones basadas en MPI tienen una gran variedad de funciones, sin embargo para escribir un programa paralelo básico se pueden emplear solo 6, las cuales se presentan en la tabla D.1 con su respectivo significado y su respectiva sintaxis se presenta a continuación.

- **MPI_Init**

MPI_Init(int ****argc*, char *****argv***)**

arg Puntero al numero de argumentos.

argv Puntero al vector de argumentos.

Antes de poder hacer una llamada a cualquier otra función MPI se debe hacer una llamada a **MPI_Init()**; esta función sólo debe ser llamada una vez. Sus argumentos son punteros a los parámetros de la función **main()**, **argc** y **argv**. Dicha función permite al sistema hacer todas las configuraciones necesarias para que la librería MPI pueda ser usada.

- **MPI_Finalize**

int MPI_Finalize()

Después de que el programa haya acabado de utilizar la librería MPI se debe hacer una llamada a **MPI_Finalize()**. Esta función limpia todos los trabajos no finalizados dejados por MPI (por ejemplo, envíos pendientes que no hayan sido completados, etc.).

- **MPI_Comm_size**

int MPI_Comm_size(MPI_Comm *comm*, int **size*)

Entrada:

***comm**:* comunicador (handle)

Salida:

size: puntero a un número entero que recoge el número de procesos en el grupo de ***comm***

Esencialmente un comunicador es una colección de procesos que pueden enviarse mensajes entre sí. Normalmente para diseñar programas básicos el único comunicador que necesitaremos será **MPI_COMM_WORLD**. Está predefinido en MPI y consiste en todos los procesos que se ejecutan cuando el programa comienza.

- **MPI_Comm_rank**

int MPI_Comm_rank (MPI_Comm *comm*, int **rank*)

Entrada:

***comm**:* comunicador (handle)

Salida:

rank: puntero a un número entero que recoge el rango del proceso actual en el grupo de ***comm.***

Los rangos se enumeran del 0… N-1 si N es el número de procesos (***size*** en **MPI_Comm_size**).

TIPO DE DATOS MPI	TIPO DE DATOS C
MPI_CHAR	Signed char
MPI_SHORT	Signed short int
MPI_INT	Signed int
MPI_LONG	Signed long int
MPI_UNSIGNED_CHAR	Unsigned char
MPI_UNSIGNED_SHORT	Unsigned short int
MPI_UNSIGNED	Unsigned int
MPI_UNSIGNED_LONG	Unsigned long int
MPI_FLOAT	Float
MPI_DOUBLE	Double
MPI_LONG_DOUBLE	Long double
MPI_BYTE	
MPI_PACKED	

Tabla D.2. Equivalencia tipos de datos MPI y C

El paso de mensajes en la forma mas básica se lleva a cabo por las funciones **MPI_Send()** y **MPI_Recv()**. La primera función envía un mensaje a un proceso determinado. La segunda recibe un mensaje de un proceso. Para que el mensaje sea comunicado con éxito, el sistema debe adjuntar alguna información a los datos que el programa intenta transmitir. Esta información adicional conforma el *entorno* del mensaje. En MPI el entorno contiene la siguiente información:

1. El identificador del proceso receptor del mensaje.
2. El identificador del proceso emisor del mensaje.
3. Una etiqueta.
4. Un comunicador.

Estos datos pueden ser usados por el proceso receptor para distinguir entre los mensajes entrantes. El *origen* puede ser usado para distinguir mensajes recibidos por distintos procesos.

- **MPI_Send**

int MPI_Send(void ****buf*, **int *count*, **MPI_Datatype** *datatype*, **int** *dest*, **int** *tag*, **MPI_Comm** *comm* **)**

Entrada:

buf*:* Dirección del primer elemento del buffer.

count: Numero de elementos en el buffer.

datatype*:* Tipo de datos de cada elemento en el buffer

dest*:* Rango del proceso destinatario.

tag*:* Identificador del mensaje

comm*:* comunicador

El tipo de datos de cada elemento puede ser uno de los especificados en la tabla D.2.

- **MPI_Recv**

int MPI_Recv(void ****buf*, **int *count*, **MPI_Datatype** *datatype*, **int** *source*, **int** *tag*, **MPI_Comm** *comm*, **MPI_Status ****status* **)**

Salida:

buf*:* Dirección del primer elemento del buffer donde recibe.

status*:* Estructura que indica el estado.

Entradas:

count: Numero máximo de elementos para el buffer.

datatype*:* Tipo de datos de cada elemento en buffer

source: Rango del proceso remitente.

***tag**:* Identificador del mensaje

***comm**:* comunicador

Se utilizan los mismos tipos de datos como en MPI_Send. El parámetro count indica el número máximo de elementos que pueden recibirse.

OTRAS FUNCIONES UTILES

- **MPI_Barrier:** Bloquea hasta que todos los procesos han alcanzado esta rutina.

  int MPI_Barrier (MPI_Comm *comm*)

  Entrada:

  ***comm**:* comunicador

Esta función es útil para asegurar que todos los procesos se encuentran en un cierto estado antes de seguir en el cálculo

- **MPI_Wtime:** devuelve los segundos desde un momento dado no especificado.

  double MPI_Wtime()

  Salida: tiempo en segundos desde un instante arbitrario

Esta función puede emplearse para medir el tiempo de cálculo transcurrido:

t1=MPI_Wtime(); ... cálculos...; t2=MPI_Wtime(); printf("%e\n",t2-t1);

- **MPI_Get_processor_name:** devuelve el nombre del procesador actual

  int MPI_Get_processor_name(char **name*, int **longname*)

Salidas:

name: cadena de caracteres que recibe el nombre del nodo.

longname: longitud del nombre devuelto

El parámetro **name** es una cadena (vector de caracteres) cuyo tamaño debe ser al menos igual a la constante **MPI_MAX_PROCESSOR_NAME**. En dicho vector quedará almacenado el nombre del procesador. El parámetro **longname** es otro parámetro de salida que nos informa de la longitud de la cadena obtenida.

Esta función es útil para asegurar que los procesos se ejecutan en las máquinas que se han especificado.

ANEXO E

INSTALACION DE MPITB

TOOLBOX - MATLAB/LINUX DE PROCESAMIENTO PARALELO

MATLAB es un estándar para modelado y simulación de prototipos. Se utiliza ampliamente, tanto académicamente como en industria, para resolución de problemas en un extenso espectro de dominios de aplicación. Por otro lado, el estándar MPI fue desarrollado con el fin de evitar tener que aprender diferentes funciones para desarrollar aplicaciones en diferentes sistemas paralelos. MPI especifica una librería de funciones que puede ser llamada desde C o Fortran y que permite paso de mensajes. Existen diversas toolbox para el procesamiento paralelo bajo el entorno MATLAB tales como: MultiMATLAB[9], MatlabMPI[10], pMatlab[11], MPITB/PVMTB[12], MULTI[13],…; las cuales convierten a MATLAB en un entorno paralelo para modelado y simulación, estas interfaces facilitan la introducción del amplio número de usuarios MATLAB en el procesamiento paralelo. En el desarrollo del presente trabajo de grado se trabajo con MPITB una *Toolbox* de programación paralela bajo Linux/MATLAB desarrollada por Javier Fernández Baldomero, la cual es de libre distribución. Esta guía pretende dar las pautas para la instalación de MATLAB en Linux y la toolbox de

[9] http://www.cs.cornell.edu/Info/People/lnt/multimatlab.html
[10] http://arXiv.org/abs/astro-ph/0107406
[11] http://www.ll.mit.edu/hpec/agendas/proc03/abstracts/kepner-pMatlab.pdf
[12] http://atc.ugr.es/javier-bin/mpitb_eng http://atc.ugr.es/javier-bin/pvmtb_eng
[13] http://www.lapsi.eletro.ufrgs.br/Disciplinas/ENG_ELETRICA/CAD-ENG/Matlab/CommSim/

procesamiento paralelo MPITB, sin embargo no pretende ser una guía para la utilización de MPITB ya que esta toolbox viene con un tutorial autodidáctico muy completo.

INSTALACION DE MATLAB BAJO LINUX

A continuación se describen los pasos necesarios para la instalación de MATLAB 7 bajo el sistema operativo Linux, si todo sale bien, finalmente se visualizara en pantalla similar a la figura E.1.

REQUISITOS

- Sistema operativo Linux RedHat 9 instalado previamente.
- Instaladores de MATLAB 7 para el sistema operativo Linux.
- Archivos de licencias para MATLAB 7[14]:
    - *license.dat*
    - *license.lic*
    - *flexlm* - License Manager Software.

[14] Estos archivos pueden encontrarse en Internet, si es miembro de la comunidad UIS, puede escribir un correo a renea_barrera@hotmail.com para obtener los respectivos archivos, para el respectivo uso academico.

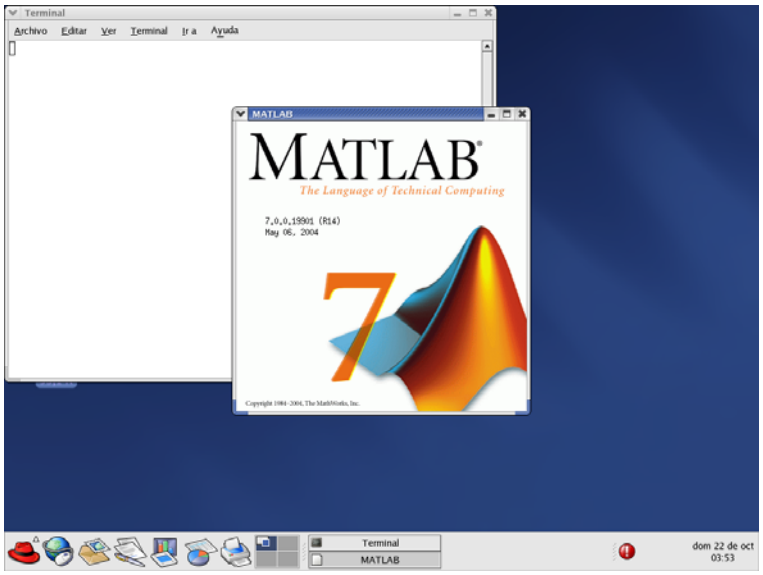

Figura E.1 Pantalla Inicial de MATLAB/LINUX

INICIAR EL INSTALADOR

Lo primero que se debe realizar es la creación del directorio donde se va a instalar MATLAB; comúnmente en Linux el software de usuario se instala en el directorio ***/usr/local/***, por lo tanto se creara una carpeta para los archivos de MATLAB en ese directorio. Para tal fin ejecute los siguientes comandos:

cd /usr/local/ # ubicarse en el directorio

mkdir MATLAB7 #crea el directorio donde se instalara MATLAB[15]

Una vez creado el directorio MATLAB7, copie el archivo *license.dat* en esa ubicación.

[15] El directorio de instalación para MATLAB puede ser cualquiera, sin embargo para esta guía será MATLAB7

cp *license.dat* /usr/local/MATLAB7/ #copia el archivo

Este archivo permite realizar la instalación de MATLAB con los instaladores provistos. Inserte el disco 1 de instalación y ejecute en modo grafico el archivo install que viene en el disco, elija la opción ejecutar en una Terminal. Si todo va bien debe aparecer una pantalla similar a la figura E.2. Allí ingrese la dirección donde realizara la instalación de MATLAB (en este caso ***/usr/local/MATLAB7***), y luego OK. En este punto lo que sigue es terminar la instalación, la cual es muy sencilla (OK→ OK→ OK) por lo tanto no se especifica en este documento.

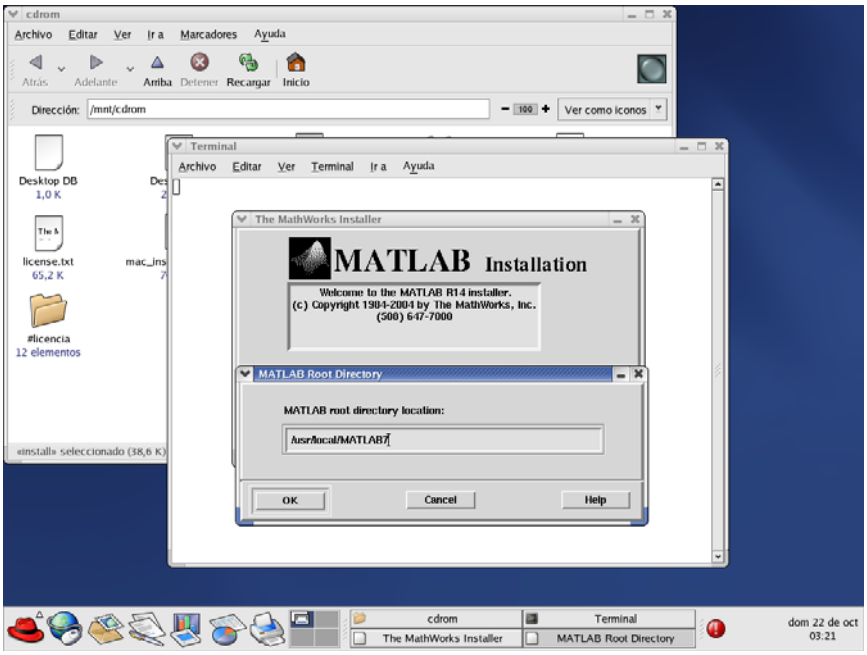

Figura E.2. Pantalla de instalación MATLAB

ARCHIVO LISENCE.LIC

Una vez finalizada la instalación se debe proseguir con la autentificación de la licencia, para lo cual se dispone del archivo *lisence.lic*; este archivo contiene una serie de información que revisa MATLAB al inicializarse y entre la cual esta su hostname[16], por eso es necesario editar el contenido de este archivo; para obtener su hostname ejecute en una consola el comando:

hostname # Devuelve su hostname (Ej. Nodo1.E3T.cluster)

Una vez obtenido su hostname, edite el archivo *lisence.lic* reemplazando el campo YOUR_HOSTNAME con su respectivo nombre de red o hostname.

LICENSE MANAGER SOFTWARE

MATLAB esta protegido por el FLEXLM Manager de Licencias. El manager debe estar corriendo antes de iniciar MATLAB. Se puede iniciar el manager manualmente o mediante scripts[17] en el inicio del sistema. Para realizarlo manualmente cada vez que inicie MATLAB antes debe ejecutar el siguiente comando:

/usr/local/MATLAB7/etc/lmstart

Para automatizar esta tarea se puede utilizar el script flexlm. Para tal fin ejecute

[16] Nombre exclusivo asignado a un PC en la red
[17] Archivo de secuencia de comando en Linux

como root[18]:

cp *flexlm* /etc/init.d/

Redhat 9 tiene un panel de servicios, el cual permite habilitar, deshabilitar, iniciar o detener servicios en el sistema. Para utilizar esta aplicación es necesario ingresar el *flexlm* script en este panel, por lo tanto ejecute:

chkconfig --add *flexlm*

Ahora debe abrir el panel de servicios de Redhat 9[19], en el cual debe estar listado a la izquierda el flexlm, marque el checkbox de la izquierda de *flexlm* para habilitarlo, luego haga clic en Start, con lo cual debe aparecer en la casilla status "FLEXlm License Manager running..." (Ver figura E.3); con esto el manager de licencia Flexlm se iniciara automáticamente al arrancar el sistema.

[18] Estas especificaciones son para Linux REDHAT 9, si tiene otra distribución Linux posiblemente deba alterar el procedimiento.
[19] Ruta para abrir la aplicación: (redhat→config→services)

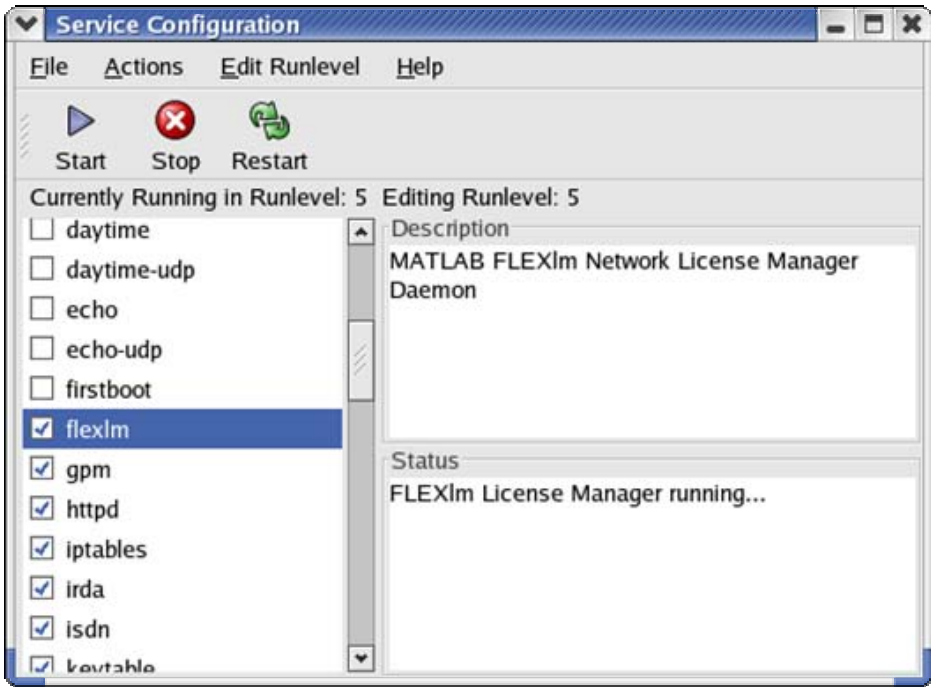

Figura E.3. Activación de Flexlm en el panel de servicios de RedHat 9.

Finalmente ya puede iniciar MATLAB. Solo debe escribir "**matlab**" en una consola y debe aparecer algo similar a la figura E.1; sino funciona tal vez no se crearon los enlaces simbólicos respectivos, por lo tanto alternativamente puede escribir la dirección completa del script ***/usr/local/MATLAB7/bin/matlab***; si definitivamente no lo logra revise los pasos anteriores.

INSTALACION DE MPITB

MPITB es un toolbox usada para llamar rutinas de la librería MPI desde el interprete de comandos de MATLAB, se pueden arrancar procesos hijos y organizarlos en topologías, enviar y recibir variables del entorno MATLAB y hasta pasar comandos a los nodos de computo deseados. Es útil para la enseñanza de LAM-MPI, así como también para investigación debido a la creación rápida de prototipos en el entorno MATLAB.

REQUISITOS

- Una instalación de LAM 7.1.x con las librerías compartidas. Para ello LAM puede ser configurado en el momento de la instalación usando[20]:

  **./configure -prefix=/<LAMHOME[21]> --enable-shared --disable-static **

  **--without-fc --without-mpi2cpp --without-romio **

  --with-threads=posix --with-trillium --with-modules

- Un MATLAB utilizable en cada nodo (en el cual se desee desplegar un proceso MATLAB)
- Código fuente y precompilado de MPITB para MATLAB 7.0.1 R14SP1, LAM 7.1.1 (mpitb-FC3-R14SP1-LAM711.tgz)[22]

[20] Ver anexo C – Instalación de LAM-MPI
[21] Ubicación para la instalación de LAM-MPI
[22] Puede descargarlo de: http://atc.ugr.es/~javier/investigacion/mpitb/mpitb-FC3-R14SP1-LAM711.tgz

INSTALACION

- **Decidir Ubicación**

1. MPITB para todos los usuarios MATLAB (No usual):

Se asume que MATLAB esta instalado en **$MATLAB= /usr/local/MATLAB7**

Situé el archivo del código fuente de MPITB en donde las toolbox de MATLAB son usualmente almacenadas ***($MATLAB/toolbox/)***

cp mpitb.tgz /usr/local/MATLAB7/toolbox/

cd /usr/local/MATLAB7/toolbox/

2. MPITB para un solo usuario (configuración usual):

Se asume que el usuario ***lam*** instalara MPITB

De no tenerlo, cree un directorio de trabajo para MATLAB y ubique el archivo fuente de MPITB alli.

mkdir /home/lam/matlab

cp mpitb.tgz / home/lam/matlab

cd / home/lam/matlab

- **Descomprimir y desempaquetar**

El archive fuente de MPITB viene compreso y empaquetado por lo tanto se debe ejecutar lo siguiente:

tar zxvf <donde_este>/mpitb.tgz

Con lo cual debe aparecer el subdirectorio (***/home/lam/matlab/mpitb*** para el

usuario ***lam*** o ***/usr/local/MATLAB7/toolbox/mpitb***)

- **Configurar el arranque de MATLAB (startup)**

MPITB viene con un archivo startup.m el cual permite que MATLAB lo reconozca al iniciar una sesión.[23]

1. Si esta instalando en ***$MATLAB=/usr/local/MATLAB7*** haga lo siguiente:

   cd $MATLAB/toolbox/local

   ln -s ../mpitb/startups/startup_MPITB.m .

Añada estas líneas al startup.m original en ***$MATLAB/toolbox/local***

```
if exist('startup_MPITB','file')
   startup_MPITB
end
```

2. Si esta instalado MPITB para el usuario ***lam***, se asumirá que este usuario no posee ningún archivo startup.m previo.

   cd /home/lam/matlab

   ln -s mpitb/startups/startup_MPITB.m startup.m

Si se realiza todo correctamente, la próxima vez que MATLAB sea iniciado MPITB imprimirá un par de líneas como:

```
---------------------------------
```

[23] Similar a la acción addpath en MATLAB/Windows

Set SSI rpi to tcp with the command:
putenv(['LAM_MPI_SSI_rpi=tcp']), MPI_Init
Help on MPI: help mpi
>>

- **Configurar el script de inicio de los usuarios**

La librería LAM/MPI requiere que se defina la variable de entorno LAMHOME[24]; además la toolbox MPITB contempla que se defina la variable de entorno MPITB_ROOT. Si no está definida, pero el usuario tiene instalada la toolbox en $HOME/matlab/mpitb, se utiliza dicha ubicación. Esto último es más cómodo.

Para esta configuración edite el archivo $HOME/.bashrc[25], añadiendo las siguientes líneas:

export MATLAB=/usr/local/MATLAB7
export MPITB_ROOT=~/matlab/mpitb
alias matlab='matlab -nosplash -nojvm'[26]

Estas modificaciones son para una instalación de LAM en la cuenta del usuario, una instalación global de MATLAB y una instalación de MPITB local en $HOME/matlab/. Se debe realizar modificaciones según el caso.

Con esto se finaliza la instalación de MPITB, lo que sigue es realizar el

[24] Ver anexo C – Instalación de LAM-MPI
[25] Para usuarios del shell bash
[26] Esta opción es opcional. No despliega la pantalla de inicio de la figura E.1 y además no carga la interfaz grafica predeterminada de MATLAB (jvm)

tutorial[27] que trae la toolbox en la pagina principal o en el archivo ~/matlab/mpitb/tutorial instalado con MPITB. Esta guía es muy completa y extensa por eso no se incluye aquí, además la toolbox trae ejemplos muy interesantes que facilitan la comprensión de esta gran herramienta.

[27] http://atc.ugr.es/~javier/investigacion/mpitb/TUTORIAL-FC3-R14SP1-LAM711