UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Evaluación de Desempeño de Sistemas Paralelos con base en la Coordinación

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

Licenciado en Ciencias de la Computación

P R E S E N T A:

Francisco Javier Mena Barraza

DIRECTOR DE TESIS:
Dr. Jorge Luis Ortega Arjona
2012

Hoja de Datos del jurado

1. Datos del alumno

   Mena

   Barraza

   Francisco Javier

   5531336910

   Universidad Nacional Autónoma de México

   Facultad de Ciencias

   Ciencias de la Computación

   300163847

2. Datos del tutor

   Dr.

   Jorge Luis

   Ortega

   Arjona

3. Datos del sinodal 1

   Dra.

   Hanna

   Oktaba

4. Datos del sinodal 2

   Dr.

   Pedro Eduardo

   Miramontes

   Vidal

5. Datos del sinodal 3

   Dr.

   Héctor

   Benítez

   Pérez

6. Datos del sinodal 4

   Dr.

   Sergio

   Rajsbaum

   Gorodezky

7. Datos del trabajo escrito

   Evaluación de Desempeño de Sistemas Paralelos con base en la Coordinación

   113p

   2012

Índice general

Resumen

Conforme el mundo de la informática avanza, las necesidades y las demandas de las empresas por software de calidad crecen con él. Es por eso que los desarrolladaores de hardware y software se ven frecuentemente en la necesidad de competir por la implementación de soluciones que cumplan estas espectativas.
Dada la dimensión del trabajo que debe ser realizado, la mayoría de las soluciones resultan en sistemas de computo paralelo.
Pero ¿Cómo saber que un sistema de computo paralelo es mejor que otro? Pues bien, por lo regular se respone a esta pregunta diciendo: el mejor sistema es aquél que tenga un buen desempeño. Lamentablemente, esto está lejos de darnos una explicación. Lo cual, nos lleva a otra cuestión: ¿Cómo conocer el desempeño de un sistema paralelo?

En la actualidad, existen diferentes herramientas que ayudan a conocer el desempeño de un sistema de esta naturaleza. Sin embargo, un alto porcentaje de estas herramientas, evalúan el desempeño de un sistema con base en la cantidad de tareas de procesamiento que se puede realizar en un lapso determinado de tiempo. Esto podría ser cierto para sistemas secuenciales, pero no para sistemas parlelos. Dado que estos sistemas se ejecutan en clusters de máquinas o en máquinas multiprocesador, la coordinación y la comumunicación que se generan agregan una peso extra al desempeño total del sistema, que de la forma tradicional puede llegar a omitirse. Sin embargo, para sistemas paralelos es de gran importancia.

Un ejemplo de ello puede ser tratar de conocer el desempeño de un sistema computacional de granularidad fina. En estos sistemas, el tiempo que se invierte en procesamiento es poco, en cambio, el que se invierte en sincronización y comunicación tiende a ser elevado. De esta forma, evaluando el desempeño del sistema de la manera usual puede parecer que tiene un buen desempeño, dado que las tareas de procesamiento se ejecutan de manera rápida. Sin embargo, puede ser que dadas las exigencias de comunicación tenga un desempeño deplorable.

Es por eso que el presente trabajo analiza un sistema de cómputo paralelo con el fin de aproximar y cuantificar el impacto que tiene la coordinación en el desempeño total del sistema. Para lograrlo, se construye un sistema de cómputo paralelo tomando como base algún patrón de diseño para arquitecturas paralelas, con el fin de entender la forma en que se coordina y así poder calcular el impacto que tiene la propia coordinación.

El modelo que se trabaja es un modelo hibrido, en el cual, se modelan las tareas de procesamiento que desempeñan las componentes del sistema de manera determinística, por otro lado, se manejan los tiempos de comunicación de manera estocástica, aprovechando la aleatoriedad natural de este proceso. Con ello, se estima el impacto que tiene la coordinación en el desempeño total del sistema. Esto puede expresarse a través de la siguiente ecuación.

$$T_{total} - T_{procesamiento} = T_{Coordinacion}$$

De esta forma, conociendo el tiempo de ejecución total del sistema y asignando valores deterministas a los tiempos de procesamiento, es posible cuantificar el tiempo total que el sistema emplea en tareas de coordinación.

Dada la naturaleza estocástica de este fenómeno, se realizan una serie de experimentos variando el número de componentes de procesamiento y observando como se comporta la coordinación. Esto con el fin de establecer de que manera impacta la coordinación en el desempeño total del sistema.
El problema sobre el cual se trabaja es uno bien conocidio del cómputo paralelo: el problema de los n-cuerpos.

Para entender este problema, es posible comenzar definiendo el problema de los dos cuerpos. Esté es una simplificación del problema general y nos ayudara a tener clara la idea principal del problema a resolver. El problema de los dos cuerpos consiste en determinar el movimiento de dos partículas puntuales que sólo interactúan entre sí. Los ejemplos comunes incluyen: la Luna orbitando la Tierra en ausencia del Sol, un planeta orbitando una estrella, dos estrellas que giran en torno al centro de sus masas (estrella binaria), o incluso un electrón orbitando en torno a un núcleo atómico [4].

Si se generaliza el problema de los dos cuerpos, podemos definir el problema de los n cuerpos, como la problemática de determinar el movimiento de n partículas puntuales que solo interactúan entre sí.

Es importante señalar que a diferencia del problema de los dos cuerpos (que tiene soluciones exactas), el problema de los tres cuerpos y el problema de n cuerpos no pueden resolverse, excepto en casos especiales.
Debido a que la arquitectura sobre la cual se implementa el sistema es uno de los componentes más importantes, es necesario establecer que el sistema construido toma como base el patrón **Manager workers**.
Este patrón es considerado una variante de el patrón Maestro-Esclavo, desarrollado para sistemas paralelos. Introduce un paralelismo de tareas donde la operación es desempeñada sobre datos ordenados. Cada componente realiza la misma operación, independiente de la actividad de procesamiento de los otros componentes.Es decir, diferentes grupos de datos son procesados simultáneamente.[10].

Capítulo 1

Introducción

1.1. Contexto

Las computadoras a lo largo del tiempo han evolucionado a una velocidad asombrosa. Hoy en día, es posible adquirir por unos $1,500 dólares una computadora con 2 Gb de memoria y capaz de hacer varios millones de operaciones por segundo. Esto, sumado a un notable decremento en el costo, han hecho de las computadoras una parte importante en nuestras actividades diarias, tanto laborales como personales, generando así nuevas exigencias tanto para los diseñadores de hardware como para los de software.

Por un lado, las tendencias en cuanto a software han sido marcadas por un factor importante: el uso de **más memoria**, el cual a su vez, es resultado de otros dos factores importantes:

- Las tareas que ejecutan los programas son cada vez más complejas. Una muestra de ello es que hace algún tiempo, los programas no requerían una interfaz grafica. Todo se manejaba desde comandos de línea. Sin embargo, actualmente, todo está plagado de ventanas, con efectos muy vistosos, que aumentan el consumo dememoria.

- Las mejoras en la tecnología de DRAM han hecho que hoy en día quepa más memoria en menos espacio y a menor costo.

En conjunto, todo lo anterior logra que, ya sea el dueño de una empresa o un simple estudiante, todos los usuarios esperan que su sistema de cómputo cumpla con dos características:

- Sean eficientes, entendiendo por eficiencia que los recursos de hardware o software desarrollados para estos sistemas aprovechen al máximo las capacidades de los otros componentes.
- Sean de alto desempeño, capaces de responder con rapidez a la mayor cantidad de peticiones que los usuarios exijan, manteniendo una eficiencia media.

Pero estas exigencias se dan la mayoría de las veces, sin pensar en las restricciones de costo y las limitaciones tecnológicas inherentes a los dispositivos que utilizan. Actualmente se construyen equipos que integran estos conceptos bastante bien. Pero aún con esto, existen necesidades actuales que un simple computador no puede cubrir, lo que ha hecho que los desarrolladores tanto de hardware como de software traten de encontrar mejores soluciones que, en su mayoría, han resultado en sistemas de cómputo paralelo. Ya sea con una supercomputadora con múltiples procesadores, o clusters de máquinas conectadas a través de una red, siempre se cree "si dos cabezas piensan mejor que una", entonces dos procesadores trabajan mejor que uno.

Esto no siempre es verdad. En ocasiones, la implementación de un sistema paralelo para la solución de un problema específico, puede llegar a ser más ineficiente que la solución secuencial, ya que el éxito de estos sistemas depende en gran medida del problema a solucionar.

Es entonces cuando se tiene que identificar qué sistema de cómputo se ajusta más a mis necesidades, es decir, que sistema se desempeña mejor con base en mis exigencias.

1.2. Problema

Inicialmente, los sistemas paralelos se diseñan para obtener un mejor desempeño en procesamiento. Si bien es cierto que en muchos casos, los sistemas paralelos sí ayudan a procesar grandes cantidades de datos mas rápidamente, el procesamiento no es el único factor relevante. Existe otro factor comúnmente ignorado, pero de igual forma importante y en la mayoría de los casos más complicado de definir: la coordinación. ¿Cómo se comunican entre sí las unidades de procesamiento para devolver el resultado esperado? ¿Cómo se distribuye la carga de trabajo entre las unidades de procesamiento? De la misma forma que el procesamiento, la coordinación influye en gran medida en el desempeño del sistema, pero ¿Cómo es que estos dos conceptos se relacionan? es decir ¿Cómo impacta la coordinación el desempeño de un sistema paralelo? ¿Cómo podemos cuantificar la comunicación dentro de un sistema de computo paralelo?

En la actualidad la mayoría de lo métricas de evaluación de desempeño solo se enfocan en validar la cantidad de operaciones que pueden realizarse en una unidad de tiempo, sin mebargo, esto provoca que no se contabilicen cosas importantes como el timpo que el sitema entero pierde en coordinar sus tareas. En sistemas complejos, este tiempo puede aumentar demasiado y decrementar de manera considerable el desempeño total de la aplicación.
Es por eso que el presente trabajo se enfoca en describir una forma de cuantificar y aproximar el impacto que tiene la coordinación dentro de el desempeño total de un sistema de cómputo paralelo, apoyandose en conceptos auxiliares como lo son los patrones de arquitectura paralela, modelos determinísticos, diagrmas de UML y otros componentes que son descritos a detalle en capítulos posteriores.

1.3. Hipótesis

En un sistema de cómputo, ya sea secuencial o paralelo, el tiempo de respuesta, el rendimiento y la coordinación, son factores que marcan puntos de vista relevantes de como evaluar el desempeño. En este tipo de evaluaciones se tiene como elemento relevante el procesamiento. Esto es, que tiempo tarda el sistema, para procesar una o varias tareas. Sin embargo, dentro de un sistema paralelo, es necesario considerar como otro componente importante: *"la coordinación"*.
Por este motivo, el presente trabajo muestra la importancia de la coordinación dentro de los procesos de evaluación de desempeño de sistemas paralelos y trata de aproximar el porcentaje de su impacto a través de la respuesta a la pregunta:

> Dado un sistema de computo paralelo ¿Cuán significativo es el impacto de la coordinación dentro del desempeño total de éste.

1.4. Aproximación

El objetivo de esta tesis es analizar y cuantificar el impacto que tiene la coordinación en el desempeño total de un sistema de cómputo paralelo. Para lograrlo, se construye un sistema de esta naturaleza con base en el patrón **manager workers**[7] con el fin de tener clara la forma en que trabaja el sistema, y con ello, manipularlo para poder calcular el impacto real que la coordinación ejerce sobre su desempeño.

El modelo sobre el cual se trabaja es un modelo híbrido, en el cual se modelan las tareas de procesamiento que desempeñan las componentes del sistema de manera determinística. Por otro lado, se manejan los tiempos de comunicación de manera estocástica, aprovechando la aleatoriedad natural, y con ello, se estima el impacto que tiene la coordinación en el desempeño total del sistema. Esta idea puede plasmarse de manera más clara con la siguiente ecuación:

$$T_{total} - T_{procesamiento} = T_{Coordinacion}$$

Entonces, conociendo el tiempo de ejecución total del sistema y asignando valores determinísticos a los tiempos de procesamiento, es posible estimar el tiempo que se emplea en coordinación.

Dada la naturaleza estocástica de este fenómeno, se realizan una serie de experimentos variando el número de componentes de procesamiento y sus cargas de trabajo, con el fin de observar cómo se comporta la coordinación y aproximar su impacto en el desempeño total del sistema.
El sistema de cómputo, toma como base un algoritmo propuesto por Brinch Hansen [13] para resolver un problema bien conocidio del computo paralelo: el problema de los n-cuerpos. El cual podemos definir como el la problematíca de determinar el movimiento de n partículas puntuales que solo interactúan entre si.

Debido a que la arquitectura sobre la cual se implementa el sistema es uno de los componentes mas importantes, es necesario establecer que el sistema construido toma como base el patrón el diseño, **Manager workers**[7].

1.5. Contribuciones

Entre las principales contribuciones del presente trabajo, se encuentran:

- La implementación de un sistema de cómputo paralelo que modela el problema de los n-cuerpos, basado en el patrón arquitectónico “Manager Workers”.
- Una descripción detallada de todos los componentes del sistema de manera estática y dinámica.
- La implementación de un modelo de evaluación de desempeño híbrido, compuesto por 2 submodelos: uno determinístico y otro estocástico, los cuales en conjunto proporcionan un mejor control de los tiempos de procesamiento y los de coordinación, facilitando su medición y por ende, facilitando la aproximación de como la coordinación impacta el desempeño total del sistema.
- Construye base sobre la cual se pueden realizar posteriores investigaciones sobre el impacto de la coordinacion en sistemas paralelos.
- Presenta una métrica sencilla que aproxima el impacto de la coordinación dentro de un sistema de cómputo paralelo, definiendo una base para el estudio y generación de nuevas métricas cada vez mas precisas.

1.6. Estructura de la tesis

A continuanción se muestra la organización y descripción de los contenidos de esta tesis.

- **Capítulo 2 Antecedentes**

  Se Identifican de manera general los principales conceptos que definen el cómputo paralelo (procesos concurrrentes, tipos de paralelismo, granularidad, etc). Con el fin de facilitar y proveer herramientas para la coomprensión de este trabajo.

- **Capítulo 3 Trabajo relacionado**

  Algunas aproximaciones realizadas en el ambito de los sistemas de cómputo paralelo para evaluación de desempeño. Por un lado, se ponen en perspectiva algunas de las aproximaciones que se han realizado para cuantificar el desempeño de un sistema paralelo, y por otro, los problemas que pueden presentarse.

- **Capítulo 4 Caso de estudio: Problema de los N cuerpos**

  Se describe a detalle el problema de los N cuerpos y la forma en la cual será implementado como un sistema paralelo que aproxime su solución. Además, se muestran las secciones del código fuente que son relevantes para el cálculo del problema de los N cuerpos.

- **Capítulo 5 Diseñando un sistema paralelo**

  Se presentan de manera detallada las clases, funciones y paquetes que definen el sistema paralelo implementado. Además, las definiciones de los componentes de procesamiento se complementan con diagramas que muestran de forma estática y dinámica el comportamiento general del sistema.

- **Capítulo 6 Experimentación y Desarrollo**

  Se describe el proceso a seguir para definir el impacto de la coordinación dentro del sistema paralelo. Se tratan puntos importantes, como es la caracterización de la carga de trabajo y la varibilidad de componentes del sistema. La estimación del tiempo que se obtiene en esta experimentación es complementada con gráficas para un mayor entendimiento de los resultados obtenidos.

- **Capítulo 7 Evaluación**

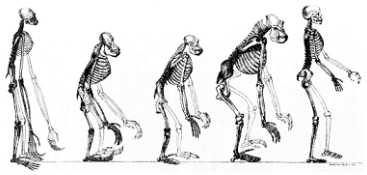
  Se evalúa la calidad del trabajo realizado y los resultados obtenidos, para tener una línea base para generar las conclusiones. Además, se comparan los métodos utilizados y los resultados obtenidos, con aquellos métodos y resultados definidos en el apartado de trabajo relacionado

- **Capítulo 8 Conclusiones**

  Declaración de las conclusiones que se han obtenido basados en los resultados del trabajo realizado.

Capítulo 2

Antecedentes



En este capítulo se presentan los conceptos básicos y términología usados a lo largo de esta tesis agrupados en tres bloques:

- Procesamiento paralelo y distribuido
- Patrones arquitectónicos para diseño de sistemas paralelos
- Desempeño de sistemas paralelos

Esta agrupación permite un acercamiento gradual a los temas facilitando asi su comprensión y ayudando a respaldar las aportaciones presentadas.

2.1. Procesamiento Paralelo y Distribuido

En esta sección se Identifica de manera general, los principales conceptos que definen el cómputo paralelo (procesos concurrrentes, tipos de paralelismo, granularidad, etc), presentando de una manera general los términos y principios sobre la forma de trabajo y las características de un sistema de esta naturaleza.

2.1.1. Conceptos básicos de cómputo paralelo

Antes de hablar de los sitemas paralelos, es necesario definir un término fundamental que será usado a lo largo del presente trabajo, la “concurrencia”, para efectos de esta tesis temaremos a la concurrencia como la ejecución de múltiples procesos interactivos que son ejecutados simultaneamente. Con esto, será posible entender como es que se constituyen los sistemas paralelos.

Con la defnición anterior, podemos entender un poco más el “cómputo paralelo”. Es por ello que para efectos de esta tesis, tomaremos la siguiente definición:

> El **cómputo** paralelo es una forma de programación en la cual multiples instrucciones se ejecutan simultáneamente[3].

Estas instrucciones pueden agruparse dando forma a lo que se conoce como procesos:

> Un **proceso**, es cualquier secuencia de operaciones que están ejecutándose en memoria activa, que realizan una o varias acciones sobre ciertos datos y permiten describir el comportamiento de un objeto mediante las acciones que es determinado a realizar[2].

Existen procesos especificos que son relevantes dentro del procesamiento paralelo, los procesos concurrentes disjuntos y cooperativos[2]:

- **Procesos disjuntos:**

  Se dice que dos o más procesos concurrentes son disjuntos si se ejecutan simultáneamente en diferentes bloques de un programa, sin posibilidad de accederse entre sí (Figura 2.1). Inician su ejecución simultáneamente y proceden concurrentemente hasta que terminan. Es importante resaltar que la ejecución de tales procesos concurrentes tienen el mismo efecto que su ejecución secuencial en cualquier orden.

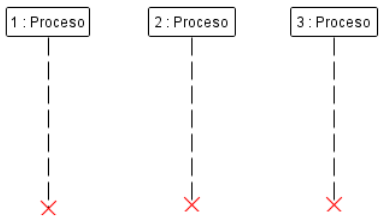


Figura 2.1: Procesos Disjuntos

- **Procesos cooperativos:**

  Los procesos concurrentes cooperativos tienen la posibilidad de comunicarse para cooperar entre sí en la realización de algúna tarea, ya sea compartiendo recursos en común o haciendo necesaria algúna forma de sincronización entre ellos para evitar la corrupción de los datos (Figura 2.2).

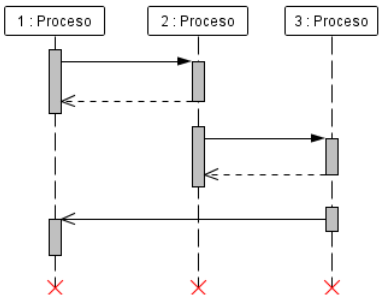


Figura 2.2: Procesos Cooperativos

Como resultado de la interacción entre los procesos concurrentes cooperativos, se han tenido que desarrollar mecanismos de sincronización para mantener la inregridad de los datos que se comparten. Al tener que ejecutarse en un solo procesador, estos procesos se encuentran con una condición de competencia por los recursos, como la memoria o el propio procesador.

Estos mecanismos de sincronización también son empleados para mantener la integridad de los de datos o para evitar problemas como los denominados “deadlocks”.
algúnos de los mecanismos de sincronización desarrollados son:

- Semáforos
- Regiones críticas
- Monitores
- Monitores anidados

Una vez planteado el concepto de proceso, es fácil intuir que éstos pueden a su vez ser agrupados en programas, generando así programas paralelos o distribuidos.

Un programa secuencial, un programa paralelo o distribuido resultan en el seguimiento de múltiples hebras de control (cada uno constituyendo un proceso secuencial), que se comunican entre sí.

Para efectos de esta tesis, se tomará la definición de un programa concurrente y uno paralelo o distribuido como sigue [2]:

- Un programa concurrente es aquél formado por varios procesos que se ejecutan (en general) en un solo procesador, siguiendo un esquema de paralelismo temporal (Figura 2.3)

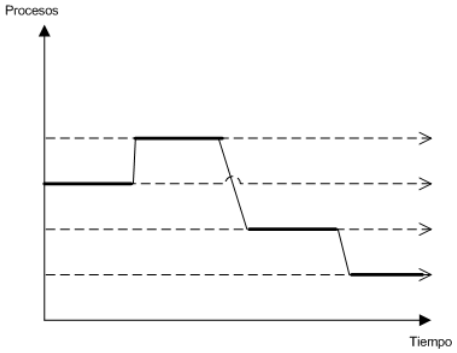


Figura 2.3: Ejecución en el tiempo de un Programa Concurrente

- Un programa paralelo o distribuido es aquél formado por varios procesos que se ejecutan en varios procesadores conectados entre sí mediante una red de comunicación, siguiendo un esquema de paralelismo espacial. Si la red se forma por conexiones dentro de una sola computadora, se considera un sistema de programación paralela; por otro lado, si la red se forma de conexiones entre diferentes computadoras, se considera un sistema de programación distribuido (Figura 2.4).

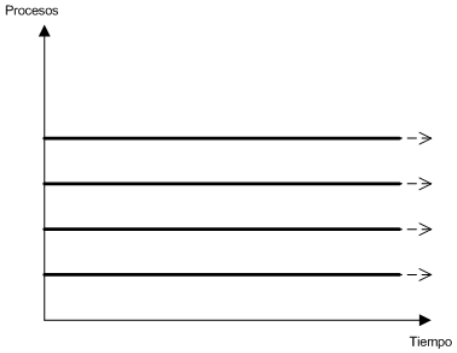


Figura 2.4: Ejecución en le tiempo de un Programa paralelo Distribuido

Los programas paralelos pueden ser clasificados de diversas maneras. Inicialmente tomaremos como base la nautraleza de sus componentes de procesamiento. De esta forma, podemos clasificarlos en **homogéneos** y **heterogéneos**. Es importante señalar que esta clasificación expone propiedades importantes que son tomadas en cuenta para definir el tipos de comunicación que se usa entre componentes.

- *Sistemas Homogéneos*. Son aquéllos basados en componentes idénticos que interactúan de acuerdo a un conjunto simple de reglas de comportamiento. Individualmente, cada componente puede ser intercambiando con otro sin algún cambio notable en la operación del sistema. Usualmente los sistemas homogéneos tienen un gran número de componentes, los cuales se comunican usando operaciones de intercambio de datos[7].

- *Sistemas Heterogéneos*. Son aquelos basados en diferentes componentes con reglas de comportamiento y relaciones especializadas. Básicamente, la operación de los sistemas depende de las diferencias entre los componentes y, por lo tanto, los componentes no pueden ser intercambiados entre si. En general, sistemas heterogéneos están compuestos de algúnos componente más que los sistemas homogéneos, y se comunican usando llamadas a funciones [7].

Hasta aqui, se ha presentado una idea general de los componentes básicos de un sistema paralelo y una primera aproximación de como pueden ser clasificados. Una forma alternativa de clasificar los sistemas paralelos se basa en las característica del problema a resolver. Con base en esto se consideran los siguientes tipos de paralelismo[7]:

- Paralelismo funcional: La característica principal de este tipo de paralelismo es que las aplicaciones son divididas en funciones. Se puede ver como paralelismo de código. Por ejemplo puede dividirse en entrada, preparación del problema, solución del problema, preparación de la salida, salida y mostrar la solución. Esto permite a todos los nodos producir una cadena. Esta aproximación es como la segmentación en funciones de un procesador. En este esquema se sigue la misma idea, pero a nivel de software: se dividen los procesos formando una cadena, dependiendo cada uno del siguiente. Aunque al principio no se logre el paralelismo, una vez que se ponen todos los nodos a trabajar (i.e. cuando hayamos ejecutado N veces lo que estemos ejecutando, siendo N el número de pasos en los que hemos dividido la aplicación) se consiguen que todos los nodos estén trabajando a la vez. Esto si todas las funciones tardan el mismo tiempo en completarse. De lo contrario, las demás funciones tienen que esperar a que se complete la función más lenta.

- El paralelismo de datos: El paralelismo de datos se basa en dividir los datos que se tienen que procesar. Típicamente los procesos que están usando esos datos son idénticos entre sí, lo único que hacen es dividir la cantidad de información entre los nodos y procesarla en paralelo. Esta técnica es más usada que la anterior debido a que es más sencillo realizar el paralelismo.

- Paralelismo de tareas: Es un paradigma de la programación concurrente que consiste en asignar distintas tareas a cada uno de los procesadores de un sistema de cómputo. En consecuencia, cada procesador efectuará su propia secuencia de operaciones. A diferencia del paralelismo funcional, la tarea resultante de un nodo, no es necesariamente la entrada del otro.

  En su modo más general, el paralelismo de tareas se representa mediante un grafo de tareas, el cual es subdividido en subgrafos que son luego asignados a diferentes procesadores. La forma como se divida el grafo impactará la eficiencia de paralelismo resultante. La partición y asignación óptima de un grafo de tareas para ejecución concurrente es un problema NP-completo, por lo que en la práctica se dispone de métodos heurísticos aproximados para lograr una asignación cercana a la óptima.

2.1.2. Métodos de paralelización

El problema principal al tratar de definir el diseño de un sistema paralelo es la forma de poder realizar la paralelización, esto es, definir como el problema a resolver puede ser transformado de tal manera que corresponda a la estructura de un sistema paralelo.

Através del tiempo han sido propuestos diferentes métodos de paralelización, cada uno de los cuales constituye una manera diferente de expresar una solución paralela a un problema dado. Es importante mencionar que el desempeño de un método de paralelización depende directamente de la topología utilizada, pero ésta no lo determina totalmente. algúnos métodos se expresan más claramente en ciertas topologías que otros. Un ejemplo de esto puede ser el caso del procesamiento de imagenes, o el cálculo de la transformada rápida de Fourier en un hipercubo.

Al mapear un problema a una plataforma paralela, primero es necesario dividir el problema en segmentos que se ejecuten en paralelo, determinando la manera como los procesadores se comunican y sincronizan entre sí.
Dentro del problema de asignación de los procesos a los procesadores existen elementos de la programación paralela que se relacionan entre sí y determinan el nivel de paralelismo utilizado. Estos son la granularidad y le balance de trabajo, las cuales para el contexto de esta tesis se definirán como sigue:

- Granularidad

  Es un indicador de la cantidad de procesamiento que cada procesador puede realizar independientemente, en relación con el tiempo que ocupa para intercambiar información con otros procesadores.

  Una aplicación de granularidad gruesa (*coarse-grained*) puede ser dividida en partes lógicas formadas por procesos de secuencias independientes, con poca sincronización o comunicación. En un esquema de granularidad gruesa, la relación de tiempo de procesameiento entre tiempo de comunicación es alta.

  Por otro lado, en una aplicación de granularidad fina (*fine-grained*) se realizan unas cuantas instrucciones de comunicación entre procesadores. En el esquema de granularidad fina, la relación de tiempo de procesamiento entre tiempo comunicación es baja.

  La finalidad de la granularidad es determinar la mejor forma de divididr el problema en conjuntos de tareas, de tal manera que el tiempo de ejecución de cada conjunto sea mínimo. El tamaño del conjunto puede ser alterado añadiendo o eliminando tareas, las cuales se definen como granos.

  Si la granularidad es gruesa, el tamaño de los granos es demasiado grande y por consecuencia, el nivel de paralelismo disminuye, ya que algúnas tareas que son potencialmente paralelas se han agrupado en granos que se ejecutan secuencialmente en un procesador. En caso contrario, cuando la granularidad es muy fina,

el tamaño de los granos es tan pequeño que existe una pérdida de tiempo de ejecución por un aumento en los tiempos de comunicación entre procesadores.

Lamentablemente, no hay una solución general al problema de selección del grano a utilizar. La granularidad de una aplicación se obtiene en algúnos casos a partir del grado de interacción dentro de las tareas de un problema y algúnas veces a partir de la arquitectura paralela a ser usada. Un sistema formado por pocos procesadores potentes y unidos entre sí por ligas de comunicación relativamente lentas se ajusta más a un esquema de granularidad gruesa; en el otro extremo, un sistema con elementos de procesamiento muy pequeños, pero que se comunican relativamente rapido, se presta para un esquema de granularidad fina[2].

- Balance de trabajo

El balance de trabajo se refiere a la manera de repartir el total de tareas a realizar entre el total de procesadores disponibles.

Un balance de trabajo óptimo es aquél que mantiene a cada procesador ocupado y procura que todos los procesadores terminen casi al mismo tiempo. Se dice que un programa está balanceado si su procesamiento se encuentra distriubido equitativamente entre todos los procesadores.

El balance de trabajo puede realizarse de dos maneras pricipalmente: estática y dinámicamente.

Una aplicación puede ser analizada con el fin de conocer si es posible balancearla, esto es, conocer de antemano la carga de trajabajo por procesador, determinando estáticamente una distribución de trabajo entre los procesadores durante la compilación. A esto se le conoce como balance de trabajo estático.

Si por el contrario, no es posible conocer previamente la carga de trabajo, se puede optar por ejecutar dinámicamente el programa. A esto se le conoce como un balance de carga dínámico.

Existen varias técnicas para realizar de la mejor forma el balance de trabajo de un programa paralelo. Las técnicas basadas en el balance estático pueden ser utilizadas directamente por el programador, mientras que las técnicas de balance dinámico se aplican ya sea mediante un sistema operativo o el uso de algún tipo de software durante la ejecución del programa[2].

2.2. Patrones Arquitectónicos para diseño de Sistemas Paralelos

Los patrones arquitectónicos pueden ser vistos como plantillas, expresiones o especificaciones de las propiedades estructurales de un sistema,

asi como las relaciones y responsabilidades de los componentes que lo definen.

La selección de un patrón de arquitectura se considera una desicisión fundamental durante el diseño de un sistema de software[7].

Los patrones arquitectónicos para programación paralela son definidos y clsificados de acuerdo a las restricciones en el orden de datos y en las operaciones, así como también por la naturaleza de los componentes de procesamiento del sistema a modelar.

A continuación se listan cinco patrones arquitectónicos comúnmente usados en sistemas de programación paralela. Estos patrones serán explicados con mas detalle a continuación.

- *Parallel pipes and Filter*
- *Parallel Hierarchies*
- *Communicating Sequential Elements*
- *Manager-Workers*
- *Shared Resource*

2.2.1. Paralell pipes and Filters

Es una muestra de paralelismo funcional, ya que sigue un orden precisio de operaciones y datos. Por lo regular, cada componente representa un paso diferente del cómputo total los datos son enviados de un componente de procesamiento a otro (*filter*) mediante una una estructura de comunicación (*paralell pipe*)[11].

- *Estructura*

  Este patrón es llamado *Parallel pipes and Filters* debido a que los datos que se deben procesar son enviados como un flujo de un componente de procesamiento (*filer* o *filtro*)a otra mediante una tuberia (*pipe*) que los une. Su principal característica es que los datos resultantes son procesados de una sola manera através de la estructura. El resultado se va construyendo eventualmente mientras los datos son enviados de una unidad de procesamiento a otra. Difrerentes componentes existen y son procesados simultáneamente durante el tiempo de ejecución.

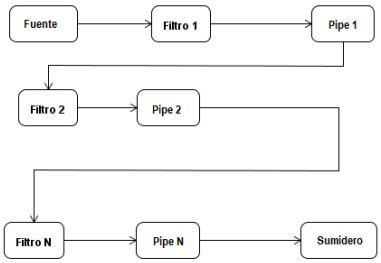


Figura 2.5: Diagrama de objetos del Patrón Parallel pipes and Filter

- *Componentes*

  - **Filtro**: Su responsabilidad es la de obtener datos de entrada de un elemento *tubo* , para ejecutar operaciones localmente sobre ellos, y enviarlos como datos de salida a un conjunto de objetos *tubo*.
  - **parallel pipe o tubo**: La responsabilidad de una componente *pipe* es transferir los datos entre las componentes *filtro*. Algúnas veces sirven como buffers de datos o como métodos de sincronización de actividades entre componentes de procesamiento *filtro* vecinas.
  - **Source o Recurso**: La responsabilidad de un componente recurso es generar los datos iiciales al primer filter, para poder iniciar con esto el proceso de procesamiento.
  - **Sink o Resultado**: La responsabilidad de un componente sink es obtener y reuinir el resultado final del cómputo total.

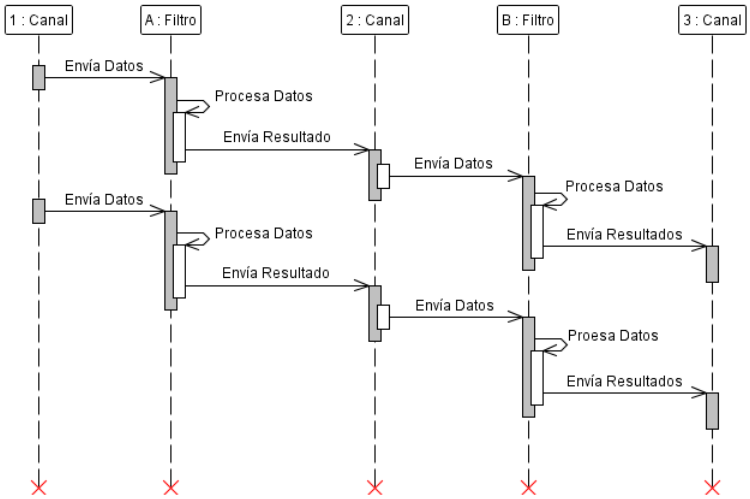


Figura 2.6: Diagrama de Secuencia del Patrón Parallel pipes and Filter

2.2.2. Parallel Layers

Este patrón es una aproximación del patrón de capas para sistemas paralelos. Considera un paralelismo funcional, ya que el orden de las operaciones sobre los datos es una restricción importante. El paralelismo es introducido cuando más componentes de una capa tiene la posibilidad existir simultánemente. Estos componentes pueden ser creados estáticamente, solo esperan por llamadas de capas superiores, o dinámicamente, cuando son generados como resultado de una llamada [12].

- *Estructura*

  Este patrón está compuesto de entidades conceptualmente independientes, ordenadas en en forma de niveles. Cada nivel, implica un diferente nivel de abstracción compuesto de elementos que desempeñan la misma operación. Para comunicar diferentes niveles de la estructura se usan llamadas a funciones. El cómputo se realiza por componentes agrupados por funcionalidad. La Figura 2.7 muestra una red de componentes que siguen este patrón estructural.

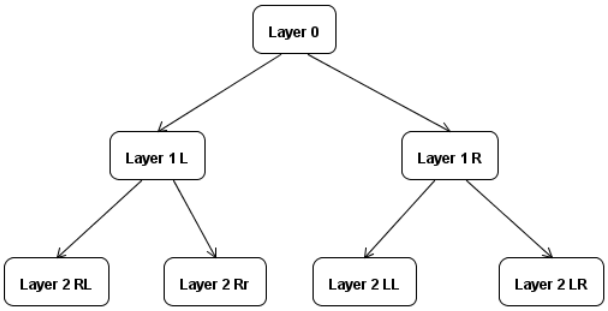


Figura 2.7: Diagrama de Objetos del Patrón Parallel Layers

- *Componentes*

  - **layer**. La responsabilidad de una componente *layer* es proveer una estructura algorítmica en forma de árbol, la cual presenta niveles operación o funcionalidad, las cuales ban delegando ope-raciones o funcionalidades a capas inferiores. El resultado final se logra distribuyendo datos através de las capas inferiores, y recibiendo resultados parciales de estos componentes. Las actividades de los componentes son independientes entre sí, por lo que es posible ejecutarlas fácilmente en paralelo.

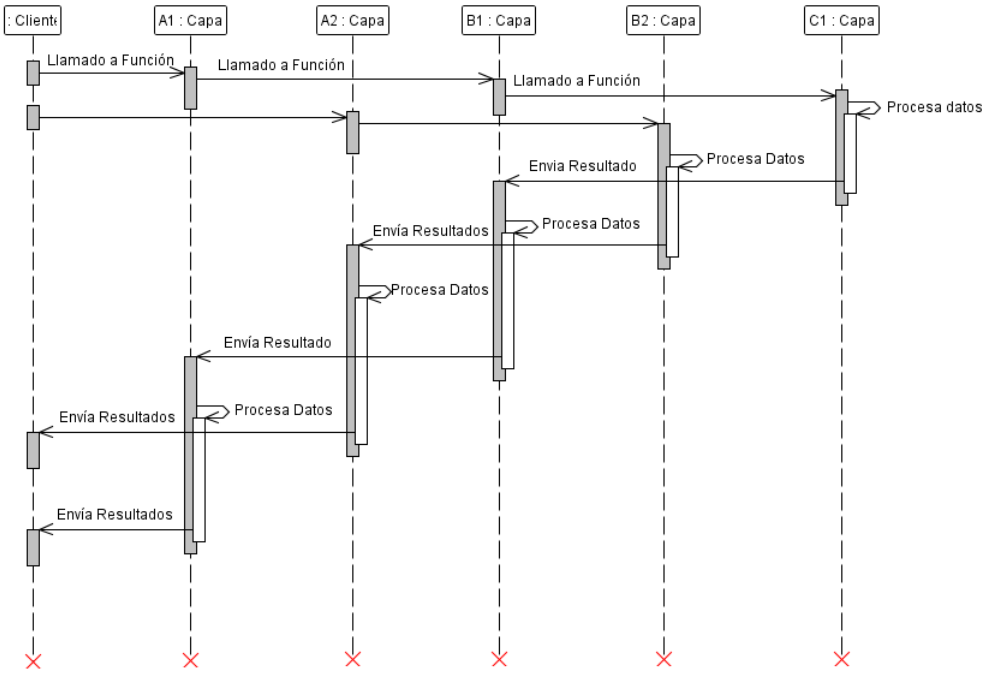


Figura 2.8: Diagrama de Secuencia del Patrón Parallel Layers

2.2.3. Communicating Secuential Elements

Este patrón es usado cuando el diseño de un problema puede ser definido en términos de un paralelismo de datos. La misma operación es realizada simultáneamente sobre diferentes datos ordenados. Las operaciones de cada componente representan un resultado parcial de los componentes vecinos. Usualmente, este patrón se presenta como una red o estructura lógica con base en el orden de los datos [8].

- *Estructura*

  En este patrón, la misma operación es aplicada simultáneamente a diferntes piezas de datos. Sin embargo, la operación en cada elemento depende de un resultado parcial de operaciones en otros componentes. La estructura de la solución resulta en una estructura lógica ordenada, generada a partrir de la estructura de datos del problema. Sin embargo, la solución es presentada como un red de e-lementos que en general siguen la forma impuesta por la estructura. Componentes idénticos existen y procesan simultáneamente durante el tiempo de ejecución (Figura 2.9).

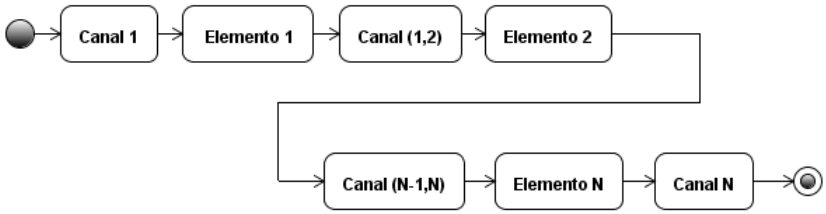


Figura 2.9: Diagrama de Objetos del Patrón Communicating Secuential Elements

- *Componentes*

  - **Elementos secuenciales**. Tiene el papel de un elemento de procesamiento, el cual es desempeñar un conjunto de operaciones en datos locales y proveer un interfaz general para enviar y recibir mensajes.
  - **tuboes de comunicación**. Como su nombre lo indica, su papel es representar un medio para enviar y recibir datos entre los elementos de procesamiento y sincronizar la comunicación entre ellos.

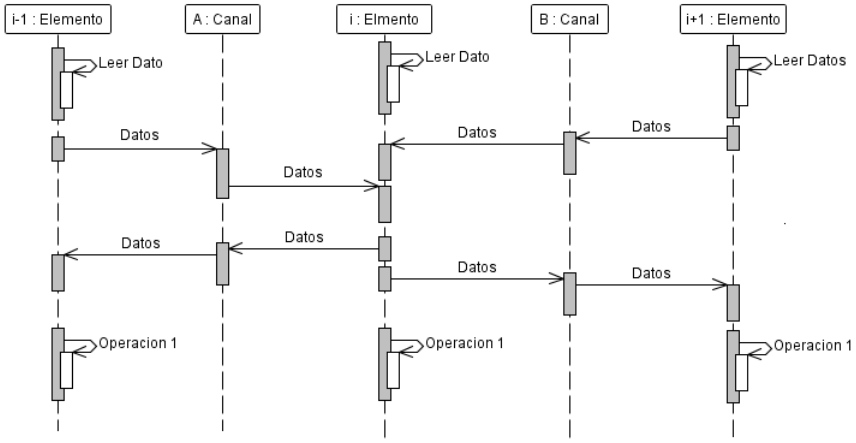


Figura 2.10: Diagrama Secuencial del Patrón Communicating Secuential Elements

2.2.4. Manager-Workers

Este patrón es considerado una variante del patrón Maestro-Esclavo para sistemas paralelos. Introduce un paralelismo de tareas, donde la operación es desempeñada sobre datos ordenados. Cada componente desempeña la misma operación, independiente de la actividad de procesamiento de los otros componentes. Diferentes grupos de datos son procesados simultáneamente. [10].

- *Estructura*

  El patrón *Manager-Workers* para programación paralela es usado cuando el diseño del problema puede ser comprendido en términos de paralelismo de tareas. Este patrón propone una solución en la cual la misma operación puede ser realizada simultánea e independientemente sobre diferentes piezas de datos.

  La pieza principal de esta estructura es denominada *manager* y realiza la tarea de preservar el orden de los datos y controlar el grupo delementos de procesamiento denominados *workers*. Usualmente se maneja un solo manager para un grupo delementos worker idénticos, que existen y procesan datos simultáneamente durante el tiempo de ejecución. Las operaciones realizadas por cada worker son aplicadas simultáneamente y de manera independiente a las de los otros workers. La solución finalmente puede verse como una red centralizada delementos que siguen la estructura manager-workers (Figura 2.11).

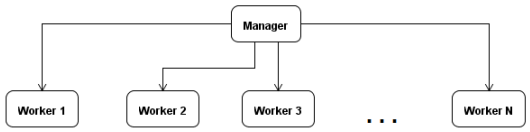


Figura 2.11: Diagrama de Objetos del Patrón Manager-Workers

- *Componentes*

  - **Manager**. La responsabilidad de un *manager* es crear un número determinado de workers, para particionar el trabajo através de ellos y agrupar el resultado general.
  - **Worker**. La responsabilidad de un *worker* es implementar el cómputo realizando un conjunto de operaciones requeridas por el manager.

- *Comportamiento*

  A continuación se muestra en un sencillo escenario el funcionamiento general de este patrón arquitectónico. En este ejemplo, cada worker realiza la misma operación sobre las piezas de datos que le son proporcionadas. Tan pronto como el trabajo de procesamiento se termina, envía el resultado al manager y solicita más datos. La comunicación entre workers no esta contemplada.

  1. Todos los elementos del sistema son creados y esperan hasta que algún cómputo sea requerido por el manager. Cuando los datos le son proporcionados al manager, éste los divide y envía las piezas de datos a los workers que se encontraban en espera de ellos.

  2. Cada worker recibe los datos y comienza con el proceso de ejecutar la operaciones sobre ellos. Al terminar regresa el resultado al manager y solicita a esta mas datos. Si existen datos para ser procesados, el proceso se repite.

3. El manager por lo regular satisface las peticiones de datos de los workers o recibe resultados parciales de éstos. Una vez que todas la piezas han sido procesadas, el manager ensambla el resultado a partir de los resultados parciales enviados y el programa termina (Figura 2.12).

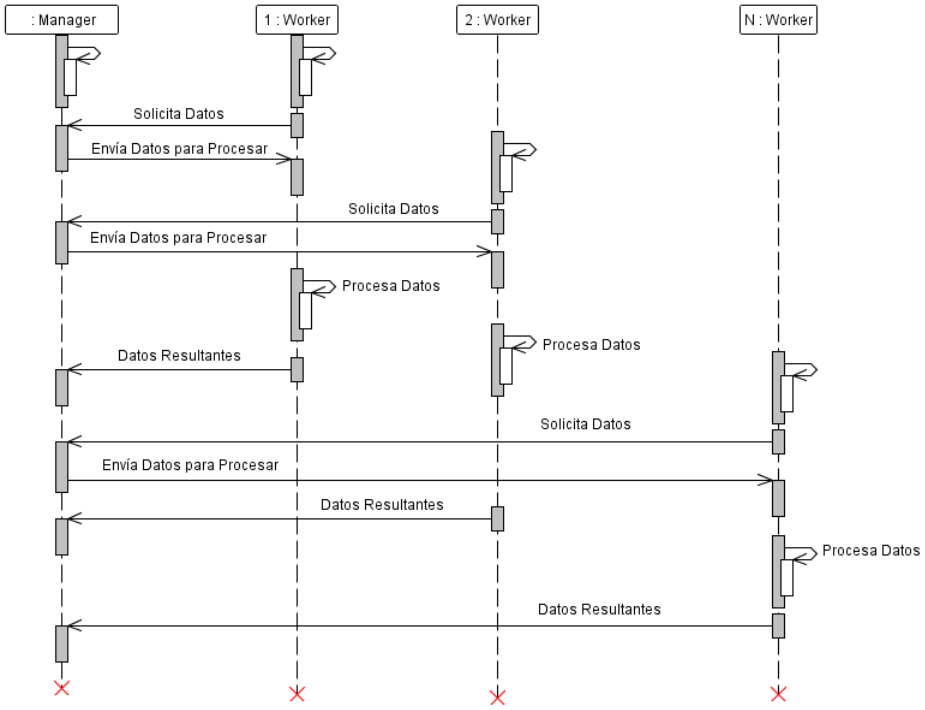


Figura 2.12: Diagrama de Secuencia del Patrón Manager-Workers

2.2.5. Shared Resource

Este patrón es usado cuando el diseño de un problema puede ser visto en términos de paralelismo de tareas. Este patrón propone una solución en la cual diferentes operaciones son desempeñadas simultáneamente sobre piezas de datos en un recurso compartido [9].

- *Estructura*

  En este patrón, las diferentes operaciones son aplicadas simultáneamente a diferentes piezas de datos. Se basa únicamente en un recurso compartido como estructura central que controla el acceso de diferentes componentes *Sharer* a la estructura central de datos. Usualmente el recurso compartido y otras componentes *sharer* existen simultáneamente y procesan durante todo el tiempo de ejecución(Figura 2.13).

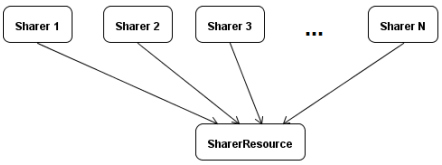


Figura 2.13: Diagrama de Objetos del Patrón Shared Resource

- *Componentes*

  - **Recurso Compartido**. La responsabilidad del recurso compartido es coordinar el acceso de los componentes *sharer*, preservando la integridad de los datos.
  - **Sharer**. La responsabilidad de una componente *sharer* es desempeñar un cómputo independiente hasta que se requiera del servicio de un recurso compartido. Entonces, el *sharer* tiene que contar con algúna restrición de acceso impuesta por el recurso compartido. Mientras el cómputo que realicen sea independiente los componentes pueden ejecutarse en paralelo.

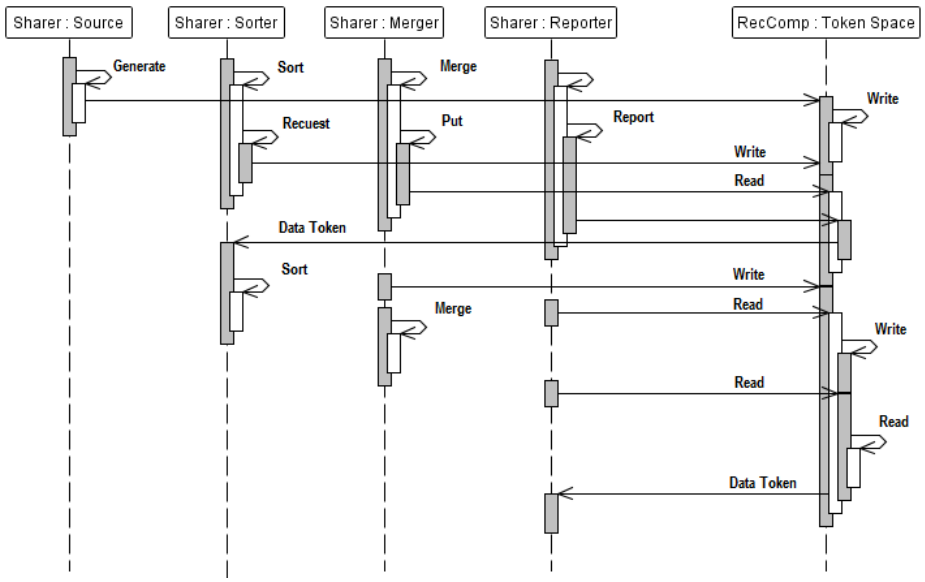


Figura 2.14: Diagrama de Secuencia del Patrón Shared Resource

2.3. Desempeño de sistemas paralelos

En la práctica, se ha demostrado una mejora considerable en la ejecución de un programa paralelo respecto a su contraparte secuencial. Considerando esto último, se han

generado una serie de métricas de desempeño para sistemas paralelos, algúnas dellas basadas en los tiempos de ejecución sobre en un número determinado de procesadores. Debido a esto, han surgido métricas como el propio tiempo de ejecución, el speedup. la eficiencia y la fracción serial[2].

- Tiempo de procesamiento y de comunicación.

  Una de las principales caracteríasticas que se consideradan respecto al desempeño de sistemas paralelos es el tiempo de ejecución de un programa. Con un sistema paralelo se pretende que tal tiempo sea disminuido en proporción al tiempo utilizado en un sistema uniprocesador. En base a esta relación, el tiempo de ejecución refleja el desempeño global de un sistema.

  El tiempo de ejecución de un programa en un sistema paralalo puede subdividirse en un componente secuencial y uno paralelo. También respecto a la granularidad, o incluso respecto a un tiempo de procesamiento y un tiempo de comunicación. Estos últimos forman la base sobre la que se sustenta la mayoría de las métricas de desempeño de un sistema paralelo.

  Una aplicación paralela trivial es aquélla que no requiere comunicación entre procesadores. El tiempo de ejecución de una aplicación de este tipo tiene un comportamiento completamente lineal. Sin embargo, en la realidad las aplicaciones paralelas no son triviales, pues la comunicación entre procesadores es inevitable en la mayoría de los casos. Al tiempo de retraso introducido por la comunicación entre procesos se le conoce como “retraso de comunicación” (comunication delay), y se genera a partir de la espera en algúno de los procesadores al intentar una comunicación.

  En contraste con el retraso de comunicación, existe el concepto de “tiempo de procesamiento”, el tiempo efectivo que toma a un procesador realizar las tareas que le son asignadas, independientemente de las comunicaciones.

  El desempeño de un sistema paralelo depende entonces de ambos tiempos, encontrándose un punto medio en función de la cantidad de procesamiento realizado por procesador y la comunicación entre ellos. Para obtener un reflejo del desempeño de un sistema paralelo, se utilizan una serie de métricas basadas principalmente en el tiempo de procesamiento global y el número de procesadores[2].

- Speedup

  Es posble categorizar diferentes configuraciones paralelas respecto a una aplicación determinada utilizando exclusivamente el tiempo de ejecucion como parámetro de comparación. Sin embargo, es importante realizar una evaluación más objetiva del desempeño. Tal medición puede ser el speedup.

El speedup es una relación del tiempo de ejecución de un programa ejecutándose en un solo procesador sobre el tiempo de ejecución del mismo programa ejecutándose en n procesadores. Esta relación puede obtenerse de la siguiente forma:

$$s(n) = \frac{T(1)}{T(n)}$$

donde n es el número de procesadores utilizados, T(1) es el tiempo de ejecución en un solo procesador y T(n) es el tiempo de ejecución en n procesadores.

El speedup es una medida del desempeño en función únicamente del número de procesadores utilizados en la ejecución de un programa y del tiempo de ejecución. No considera otros factores que influyen sobre el desempeño, como la topología utilizada, el balance de trabajo, la granularidad, etc[2].

- Eficiencia

Otra medida importante del desempeño de la programación paralela ligada directamente con el tiempo de ejecución y el speedup es la eficiencia. La medida de la eficiencia de un sistema paralelo se asocia a la idea que "n trabajadores deben hacer el trabajo en una fracción $\frac{1}{n}$ del tiempo que la lleva a un solo trabajador". La relación matemática que describe ésto es:

$$E(n) = \frac{T(1)}{nT(n)} = \frac{S(n)}{n}$$

en donde n es el número de procesadores utilizados, T(1) es el tiempo de ejecución del programa en un solo procesador, T(n) es el tiempo de ejecución del programa en n procesadores, y S(n) es el speedup.

Es importante notar que el valor de la eficiencia refleja el aprovechamiento de los recursos de hardware del sistema. Por lo tanto, el valor siempre es inferior a uno, ya que el tiempo de ejecución de un programa paralelo se ve afectado por el tiempo total de comunicaciones entre los procesadores, excepto en el caso de un sistema paralelo trivial[2].

2.4. Resumen

La computación paralela es una técnica en la que se ejecutan dos o más instrucciones simultáneamente. Las instrucciones pueden ser agrupadas en bloques con el fin de realizar algúna acción en común, generando lo que se conoce como procesos.

Un programa paralelo o distribuido, se caracteriza por tener dos o más procesos que se ejecutan simultáneamente en un tiempo determinado y que cooperan entre sí para realizar una tarea.

Los procesos que componen este tipo de programas se ejecutan en varios procesadores conectados entre sí mediante una red de comunicación, siguiendo un esquema de paralelismo espacial. Si la red se forma por conexiones dentro de una sola computadora, se considera un sistema de programación paralela; por otro lado, si la red se forma de conexiones entre diferentes computadoras, se considera un sistema de programación distribuido.

Dada la naturaleza de los componentes de procesamiento de estos sistemas, es posible manejar otro criterio de clasificación. Generalmente, los componentes de los sistemas paralelos desempeñan actividades de coordinación y procesamiento. Considerando solo las características de procesamiento de los componentes, los sistemas paralelos pueden ser clasificados en *homogéneos* y *heterogéneos*, de acuerdo a la naturaleza de sus componentes.

Existen diferentes tipos de paralelismo, los cuales se distinguen por la caractéristica del sistema en la cual están especializados, estos son:

- Paralelismo Funcional
- Paralelismo de Datos
- Paralelismo de Tareas

Al mapear un problema a una aqruitectura paralela, es necesario primero dividir el problema en segmentos que se ejecuten en paralelo, luego, determinar la manera como los procesadores se comunican y sincronizan entre si. Dentro del problema de asignación de procesos a los procesadores, se debe tomar en cuenta aquellos elementos de la programación paralela que se relacionan entre sí y determinan el nivel de paralelismo a utilizar. Estos son la granularidad y él balance de trabajo.

Los patrones arquitectónicos para programación paralela son definidos y clasificados de acuerdo a las restricciones de orden de datos y operaciones, así como también de la naturaleza del procesamiento de sus componentes. A continuacion se listan los patrones arquitectónicos explicados en este capítulo.

- parallel and Filter
- Parallel Hierarchies
- Communicating Sequential Elements

- Manager-Workers
- Shared Resource

Las restricciones de orden definen la forma en la cual el sistema paralelo se desempeña y también el impacto en el diseño de software. Siguiendo esto, es posible considerar que la mayoría de la aplicaciones paralelas esten clasificadas dentro de una de las 3 formas de paralelismo:

- Paralelismo Funcional
- Paralelismo de Datos
- Paralelismo de Tareas

Con todo lo anterior es posible tener un panorma general de los componentes de un sistema paralelo y todo aquéllo que se debe tomar en cuenta para poder implementarlo y evaluarlo.

Capítulo 3

Trabajo Relacionado

En este capítulo, se presentan algunas de las contribuciones realizadas en el campo de los sistemas de cómputo paralelos en materia de evaluación de desempeño. Se pretende analizar algunos trabajos para conocer los aspectos que se consideran fundamentales para la medición de desempeño; también, la forma en cual realizan sus estimaciones, ya sea con algún bechmark u otra herramienta que permita evaluar el desempeño de un sistema cómputo paralelo. Todo lo anterior con el fin de poder proponer semejanzas y diferencias con la modelación prpuesto en esta tesis.

3.1. Aproximación

Como una primera aproximación a los métodos de evaluación de desempeño, es necesario conocer algunos de los aspectos que se deben tomar en cuenta para evaluar el desempeño. La primera de éstas, es saber en términos de qué características del sistema se esta evaluando el desempeño. Para esto tendremos que definir una métrica.

Una métrica es un conjunto de criterios medibles, que serán considerados para cuantificar la evaluación de desempeño que realizamos. Diferentes métricas pueden resultar en evaluaciones de desempeño totalmente diferentes. Conocer la métrica usada, sus relaciones y sus efectos sobre los parámetros del sistema es importante.

En base a las características que se definan para evaluar el desempeño, se cuenta con la posibilidad de trabajar con alguna cierta metodología. Esto debido que en la actualidad ya existen formas sistemáticas que pueden ser usadas y que por sus antecedentes, arrojan resultados en buena medida confiables.

Una vez que se tienen definidas las características del sistema que serán medidas para determinar su desempeño, es necesario saber de qué forma podemos explotarlas y con

ello evaluar de una manera más precisa el desempeño. Con frecuencia un sistema de cómputo paralelo es diseñado para trabajar en un ambiente particular con cargas de trabajo determinadas. Por lo tanto, seleccionar la carga de trabajo apropiada es el siguiente paso dentro de la evaluación de desempeño.

La tarea de definir la carga de trabajo usada dentro del sistema no es fácil. Es por eso que existen toda una gama de técnicas de análisis estadísticos que son usadas para caracterizar la carga de trabajo, como ejemplos tenemos[6]:

- Promedio
- Análisis de distribución
- Análisis de componentes principales
- Clustering
- Modelos de Markov

Dichas técnicas son comúnmente usadas en sistemas secuenciales y paralelos. Por si fuera poco, en la actualidad existen nuevas técnicas y métricas que tienen como objetivo los sistemas de cómputo paralelo, tal es el caso de las siguientes:[6]

- Gráfica de tareas
- Perfiles y forma
- Fase de estacionaría y transicional
- Ejecución de tareas
- Trabajo de procesador
- Aseguramiento
- Modelos Analíticos
- Simulación

La habilidad y precisión de cada técnica depende sobre todo del contexto durante el cual el sistema desarrolla su ciclo de vida.

La evaluación de desempeño no se tiene únicamente para sistemas ya construidos y funcionales. Dependiendo de la etapa de desarrollo del sistema puede considerarse ciertas mediciones o predicciones desempeño.

En un diseño temprano, cuando el sistema no ha sido construido, es posible asegurar un cierto desempeño, es decir, es posible realizar un modelo analítico simple o de simulación. Tan pronto como el diseño del sistema avanza, se obtienen más detalles y conocimientos sobre el sistema. Por tal motivo puede ser empleada una simulación o alguna técnica de modelo analítico más sofisticado. Finalmente, cuando el sistema ha sido completado y construido, podemos evaluar su desempeño. Detrás de estos métodos,

existen métricas fundamentales como el speedup, eficiencia y escalabilidad, pueden ser usadas de manera directa para el análisis de desempeño.[6]

3.2. Métodos de evaluación de desempeño

3.2.1. Modelo Determínistico

En estadística, un fenómeno determinístico es aquél que obtiene siempre el mismo resultado bajo las mismas condiciones iniciales.[5]

Este tipo de modelos surgen como una forma de aproximar el desempeño de un sistema paralelo, estandarizando los tiempos que consumen para realizar ciertas acciones, ya sea comunicación entre componentes o tiempo de procesamiento; ya que al tener un sistema paralelo el tiempo de procesamiento, sincronización y comunicación de sus componentes no es constante, aún cuando los parametros de entrada para el sistema sean siempre los mismos. Las causas pueden ser muy diversas: el uso del procesador, la saturación del canal de comunicación, etc.
Es importante resaltar que a diferencia de los modelos estocásticos, los modelos determinísticos siempre devuelven un valor constante para un conjunto de parametros idénticos, lo cual no ocurre con los estocásticos.

En el primer trabajo analizado [5], se presenta un modelo de evaluación de desempeño híbrido, basado en dos sub-modelos: un modelo estocástico, con el cual modelan los retrasos aleatorios dentro de la comunicación entre procesos, y un modelo determinístico, que se usa para modelar los tiempos de las componentes de procesamiento, teniendo así una propuesta integral para evaluación de desempeño de sistemas paralelos.

Dentro del modelo determinístico, es necesario identificar los requerimientos de CPU con el fin de que sean modelados como constantes. Además, es necesario realizar un monitoreo de las tareas que se ejecutan, con el fin de conocer el tiempo que destinan a procesar.

Para lograr esto, se presenta un algoritmo de monitoreo de tareas en forma de funciones del tipo: **Sched(ready_task_list,p)**, con el cual se genera una lista de tareas que pueden ser ejecutadas y una lista de procesos P, los cuales, de estar disponibles, se encargarán de ejecutar la tarea que se les asigne (ver tabla 3.2.1).

N	Número de tareas (Se asume que la tarea 1 es la única tarea sin predecesores)
Parentesis(i)	Lista de predecesores directo para cada tarea i, $1 \leq i \leq N$
T_i	Requerimientos de CPU para cada tarea i, $i \leq i \leq N$
$M_{i,j}$: $1 \leq j \leq N_p aram$	Parámetro de recurso usado para cada tarea i, $1 \leq i \leq N$
Sched(ready_task_ list, Idle-Proc)	Función de monitoreo: Especifíca cual de las tareas en la lista: ready_task_ list, de ser posible, será ejecutada por el proceso Idle-Proc
P	Número de Procesadores

Cuadro 3.1: Entradas del modelo determinístico

En este modelo se manejan los costos de comunicación y contención de recursos como parte de un modelo estocástico. Además, se propone que para un programa paralelo y un conjunto bien definido de datos, se tiene que [5]:

- Los tiempos de ejecución de tareas son cada uno determinísticos y conocidos.

- Los costos de comunicación y contención serán insignificantes.

Al cumplirse los requisitos anteriores, para un número particular de procesadores, se tiene una única secuencia de ejecución, lo que implica un único tiempo de inicio y fin para cada tarea con relación al inicio del programa.

Se incluye también, un simple algoritmo que puede ser usado para establecer la secuencia de ejecución y entonces, obtener el tiempo que tarda el programa. Este algoritmo se muestra en la figura 3.2.

Enradas
Entradas listadas en la tabla 3.2.1 excepto aquellos definidos como $\{M_{i,j}\}$
Algoritmo
$T_{remain}(i) \leftarrow t \epsilon T_i, 1 \leq i \leq N$
$E_{set} \leftarrow \{Task1\}$
Do N times $\{$
next_task_done $\leftarrow t \epsilon E_{set} : T_{remain}(t)$ es minimo.
Delete next_task_done from E_{set}
$T_{elapse} \leftarrow T_{remain}$ (next_task_done)
$T_{total} \leftarrow T_{total} - T_{elapse}$
$T_{remain}(t) \leftarrow T_{remain}(t) - T_{elapse} \forall t \epsilon E_{set}$
For all inmediate successors c of task next_task_done
If all parents of C are done
Insert C in ready_task_list
For each free process P
if sched(read_task_list, P) finds a task t to execute
Add t to E_{set}
$\}$
Total Program Execution $Time = nT_{total}$

Cuadro 3.2: Modelo determinístico Básico

En esencia, la aplicación de tal algoritmo consiste en repetir los siguientes cuatro pasos N veces [5]:

- Elimina tareas con requerimientos mínimos de CPU de un conjunto E.
- Actualiza el tiempo restante de otras tareas en un conjunto E.
- Encuentra alguna tarea de lectura para agregarla a una ready_task_list (lista de tareas realizadas).
- Para cada proceso inactivo, selecciona una tarea de la lista definida en el punto anterior para asignarla, monitorearla y agregarla al conjunto E.

Una vez generado un modelo inicial, se evalúa con diferentes programas (como un simulador de particulas, un programa de cálculo de raíces de polinomios, entre otros) con el fin de tener un punto de partida y comparación con otros modelos, y así mostrar la exactitud del modelo presentado.

3.2.2. El modelo Estocástico

Se denomina estocástico a aquél sistema que funciona, sobre todo, por el azar. Las leyes conocidas de causa-efecto no explican cómo actúa el sistema de manera determinista, sino en función de probabilidades.

Un modelo estocástico de evaluación de desempeño para un sistema paralelo, consiste en un modelo algorítmico similar al determinístico, pero con una importante diferencia. Dentro de este modelo se manejan variables aleatorias con el fin de acercarse al

comportamiento real de ciertas funciones específicas del sistema paralelo, como lo es, el tiempo que las componentes de procesamiento emplean en comunicación. [5]

La exactitud de este tipo de modelos depende en gran medida de que tan exactas han sido estimadas las variables aleatorias del sistema que se modelan. En el caso del trabajo analizado [5], se trata de los costos de sincronización, los cuales a su vez dependen de qué tan precisa sea la distribución de tiempos de desempeño de cada proceso, entre los puntos de sincronización que son modelados. Esto, aunado a la necesidad de complejas eurísticas, hace en cierta medida complejo el modelo presentado.

3.3. Semejanzas con el trabajo relacionado

En [6] se hace un análisis de las características que se desea medir para definir el desempeño del sistema. En este caso se toma la comunicación como la característica medible y cuantificable en base a la cual se define el desempeño del sistema. Entre las principales razones por la cual se toma esta medida, es debido a que la mayoría de las evaluaciones de desempeño toman como base el tiempo de procesamiento y dejan de lado esta características fundamental.

De igual forma, se define una modelo en base al cual se debe evaluar el desempeño. Se toma un modelo analítico híbrido, el cual evalúa el tiempo que el sistema invierte en comunicar sus procesos, definiendo con base en esto el desempeño total del sistema.

Respecto a [5], en esta tesis se trabaja sobre un modelo híbrido para evaluar el desempeño de un sistema paralelo. Por una parte, se toman los tiempos de procesamiento como constantes, y por otra, los tiempos de comunicación y sincronización como aleatorios (con base en la aleatoriedad natural del sistema). Esto, genera un modelo determinístico para el tiempo de procesamiento y una especie de modelo estocástico para las comunicaciones.

La razón de tener un modelo híbrido para evaluar el desempeño es sencilla: existen caracteristícas que definen al sistema de cómputo paralelo, que pueden influir en el resultado final de manera considerable, haciendo que el desempeño del sistema sea calculado incorrectamente.

Por ejemplo: Si únicamente la evaluación de desempeño se basa en el tiempo de procesamiento, despreciando la comunicación, es posible que el sistema a evaluar posea una granularidad fina, esto es, el sistema puede ocupar un porcentaje pequeño del tiempo total de ejecución, procesando datos y el resto en comunicación, haciendo que la evaluación realizada sea imprecisa en un porcentaje considerable.

Esta idea es muy parecida a la que se plantea dentro del trabajo analizado[5], en el cual se usa un modelo deterministíco para modelar el procesamiento del sistema sistema paralelo, estableciendo como datos conocidos y constantes los tiempos de procesamiento de cada componente, y como estocásticos los tiempos de comunicación entre los componentes de procesamiento.

3.4. Diferencias con el trabajo relacionado

De igual forma que las ideas propuestas en el articulo [6], en esta tesis se toma como primera medida el tratar de definir un modelo en base al cual sea posible evaluar el desempeño de un sistema paralelo, dando la importancia que requiere a la comunicación. Para ello se toma un modelo analítico híbrido, el cual evalúa el tiempo que el sistema invierte en comunicar sus procesos, definiendo con base en esto el desempeño total del sistema.

En [6] se muestran diferentes formas de evaluar el desempeño de un sistema paralelo, a través de métodos como:

- Gráfica de tareas
- Perfiles y forma
- Ejecución de tareas
- Trabajo de procesador
- Aseguramiento
- Modelos Analíticos
- Simulación

Sin embargo, en la aproximación que presenta para cada uno de ellos, no otorga gran importancia al tiempo que el sistema invierte en comunicar y coordinar sus procesos, el cual es el tema fundamental de esta tesis.

Con respecto a [5], existe una diferencia importante entre la evaluación de desempeño que se muestra y la que se presenta en esta tesis. Esta radica en que el trabajo revisado emplea el modelo híbrido como una forma de predecir el desempeño de un sistema paralelo. Sin embargo, en esta tesis el modelo híbrido generado se emplea para evaluar el impacto que tiene la coordinación en el desempeño total del sistema.

Dicho de otra forma, mientras en [5], se parte de la estimación del tiempo de procesamieto y de comunicación para lograr predecir el tiempo total de ejecución del sistema. Es importante señalar que para simplificar los cálculos, no se consideraron las incertidumbres o alteraciones en cada ejecución del sistema.

$$T_{procesamiento} + T_{comunicacion} = T_{total}$$

En esta tesis, se parte de conocer el tiempo total de ejecución del sistema y mediante un modelo determinístico que estima el tiempo que el sistema dedica al procesamiento, conocer el impacto que tiene la comunicación en el tiempo total de ejecución.

$$T_{total} - T_{procesamiento} = T_{comunicacion}$$

Es importante señalar que el uso de un modelo estocástico en esta tesis para la comunicación entre componentes del sistema paralelo, es con base en el hecho de que la comunicación se presenta como fénomeno estocástico de manera natural dentro del sistema paralelo.

Otra de las principales diferencias con [5] se encuentra en el uso de un patrón de arquitectura paralela, ya que ninguno de los trabajos presentados, usa o define algún patrón para diseñar los sistemas paralelos que prueba. Es una importante diferencia, ya que con este adelanto, se tiene de una manera clara, los puntos de comunicación, sincronización y procesamiento del sistema, facilitando con esto la evaluación del desempeño.

3.5. Resumen

En la actualidad, la mayoría de los modelos de evaluación de desempeño de sistemas paralelos toman como base el tiempo que el sistema dedica al procesamiento de datos, despreciando o minimizando los tiempos de comunicación. Sin darse cuenta que actualmente este factor tiene un gran impacto en los sistemas paralelos. En contraste, el modelo propuesto en esta tesis pretende tomar como constante los tiempos que las componentes del sistema dedican al procesamiento, logrando con esto estimar el tiempo que el sistema invierte realmente en comunicar y sincronizar los procesos.

A través de las ideas planteadas en este capítulo, es posible identificar alternativas de como ha sido resuelto el problema de evaluación de desempeño en sistemas paralelos. En particular, aquellas soluciones que emplean alguno de los siguientes modelos:

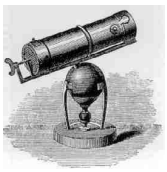
- Modelo deterministíco
- Modelo estocástico.

Con base en los trabajos analizados, en esta tesis se plantea un modelo de evaluación de desempeño hibrido basado en estos dos submodelos.
Por un lado tenemos un modelo deterministíco que se usa para modelar los tiempos que los componentes invierten en procesamiento y por otro lado como modelo estocastíco, se toma la aleatoriedad natural dentro de los mecanismos de comunicación.

Ambos modelos nos sirven para conocer maneras en las cuales se puede llevar a cabo la evaluación de desempeño o que caractaristícas del sistema evaluar y con ello, poder tener un punto de comparación integral sobre el sistema.

Capítulo 4

Caso de estudio: Problema de los N cuerpos



En esta sección se define el problema de los N cuerpos, sus características principales y una aproximación de la forma en la cual es posible programar una solución. La solución definida en este capítulo es implementada más adelante sobre un patrón de diseño paralelo y actúa como objeto de estudio para desarrollar el tema principal de esta tesis: **Conocer el impacto de la coordinación en el desempeño total de un sistema paralelo** .

4.1. Problema de los N Cuerpos

El problema de los N-cuerpos es un problema de mecánica que tiene como base las tres leyes de Newton. Cabe resaltar que Newton fue el primero en formular el problema de los N-cuerpos de una forma precisa, enuncindolo como sigue:

> Dadas en un instante las posiciones y las velocidades de tres o más partículas que se mueven bajo la acción de sus atracciones gravitatorias mutuas, siendo conocidas las masas de las partículas, es posible calcular sus posiciones y velocidades para otro instante [4].

Para poder comprenderlo mejor, comenzaremos reduciendo este problema únicamente a dos cuerpos. Con esto, el problema se simplifica a determinar el movimiento de dos partículas puntuales que interactúan entre sí. Los ejemplos comunes incluyen la Luna orbitando la Tierra (en ausencia del Sol), un planeta orbitando una estrella, dos estrellas que giran en torno al centro de masas (estrella binaria), y un electrón orbitando en torno a un núcleo atómico.

De esta forma, podemos redefinir el problema de los N cuerpos, como el problema de calcular las trayectorias de n cuerpos los cuales interactúan únicamente através de sus fuerzas gravitacionales dos a dos.

Por lo tanto, para encontrar la solución de este problema, es necesario dados dos cuerpos, conocer dos caracteristícas. La primera es la fuerza de atracción que ejercen uno sobre el otro y la segunda, consiste en calcular la nueva posición de los cuerpos en un tiempo determinado a partir de la fuerza de atracción que ejercen entre ellos.

A continuación, se describe la forma en la cual es posible calcular la fuerza total que ejercen todos los cuerpos sobre uno determinado y con base en esto calcular la posición de todos los cuerpos del sistema en un instante de tiempo posterior.

4.1.1. Cálculo de las Fuerzas

Para el cáculo de las fuerzas, se considera que cada cuerpo es interpretado como un punto dentro de un espacio de tres dimensiones. El estado de un cuerpo puede ser definido por su masa **m** y tres vectores que representan: su posición **r**, su velocidad **v** y la fuerza total **f**. Con base en ellos, es posible calcular la fuerza de atracción que ejerce este cuerpo sobre otros.

La figura 4.1 muestra 2 cuerpos p_i y p_j, con una masa m_i y m_j y posiciones r_i y r_j, las cuales son relativas a un origen O.

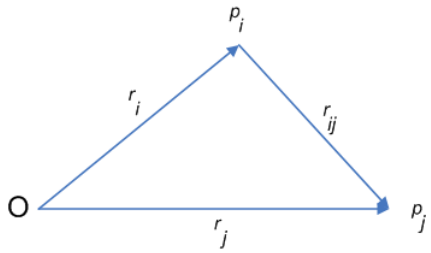


Figura 4.1: Dos cuerpos en el espacio

De acuerdo con Newton, el cuerpo p_j, atrae a p_i, con una fuerza f_{ij}. La magnitud de la fuerza es proporcional a la masa de cada cuerpos e inversamente proporcional a el cuadrado de la distancia entre ellos:

$$|f_{ij}| = \frac{Gm_i m_j}{|r_{ij}|^2}$$

En ella G es la constante de gravitación unversal. La distancia $|r_{ij}|$ es la longitud del vector

$$\overline{r_{ij}} = \overline{r_j} - \overline{r_i}$$

La ley de la gravitación universal de Newton, puede entonces describirse como un vector:

$$\overline{fij} = |f_{ij}|\overline{e_{ij}}$$

Donde e_{ij} es un vector unitario que se encuentra en la misma dirección de r_{ij}.

$$\overline{e_{ij}} = \frac{\overline{r_{ij}}}{|r_{ij}|}$$

Con esta ecuación, es posible definir el algoritmo que se muestra a continuación y que se encarga de calcular la fuerza que un cuerpo determinado ejerce sobre otro.

```
1    private double[] fuerza(Cuerpo pi, Cuerpo pj){
2          double g = 667 * Math.pow(10,-11);
3          double[] rij = u.resta(pi.getr(), pj.getr());
4          double rm = u.longitud(rij);
5          double fm = (g * pi.getm() * pj.getm())/Math.sqrt(rm);
6          double[] eij = u.productoE(rij,1/rm);
7          double[] fuerza = u.productoE(eij, fm);
8          return fuerza;
9       }
```

Por otra parte, sabemos que el cuerpo p_i atrae a p_j con una fuerza f_{ji} de la misma magnitud y en sentido contrario de f_{ij}:

$$f_{ji} = -f_{ij}$$

Entonces con base en la tercera ley de Newton la fuerza total f_i que actua sobre el cuerpo p_i, es la suma de todas las fuerzas sobre p_i, que ejercen los otros $n-1$ cuerpos.

$$f_i = \Sigma_{j=1}^{n-1} fij$$

Este proceso debe repetirse para cada cuerpo del sistema, con ello tendremos las fuerzas que el sistema ejerce sobre cada una de los cuerpos que lo componen. A continuación se presenta el algoritmo correspondiente a esta idea.

```
1       public Cuerpo fuerzaT(Cuerpo[] sist, int index){
2          int tcps = sist.length;
3          Cuerpo it = new Cuerpo();
4          Cuerpo it2 = sist[index];
5          double[] fij = new double[3];
6
7          for(int i=0; i<tcps; i++){
8              if(i != index){
```

```
                    Cuerpo sec = sist[i];
                    double[] tmpF = fuerza(it2,sec);
                    fij = u.suma(fij, tmpF);
                }
            }
            double[] rs = u.suma(it2.getf(),fij);
            it.setf(rs);
            it.setm(it2.getm());
            it.setr(it2.getr());
            it.setv(it2.getv());
            return it;
        }
```

4.1.2. Cálculo de posiciones

La acelearación a_i de un cuerpo p_i es determinada por su masa m_i y la fuerza total f_i que actúa sobre él, de acuerdo con la ley de Newton:

$$a_i = \frac{f_i}{m_i}$$

Durante un pequeño intervalo Δt, la aceleración a_i es aproximadamente constante. En consecuencia, la velocidad del cuerpo incrementa por

$$\Delta v_i = a_i \Delta t$$

Al mismo tiempo la posición r_i de el cuerpo incrementa a razón de:

$$\Delta r_i = \int_0^{\Delta t} (v_i + a_i t) dt = v_i \Delta t + 0{,}5 a_i \Delta t^2.$$

En otras palabras, el cálculo para encontrar la posición de un cuerpo determinado puede ser visto a través de la siguiente ecuación:

$$\Delta r_i = (v_i + 0{,}5 \Delta v_i) \Delta t$$

De esta forma, es posible crear el siguiente algoritmo que nos permite calcular la posición de un objeto del sistema en algún instante determinado.

```
    public Cuerpo mueveC(Cuerpo pi, double incremento){
        double[] ai = u.productoE(pi.getf(), 1/(pi.getm()));
        double[] dvi = u.productoE(ai, incremento);
        double[] dri = u.productoE(u.suma(pi.getv(), u.
            productoE(dvi, .5)), incremento);
        pi.setv(u.suma(pi.getv(), dvi));
        pi.setr(u.suma(pi.getr(),dri));
        return pi;
    }
```

Tomando los algoritmos anteriores para el cálculo de las fuerzas y para el Cálculo de posiciones, es posible consruir las rutinas de procesamiento principales de un sistema que **aproxima una solución** para el problema de los N-cuerpos.

La forma de extraporlarlo para el calculo de N-cuerpos, resulta de aislar la fuerza que ejercen los cuerpos dos a dos en un instante específico. De esta forma al comparar un cuerpo determinado con todos los demás, sabremos la fuerza que ejerce el sistema total sobre sobre este cuerpo y por lo tanto la posición que tendrá en el siguiente instante de tiempo.

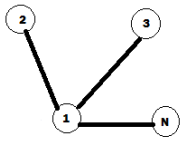


Figura 4.2: Estrategía para el cálculo de fuerza

Realizando el proceso anterior para cada uno de los cuerpos, es posible calcular las nuevas posiciones de todos los elementos que componen nuestro sistema.

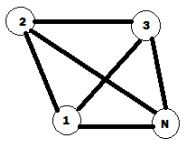


Figura 4.3: Cálculo de N-cuerpos con Distancias intermedias

4.2. Resumen

En este capítulo se plantean los cálculos principales para obtener una solución para el problema de los N-cuerpos, esta solución divide al problema en dos fases principales para cada uno de los cuerpos de nuestro sistema:

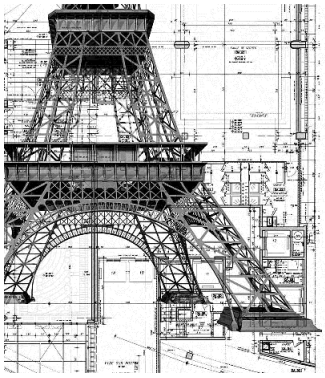
- Dado un cuerpo i, es necesario calcular la fuerza de atracción total que ejercen los $N - 1$ cuerpos sobre i, para ello se debe calcular por separado la fuerza de atracción que ejercen cada uno de los cuerpos del sistema sobre el cuerpo i y posteriormente sumarlas para encontrar un vector de fuerza total resultante.

- Una vez encontrado el vector de fuerza de atracción total para un cuerpo i, es posible calcular la nueva posición de este cuerpo en el siguiente intervalo de tiempo.

Estas dos fases se realizan a intervalos de tiempo constante para cada uno de los cuerpos que componen el sistema. Con ello, se calcula la posición de cada uno de los cuerpos en un tiempo determinado, generando así la solución para el problema de los N-cuerpos.

Capítulo 5

Diseñando un sistema paralelo



En este capítulo se define la forma de construir un programa de cómputo paralelo que sea capaz de resolver el problema de los N-cuerepos planteado en el capítulo 4. Esta solución es fundamental para le presente trabajo, ya que se toma como caso de estudio para conocer el impacto que tiene la coordinación dentro de un sistema de cómputo paralelo.

Es importante resaltar que para desarrollar la solución del problema de los N cuerpos, se tomo como base la implementación del problema de las N reinas [13]. Se respeto el patrón de diseño y se modificaron las clases de procesamiento y mensajes. Con el fin de tener una diseño más limpio del programa. Las clases que se crearon o redefinieron se agruparon en el paquete *ncuerpos*. Los otros paquetes necesarios para la ejecución de programa se conservaron intactos para mantener la integridad del patrón originalmente diseñado.

5.1. Estructura

Para elegir un patrón de diseño se debe tomar en cuenta la naturaleza del problema y su solución. Para el problem de N-Cuerpos, la solución consiste en encontrar las trayec-

torias de N-cuerpos que interactúan entre sí, con base en su fuerza de atracción. De esta forma, una buena aproximación para la implementación de una solución para este problema puede ser que cada uno de los cuerpos calcule su posición para el siguiente instante del tiempo con base en el estado actual del sistema. Con esta idea, es posible agregar un proceso que se encarge de dar a conocer las características del sistema a cada cuerpo que lo compone y de la misma forma, recolectar las posiciones de todos los cuerpos para un determinado instante del tiempo, logrando así tener un historial de las trayectorias de los cuerpos,lo cual es en esencia la solución del problema. Esta solución plantea la posibilidad de comprender el problema en términos de un paralelismo de tareas, en el cual, cada uno de los cuerpos conoce el estado actual de todo el sistema y con base en él puede calcular su posición para el siguiente instante de tiempo. Una de las características relevantes de esta aproximación es que las tareas son independientes y no requieren de comunicación entre ellas.

Entonces, es posible definir este proceso general que puede identificar un cuerpo dentro del sistema y darle a conocer el estado actual del mismo, con el fin de que este cuerpo de manera independiente pueda calcular su posición para el siguente instante de tiempo y lo regrese como resultado. Es decir, definir un proceso que se encargue de manejar las acciones de asignación de trabajo y recolección de reultados sobre los cuerpos del sistema. De esta forma, la implementación de un sistema que resuelva este problema debe constar de un proceso principal que envía a cada uno de los procesos independientes del sistema (trabajadores) el estado actual del mismo y el identificador del cuerpo sobre el cual se va a interactuar. Una vez enviada la información, cada uno de estos procesos debe devolver la nueva posición del cuerpo con el que se les indicó trabajar. Entonces este proceso principal se encaga de recolectar esta información y actualizar el sistema generando un nuevo estado que define la nueva posición de todos los cuerpos. Almacenando los estados del sistema y repitiendo este proceso, es posible definir las trayectorias de los N-cuerpos dentro del sistema.

Dada la forma en que se plantea la solución del problema, resulta conveniente elegir el patrón: **Manager Workers**, ya que, recapitulando, se ajusta mejor a nuestras necesidades y concepciones de la solución del problema.

> El patrón Manager-Workers para programación paralela es usado cuando el diseño del problema puede ser comprendido en términos de paralelismo de tareas. Este patrón propone una solución en la cual la misma operación puede ser realizada simultanea e independientemente sobre direntes piezas de datos. Es representado por un manager que preserva el orden de los datos y controla un grupo de elementos de procesamiento denominados workers. Éstos tiene el acceso a las diferentes piezas de datos al mismo tiempo. [10]

Tomando como punto de partida el patrón Manger Workers y apuntando hacia una implementación sobre un lenguaje Orientado a Objetos (Java), podemos agrupar los componentes del sistema a implementar en dos principales: un objeto NbodyWorkers y otro NbodyManager. Estos objetos están encargados de interactuar para procesar y coordinarse con el fin de encontrar la trayectoria de cada uno de los cuerpos que componen el sistema.La figura 5.1 muestra la estructrua básica del sistema.

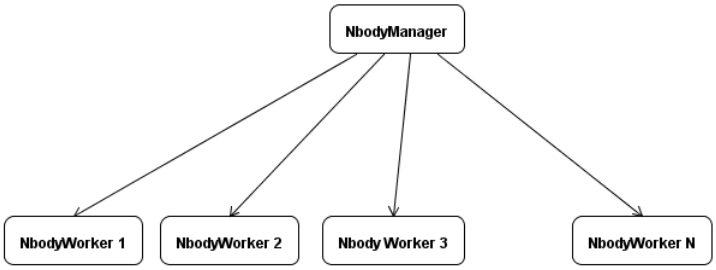


Figura 5.1: Esquema Manager Workers para el problema de los N-Cuerpos

Una vez definidos los componentes del sistema es necesario establecer las tareas asignadas a cada uno de estos. Estas tareas se muestran a continuación:

- **NbodyWorker** : Proceso que identifica un cuerpo dentro de nuestro sistema de N-cuerpos y se encarga de calcular su nueva posición a partir de las características de todos los demás cuerpos que componen el sistema en un momento determinado.

  Solo tiene definida una tarea, la cual consiste en calcular su nueva posición a partir del estado actual del sistema y notificarla al objeto NBodyManagaer.

- **NbodyManager** : Proceso que coordina y estructura la información que los procesos NbodyWorker envían con el fin de actualizar cada una de las posiciones de los cuerpos que componen el sistema.

  Tiene asignadas dos tareas fundamentales: la primera es enviar un mensaje a cada NbodyWorker con el estado astado actual del sistema, indicándole el identificador del cuerpo que representa con el fin de que calcule su nueva posición. La segunda tarea consiste de recibir y organizar las respuestas de cada NbodyWorker, generando el nuevo estado para el siguiente intervalo de tiempo, para tener una vista dinámica del comportamiento e interacción general. A continuanción se muestra el diagrama de secuencia correspondiente (Figura 5.2).

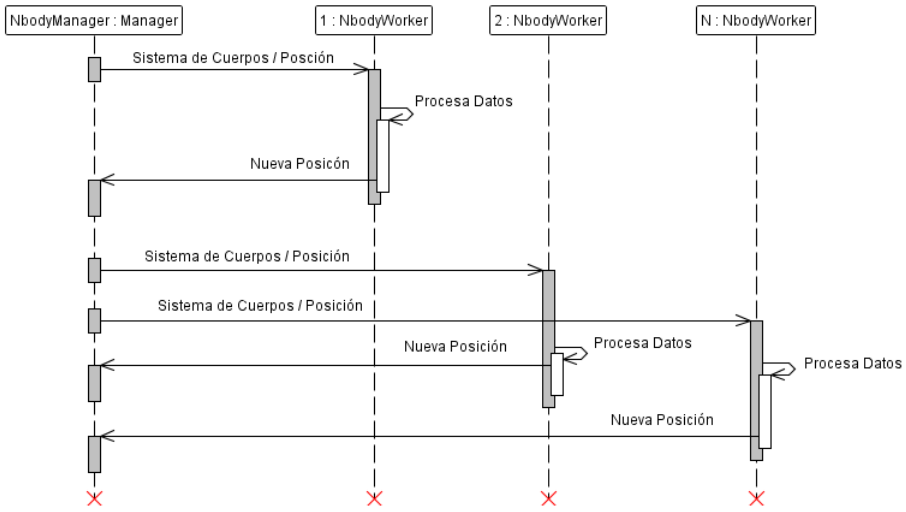


Figura 5.2: Diagrma de Secuencia

5.2. Componentes

Apartir de una vista de alto nivel del nuestro sistema, se defininen los componentes estructurales y sus principales acciones. No obstante, el modelado del sistema aún no se completa. Es necesario definir estructuras y funciones de nivel intermedio que complementen el modelado y mapeo del problema.

Para facilitar la construcción del sistema, a continuación se presentan de manera concisa e incremental, las bibliotecas, rutinas principales y objetos que deben ser implementados para tener una solución integral que resuelva en forma paralela el problema de los N-Cuerpos.

5.2.1. Utilidades

Entre las consideraciones principales que se deben tener antes de comenzar a diseñar objetos muy complejos, se encuentra el definir las operaciones y tipos de datos que son importantes para el funcionamiento de nuestro modelo. Para el problema de los N-Cuerpos se tienen espacios de tres dimensiones y vectores como la posición,la velocidad y la fuerza. Estos tipos de datos no pueden ser operados bajo las reglas de suma o resta convencionales. Por este motivo, se define el objeto *utilidades*.

Este objeto contiene las operaciones básicas que actúan sobre vectores y las cuales se definen a continuación:

- **Suma de vectores**: Dados los vectores:

$$\overline{X} = (x_1, y_1, z_1)$$
$$\overline{Y} = (x_2, y_2, z_2)$$

Definimos la suma como sigue:

$$\overline{X} + \overline{Y} = (x_1 + x_2, y_1 + y_2, z_1 + z_2).$$

De esta forma es posible tomar los vectores como arreglos de en Java de longitud tres y operar con ellos para definir la función suma.

```
1      public double[] suma(double[] a, double[] b){
2          if(a.length == b.length){
3              double[] res = new double[b.length];
4              for(int i=0; i<a.length; i++){
5                  res[i] = a[i] + b[i];
6              }
7              return res;
8          }
9          else{
10             System.out.println("Longitud de los vectores
                   incompatibles");
11             double[] res = new double[a.length];
12             return res;
13         }
14     }
```

- **Resta de vectores**: Dados los vectores:

$\overline{X} = (x_1, y_1, z_1)$
$\overline{Y} = (x_2, y_2, z_2)$

Definimos la resta como sigue:

$$\overline{X} - \overline{Y} = (x_2 - x_1, y_2 - y_1, z_2 - z_1).$$

De esta forma es posible tomar los vectores como arreglos de Java de longitud 3 y operar con ellos para definir la función resta.

```
1      public double[] resta(double[] a, double[] b){
2          if(a.length == b.length){
3              double[] res = new double[b.length];
4              for(int i=0; i<a.length; i++){
5                  res[i] = a[i] - b[i];
6              }
7              return res;
8          }
9          else{
10             System.out.println("Longitud de los vectores
                   incompatibles");
```

```
            double[] res = new double[a.length];
            return res;
        }
    }
```

- **Producto Escalar**: Dado un vector $\overline{X} = (x_1, y_1, z_1)$ y un escalar m, definimos el producto escalar como sigue:

$$m * \overline{X} = (m * x_1, m * y_1, m * z_1)$$

A partir de esto es facil modelar la función *productoE*, que recibe un arreglo de longitud 3 y un número que se denomina escalar de la manera siguiente:

```
    public double[] productoE(double[] vector, double
        escalar){
        double[] res = new double[vector.length];
        for(int i=0; i < vector.length; i++){
            res[i] = vector[i]*escalar;
        }
        return res;
    }
```

- **Norma de un Vector**: Definimos la norma de un vector $\overline{X} = (x_1, y_1, z_1)$ como sigue:

$$|\overline{X}| = \sqrt{x_1^2 + y_1^2 + z_1^2}$$

Siguiento la misma tendencia que describimos anteriormente tomamos al vector como un arreglo de longitud tres y operamos sobre él defininiendo la función longitud como sigue:

```
    public double longitud(double[] vector){
        int l = vector.length;
        double res = 0;
        for(int i=0; i<l; i++){
            res = res + Math.pow(vector[i], 2);
        }
        return Math.sqrt(res);
    }
}
```

Con estas funciones podemos operar los vectores y usarlos para trabajar con las propiedades de los cuerpos dentro del sistema, generando de esta manera la clase Utilidades (ver figura 5.3).

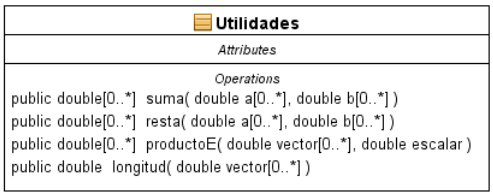


Figura 5.3: Clase Utilidades.Java

5.2.2. Cuerpo

Este objeto modela un cuerpo que se apega a las leyes físicas de Newton. Este cuerpo se ubica dentro del espacio tridimensional, y como tal, posee las siguientes características que lo definen:

- Masa
- Fuerza
- Posición
- Velocidad

Dado que todo Cuerpo del sistema debe poseer estas características, es necesario que cuando se cree un objeto de este tipo sean definidas de manera inicial. Por tal motivo el constructor de este objeto debe inicializar las variables. La manera mas sencilla y controlada de hacerlo es que reciba los valores como parametros, como se muestra a continuación

```
1       public Cuerpo(double[] r,double[] v, double[] f, double m){
2       }
```

Para este caso $r = posicion$, $v = velocidad$, $f = fuerza$ y $m = masa$.

Desde otro punto de vista podemos darnos cuenta que el momentum puede obtenerse mediente el producto escalar de la masa con la velocidad. Por tal motivo es conveniente tener otro constructor que no considere este valor y lo calcule en base a los otro parametros.

```
1       public Cuerpo(double[] r,double[] v, double m){
2           fuerza =  u.productoE(v, m);
3       }
```

Para complementar el modelo y poder trabajar con las propiedades del cuerpo, es necesario agregar funciones que nos permitan accesar y modifiar las variables que se definieron con anterioridad. Este grupo de funciones se describen a continuación:

- Funciones que operan con la masa.

```
1       public double getm(){
2           return masa;
3       }
4       public void setm(double m){
5           masa = m;
6       }
```

- Funciones que operan con la posición.

```
1       public double[] getr(){
2           return pos;
3       }
4       public void setr(double[] r){
5           pos = r;
6       }
```

- Funciones que operan con la fuerza.

```
1       public double[] getf(){
2           return fza;
3       }
4       public void setf(){
5           fza = u.productoE(vdad, masa);
6       }
7       public void setf(double[] f){
8           fza = f;
9       }
```

- Funciones que operan con la velocidad.

```
1       public double[] getv(){
2           return vdad;
3       }
4       public void setv(double[] v){
5           vdad = v;
6       }
```

El conjunto de estas funciones definen la case *Cuerpo* que se implementa como parte de la solución al problema y se presenta en la figura 5.4.

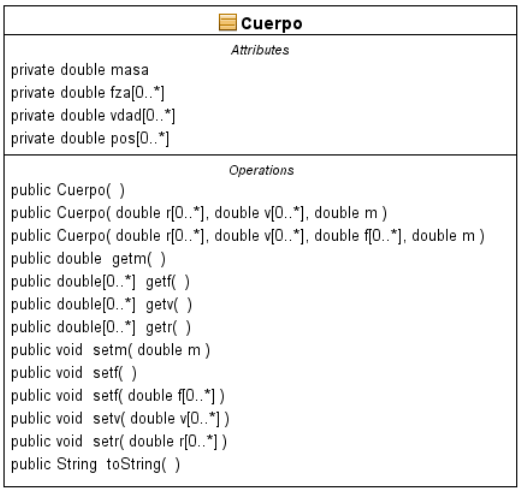


Figura 5.4: Clase Cuerpo.Java

5.2.3. NbodyWorker

Una vez que se ha modelado la colección de objetos que componen nuestro sistema, es necesario definir la forma en la cual se obtiene la nueva posición para cada uno de los cuerpos. Es importante señalar que son estas nuevas estructuras las que se encargan de implementar la parte de ejecución paralela dentro del sistema, realizando de manera simultánea los cálculos para obtener la nueva posición del cuerpo asignado.

La funcionalidad de un objeto NbodyWorker puede ser definida en base a dos actividades principales tomando como punto de partida lo expuesto en el Capítulo 4: en la primera actividad, se calcula la fuerza total que el sistema ejerce sobre un cuerpo determinado, y como segunda fase, con base en el cálculo de la fuerza anterior, se calcula la nueva posición del objeto para el sisguiente instante de tiempo.

- **Cálculo de las fuerzas**: Como se concluye en el Capítulo 4, la fuerza total que el conjunto de cuerpos ejerce sobre un cuerpo determinado i se puede obtener con base en la siguiente formula:

$$f_i = \Sigma_{j=1}^{n} fij$$

  La forma más sencilla de mapear esta fórmula a código es descomponerla en dos funciones menos complejas.

  La función base (que denominamos Fuerza), debe ser una función que reciba dos objetos de tipo *cuerpo* i , j y en base a ellos devuelva la fuerza de atracción que ejerce el cuerpo j sobre el i como un vector de 3 dimensiones. La implementación de esta función se muestra a contintuación.

```
1     private double[] fuerza(Cuerpo pi, Cuerpo pj){
2           double g = 667 * Math.pow(10,-11);
3           double[] rij = u.resta(pi.getr(), pj.getr());
4           double rm = u.longitud(rij);
5           double fm = (g * pi.getm() * pj.getm())/Math.sqrt(
                rm);
6           double[] eij = u.productoE(rij,1/rm);
7           double[] fuerza = u.productoE(eij, fm);
8           return fuerza;
9       }
```

Por último, se necesita una función que recorra todos los cuerpos restantes del sistema y con base en ellos calcule la fuerza que estos cuerpos ejercen sobre uno determinado i, la sume y pueda determinar la fuerza de atracción que ejerce el sistema total sobre este cuerpo determinado. Esta función es denominada *fuerzaT* y como tal debe recibir la colección de cuerpos que componen el sistema a modelar y el identificador del cuerpo con que se va a trabajar.

La representación de nuestro sistema de N cuerpos puede ser vista en código Java como un arreglo de objetos de tipo cuerpo. Dado que los cuerpos dentro del arreglo no intercambian posiciones dentro de este arreglo, basta con que se defina una posición dentro del arreglo para definir un cuerpo en particular.

Como resultado de esta operación la función devuelve el objeto que le fue asignado con la variable fuerza actualizada. Estos detalles se muestran en el código que se presenta a contintuación.

```
1       public Cuerpo fuerzaT(Cuerpo[] sist, int index){
2           int tcps = sist.length;
3           Cuerpo it = new Cuerpo();
4           Cuerpo it2 = sist[index];
5           double[] fij = new double[3];
6
7           for(int i=0; i<tcps; i++){
8               if(i != index){
9                   Cuerpo sec = sist[i];
10                  double[] tmpF = fuerza(it2,sec);
11                  fij = u.suma(fij, tmpF);
12              }
13          }
14          double[] rs = u.suma(it2.getf(),fij);
15          it.setf(rs);
16          it.setm(it2.getm());
17          it.setr(it2.getr());
18          it.setv(it2.getv());
19          return it;
20      }
```

Es así, como a través de estas dos funciones *fuerza* y *fuerzaT*, se modela la ecuación de cálculo de fuerza.

- **cálculo de posciciones**: Una vez definida la fuerza total que el sistema ejerce sobre un cuerpo determinado, es necesario calcular la nueva posición para este cuerpo en un siguiente instante de tiempo. Como se menciona en el Capítulo 4, esto se logra a través de la siguiente ecuación, la cual calcula la nueva posición del cuerpo i para el siguiente instante de tiempo.

$$\Delta r_i = (v_i + 0{,}5\Delta v_i)\Delta t$$

Para implementar esta función es necesario recibir como parámetro un cuerpo y el nuevo instante del tiempo que se está calculando. De esta forma podemos devolver el mismo cuerpo que se recibe como parámetro pero con la nueva posición de éste. La función que se define en Java para ello se llama *mueveC* y se describe a continuación.

```
1       public Cuerpo mueveC(Cuerpo pi, double incremento){
2           double[] ai = u.productoE(pi.getf(), 1/(pi.getm()))
                ;
3           double[] dvi = u.productoE(ai, incremento);
4           double[] dri = u.productoE(u.suma(pi.getv(), u.
                productoE(dvi, .5)), incremento);
5           pi.setv(u.suma(pi.getv(), dvi));
6           pi.setr(u.suma(pi.getr(),dri));
7           return pi;
8       }
```

Estas dos actividades definen las tareas de procesamiento del objeto NbodyWorker y con las cuales es posible calcular la nueva posición de un cuerpo que interactúa con un sistema a través de sus fuerzas gravitacionales dando lugar a la clase del mismo nombre (Figura 5.5).

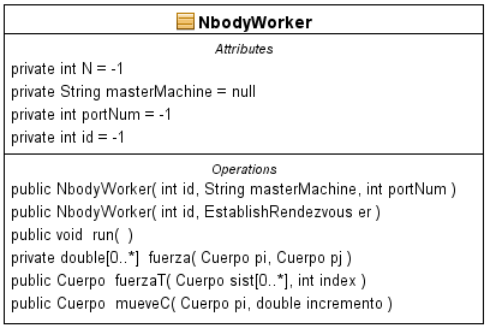


Figura 5.5: Clase NbodyWorker.Java

5.2.4. Mensajes

Una vez que se ha definido la forma en la cual modelamos nuestro sistema de N cuerpos y los procesos que ejecutados en paralelo se encargan de trajar con el sistema para encontrar la posición de un cuerpo determinado, es necesario definir la forma en que se coordinan y distribuyen las cargas de trabajo.

Para este proyecto se toma como mecanismo de sincronización y comunicación entre componentes a los mensajes. Cualquier tipo de comunicación que se lleve a cabo será a través de ellos. Por tal motivo a continuación se define su estructura.

La funcionalidad principal de los mensajes es asignar trabajo a algún objeto NbodyWorker o recibir trabajo de él. Es por eso que se definieron las siguientes variables para realizar el trabajo:

- **workerID**: Identificador que define a un objeto NbodyWorker con el cual deseamos trabajar.
- **containsResult**: Notifica si el presente mensaje mensaje contiene algún resultado, es decir un cuerpo con la posición actualizada en base al sistema.
- **res**: Objeto de tipo cuerpo con valores actualizados.
- **sistema**: Colección de cuerpos que representan el estado actual del sistema y sobre el cual se desea operar.
- **containsWork**: Variable que notifica que lo que se entrega en el mensaje es trabajo para realizar, no un resultado.
- **pos**: En caso de que el mensaje sea para realizar un trabajo, se indica la posición del cuerpo dentro del sistema.
- **incremento**: Instante del tiempo en el cual se encuentra nuestro sistema.
- **date**: Identifica el tiempo en el cual el mensaje fue enviado.

Estos conceptos son suficientes para intercambiar información y obtener el resultado esperado. Dado que por cada ocasión que un mensaje es enviado se crea un nuevo objeto, únicamente se requiere un constructor que permita asignar valores a las variables definidas. Este constructor se muestra a continuación.

```
1  public Message(int workerID, boolean containsResult, Cuerpo[]
       sist)
```

Una vez definidas las variables del objeto mensaje y sus funciones es posible definir la clase *Message* que se muestra en la figura 5.6.

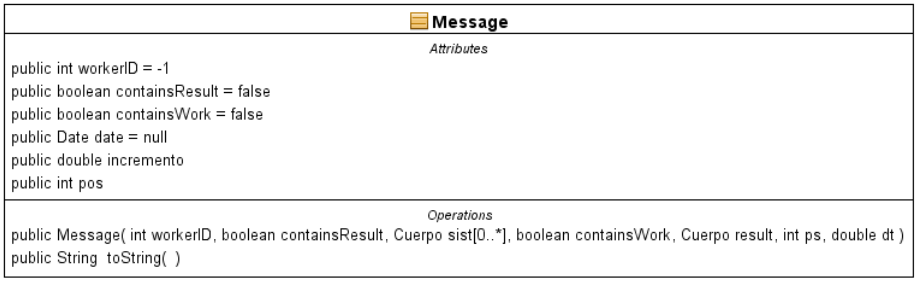


Figura 5.6: Clase Message.Java

5.2.5. NbodyManager

Este componente define el proceso principal de todo el sistema, ya que tiene a su cargo definir las cargas de trabajo para los demas componentes y organizar los resultados de procesar dichas cargas, estas definen de manera general sus tareas principales las cuales se detallan a continuación.

- **Asignación de trabajo**: Una vez que se tiene definido el arreglo de objetos de tipo cuerpo que modela nuestro sistema, y el conjunto de objetos *Nbodyworker* que realizan las tareas de procesamiento, es necesario comenzar con la asignación de trabjo. Primero, se establece la comunicación con el objeto *Nbody-Worker* a través de un objeto *EstablishRendezvous()*. Con esto, se tiene definido un canal de comunicación entre el *NbodyManager* y el *NbodyWorker* a través del cual es posible enviar un mensaje notificando el estado inicial del sistema.

```
1         for (int i = 0; i < numWorkers; i++) {
2             Rendezvous r = er.serverToClient();
3             m = (Message) r.serverGetRequest();
4             if (m.containsResult) {
5                 numResultsReceived++;
6                 System.out.println("age2() =" + age() + " m:"
                      + m);
7
8             }
9             r.serverMakeReply(new Message(-1, false, rs, true
                  , null,i,i));
10            r.close();
11        }
```

- **Recolección de Datos**: Una vez que el trabajo ha sido asignado, el objeto *Nbody-Manager* comienza a recolectar los datos envíados a los *NbodyWorkers*. Para

ello, entra en un ciclo de escucha por el canal de comunicación y hasta que no obtiene las respuestas de todos los *NbodyWorkers* a los que se les asigno trabajo, no sale del mismo, ya que se considera que la actualización del sistema aún no ha sido completa.

```
1     while (numResultsReceived < numWorkers) {
2             Rendezvous r = er.serverToClient();
3             m = (Message) r.serverGetRequest();
4             if (m.containsResult) {
5                 hist[numResultsReceived] = m.res;
6                 numResultsReceived++;
7                 System.out.println("age()=" + age() + " m:" +
                      m.toString());
8             }
9             r.serverMakeReply(new Message(-1, false, null,
                  false, null,0, 0));
10            r.close();
11        }
```

Estas tareas se ejecutan dependiendo el número de iteraciones programadas. Por cada iteración, el estado del sistema se almacena en un objeto vector denominado *hist*. Al final este objeto contiene las trayectorias de los N cuerpos en intervalos de tiempo determinados.

Dado que este proceso se considera el principal dentro de nuestro sistema, es el que comienza a ejecutarse y solo contiene un método: el método *main* que hecha a andar el sistema (Figura 5.7).

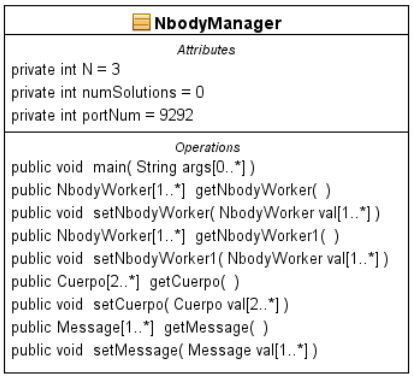


Figura 5.7: Clase NbodyManager.Java

5.3. Interacción

Con las tareas fundamentales de los componentes del sistema especificadas, es necesario comenzar a describir la interacción entre éstos. Es por eso que retomando los conceptos definidos en los apartados 5.1 y 5.2, se describe la interacción de los com-

ponenetes construidos en base a tres vistas principales:

- A nivel de clases (Através de diagramas de clases)
- A nivel de procesos (Através de diagramas de secuencia)
- A nivel de hardware (Através de diagramas de distribución)

Con ello, se pretende generar diferentes puntos de vista sobre le funcionamiento del mismo y que el lector pueda comprender el funcionamiento de la solución implementada.

A continuación la figura 5.8 presenta el diagrama de paquetes del sistema. En él, se presentan las clases involucradas en la ejecución del sistema y las relaciones entre cada una de ellas.

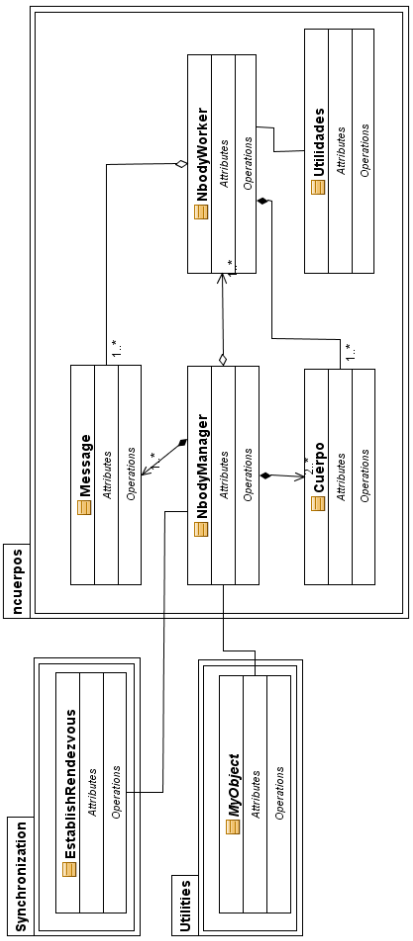


Figura 5.8: Diagrama de paquetes

Una vez que se ha definido desde un punto estático, la forma en que las clases del programa se relacionan es necesario definir ahora la forma en la cual todos estos objetos creados a partir de las clases interactúan, para obtener una solución al problema de los N-cuerpos. Esto puede verse de manera gráfica através de un diagrama de secuencia (ver figura 5.9). Este tipo de diagramas facilita al lector la comprensión del funcionamiento del sistema, debido a que muestra la interacción de los objetos que componen la aplicación a través del tiempo.

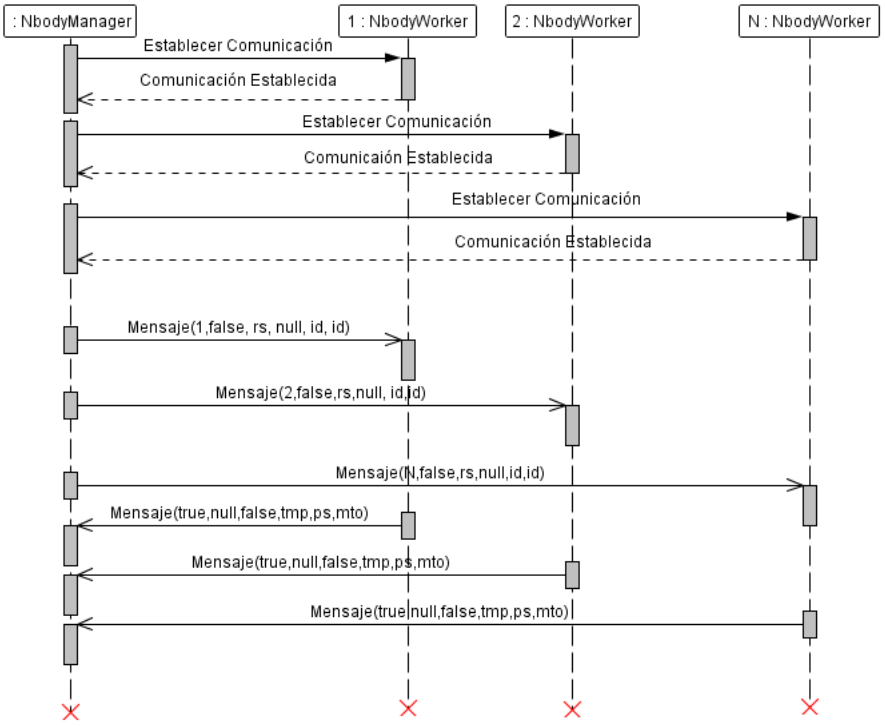


Figura 5.9: Diagrama de Secuencia del sistema

Una vez que se definieron los componentes del sistema y la manera en la cual se relacionan, es necesario dar a conocer la forma en la cual se distribuyen estos los sobre equipos de cómputo. Para ello se incluye el diagrama de Distribución del sistema (ver figura 5.10).

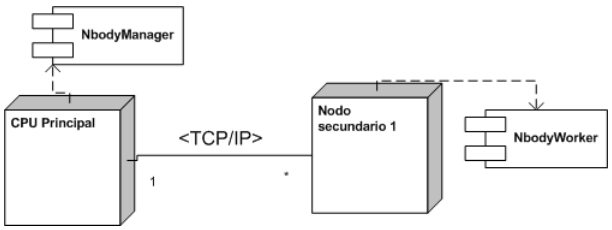


Figura 5.10: Diagrama de Distribucion

Este tipo de diagramas permiten al lector darse cuenta de la forma en la cual el sistema se encuentra implementado en el mundo real, complementando así idea que se tenía del mismo hasta el momento.

5.4. Resumen

Este capítulo plantea de manera gradual la implementación de una solución al problema de los N-cuerpos descrito en el Capítulo 4, en el cual definimos dicho problema como:

> Dadas en un instante las posiciones y las velocidades de tres o más partículas que se mueven bajo la acción de sus atracciones gravitatorias mutuas, siendo conocidas las masas de las partículas, es posible calcular sus posiciones y velocidades para otro instante [4].

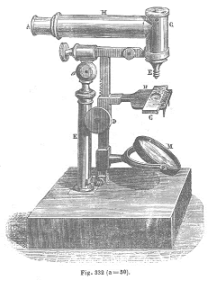
De primera instancia, se analiza un patrón de diseño que se ajuste a la taxonomía del problema planteado, el cual resultó en el patrón Manager Workers.
Después de esto, se conceptualiza la forma en la cual el problema de los N-cuerpos puede ser planteado en términos del patrón de diseño elegido. A partir de esto, se van definiendo las clases necesarias, sus funciones principales y la interacción que tienen, dando como resultado el modelo estático de la solución.

Una vez que se tiene un diseño estático que comprende las funciones principales del sistema, se introduce un modelo dinámico del mismo, através de distintos diagramas como los son, los dde secuencia y de distrubución. Esto con el fin de definir de forma clara, el funcionamiento del sistema paralelo construido.

El objetivo es tener una base sólida y clara sobre la forma en la cual debe operar el sistema y así, poder construir una aplicación. Es importante señalar que el código fuente de dicha solución se encuentra ditribuido en los apéndices de este trabajo en caso de que el lector requiera un mayor detalle.

Capítulo 6

Experimentación y desarrollo



Este capítulo describe la experimentación que se llevó a cabo para validar la hipótesis original. Así como el procedimiento a seguir y los resultados obtenidos, todo esto, con el fin de conocer la validez de las ideas planteadas en este trabjo y así, tener una línea base para generar las conclusiones.

Antes de comenzar la descripción del experimento, es necesario identificar qué características del sistema tienen impacto en su coordinación, y de ellas, cuáles nos es posible controlar para poder evaluar el impacto que la coordinación tiene en el desempeño total del sistema. Para ello, partimos de la idea de evaluar cómo se comporta un sistema de cómputo paralelo con y sin sus componentes de procesamiento, para de esta forma conocer el impacto que la coordinación tiene en el desempeño total del mismo.

Considerando esta idea, es necesario evaluar todas aquellas características de nuestro sistema paralelo (**Programa N-Cuerpos**) e identificar aquéllas que tienen mayor impacto en el comportamiento de la coordinación. Una vez identificadas, se debe elegir aquellas características que pueden ser controladas para determinar el impacto total de la coordinación dentro de nuestro sistema.

Si analizamos nuestro sistema de cómputo desde un punto de vista estructural, tomando como base el patrón **Manager Workers**, podemos darnos cuenta que entre mayor número de Workers, la comunicación que se establece con el Manager para llevar a buen fin su trabajo es mayor. Esto, debido a que el sistema mapea un componente worker para cada objeto cuerpo de nuestro sistema de N-cuerpos. Por tal motivo, para la realización del experimento es posible tomar como una variable el **número de Work-**

ers que el sistema debe generar.

Siguiendo esto, podemos ver mas allá y esta vez tomar como base la naturaleza del problema. De esta forma podemos apreciar que entre más puntos contengan las trayectorias seguidas por cada cuerpo, mayor es la comunicación entre el manager y el worker. Esto, debido a que por cada punto de la trayectoría que calcula el worker, éste debe envíar al manager un objeto cuerpo con su nueva posición. Asimismo, todos los workers del sistema deben ejecutar esta tarea para el cuerpo que tangan asignado, de manera que a mayor número de puntos, mayor es la comuncación con el manager. De esta forma, otra variable que podemos considerar es **el número de iteraciones** a realizar por núestro sistema.

Tomando las variables definidas anteriormente, nos es posible controlar el comportamiento del sistema y variar la coordinación que establece el sistema entre sus componentes. Con ello, se faciliita la posibilidad de calcular el impacto de la coordinación en el desempeño total del sistema.

Para tener un mejor control sobre la realización del experimento, éste se divide en dos fases principales:

- Fase de Evaluación Integral: Se ejecuta el programa completo, y se mide su desempeño en términos del tiempo de ejecución.

- Fase de evaulación de Coordinación: Se omiten las porciones de código que realizan procesamiento y se ejecuta el programa que representa sólo la coordinación, midiendo así su desempeño en el tiempo.

Las valores obtenidos dentro de las fases antes mencionadas, son comparados y evaluados através de métodos estadistícos con el fin de conocer el impacto que tiene la coordinación en el desempeño total del sistema (ver figura 6.1).

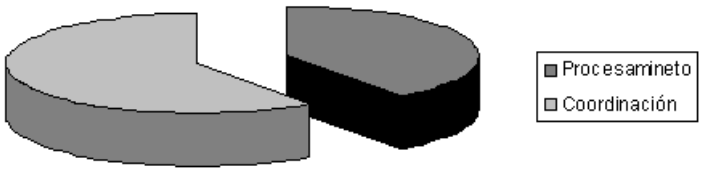


Figura 6.1: Suposición del Impacto de la coordinación

6.1. Configuración de Hardware

A continuación se presenta de manera específica la configuración de Hardware sobre el cual es ejecutado el sistema de cómputo paralelo que aproxíma una solución para el problema de los **N-cuerpos**.

Caracteristíca	Descripción
Procesadores	Dos procesadores Intel Xeon, 2.6GHz
Motherboard	SE7501BR2 Intel dual Xeon
Memoria RAM	1GB de memoria RAM
Disco Duro	SCSI Cheeta, 80GB
Unidades Opticas	CD/Writer interno Samsumg
Periféricos	Teclado y mouse Microsoft PS2
Monitor	SVGA Color Samsumg, 17 pulg
CPU	Gabinete Intel SC5250-E dual

Cuadro 6.1: Configuración de Servidor

Característica	Descripción
Procesadores	Procesador Intel Pentium 4, 2.6GHz
Motherboard	Motherboard Intel Pentium 4 BOSC845GBSRL
Memoria RAM	512 MB de memoria RAM
Disco Duro	Disco duro Seagate de 40 GB
CPU	Gabinete para P4 AOPEN Micro ATX 300W
No de Nodos	16

Cuadro 6.2: Configuración de Nodos

Caracteristíca	Descripción
Modelo	3com Superstack 3 4226T
Puertos 10/100	24
Puertos 10/100/1000	2

Cuadro 6.3: Configuración de Switch de Red

6.2. Configuración de Software

A continuación se presenta de manera específica la configuración de Software sobre el cual es ejecutado el sistema de cómputo paralelo **N-cuerpos**.

Característica	Descripción
Sistema Operativo	Linux-Debian
Compiladores	GNU project C and C++ compiler (gcc) GNU project Fortran 77 compiler (g77) Java 2 Platform, Standard Edition, v 1.4.2 (J2SE)
Ambiente de Programación Paralela	Parallel Virtual Machine (PVM) version 3.4.5; XPVM version 1.2.5 Message Passing Interface (MPI) version 2; MPICH version 0.971

Cuadro 6.4: Configuración de Software

6.3. Fase de Evaluación Integral

En esta fase, se toman las mediciones de los tiempos totales de ejecución obtenidos a partir de las ejecuciones de nuestro sistema paralelo.

Las métrica que se emplea para evaluar el desempaño consiste en relacionar el número de Workers contra el número de Iteraciones por worker, generando así, un esquema de cómo varía el desempeño total del sistema respecto al número de workers e iteraciones. (ver cuadro 6.5).

La figura 6.2 presenta los resultado de manera gráfica.

Iter \ Work	10	20	30	40	50
1	718	730	728	728	735
2	738	769	789	818	852
3	758	809	853	899	950
4	781	851	951	979	1054
5	799	890	968	1059	1149
6	819	930	1029	1138	1249
7	839	970	1090	1218	1349
8	860	1010	1149	1299	1449
9	879	1050	1210	1379	1549
10	899	1078	1269	1460	1650

Cuadro 6.5: Propiedades del Tiempos totales de ejecución del programa paralelo (en segundos)

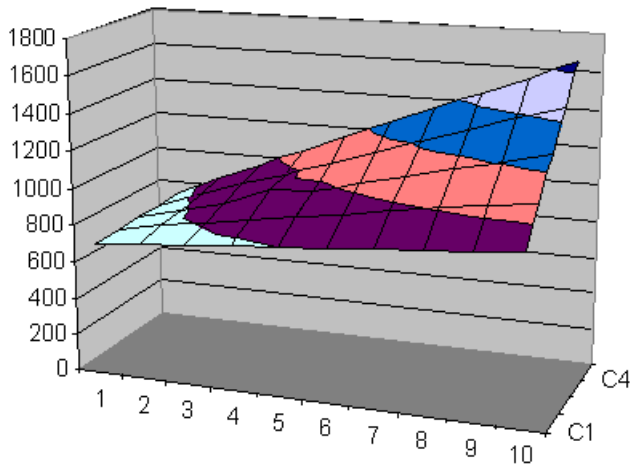


Figura 6.2: Propiedades del Tiempos totales de ejecución del programa paralelo (en segundos)

6.4. Fase de evaulación de Coordinación

En esta fase se eliminan los tiempos de procesamienton del sistema con el fin de lograr una aproximación clara del impacto de la coordinación. Para ello, cada worker debe recibir su asignación de trabajo y devolver un valor fijo establecido previamente. De esta forma, el tiempo que tarda en realizar esta tarea puede ser despreciable. Una vez eliminados los tiempos de procesamiento, es necesario tomar las mediciones de los tiempos totales de la ejecución del sistema considerando los mismos escenarios realizados anteriormente. El cuadro 6.6 muestra los resultados obtenidos en esta fase.

Para un mejor entendimiento de los resultados también se presentan de manera gráfica en la figura 6.3.

Iter \ Work	10	20	30	40	50
1	658	667	678	693	709
2	675	707	739	772	805
3	705	749	798	853	907
4	717	789	860	934	1006
5	739	829	918	1013	1106
6	756	870	984	1093	1208
7	775	914	1038	1174	1310
8	810	990	1099	1254	1408
9	825	1011	1159	1334	1509
10	836	1039	1219	1415	1601

Cuadro 6.6: Propiedades del Tiempos totales de ejecución del programa paralelo (en segundos)

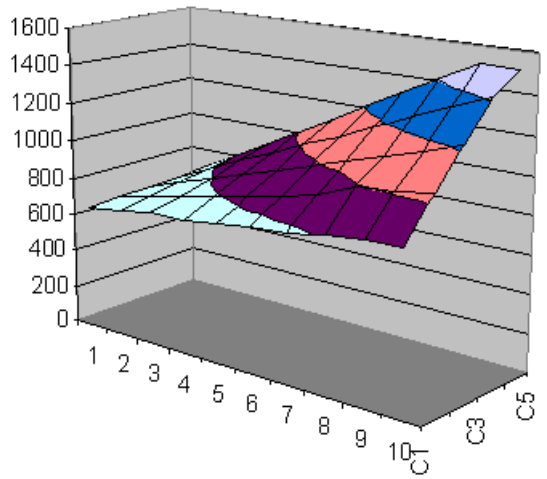


Figura 6.3: Propiedades del Tiempos totales de ejecución del programa paralelo (en segundos

6.5. Resumen

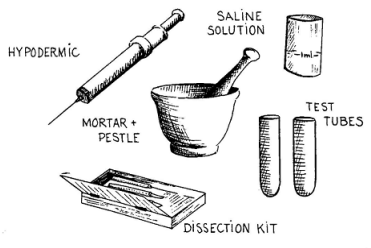
En este capítulo se presenta y describe el proceso a través del cual se evalúa el impacto de la coordinación dentro de un sistema de cómputo paralelo. Para ello, iniciamente se describen las características relevantes que rigen el comportamiento de la coordinación.

Estas características son de importancia ya que a lo largo del experimento actúan como banderas que indican el aumento o disminución de la coordinación dentro de nuestro sistema.

Los valores obtenidos como resultado de este capítulo son análizados y explicados en las siguientes secciones para concluir una aproximación del impacto de la coordinación dentro de un sistema de cómputo paralelo.

Capítulo 7

Evaluación



Se evalúa la calidad del trabajo realizado y los resultados obtenidos para tener una línea base con la cual generar las conclusiones.

7.1. Consideraciones

Tomando como base el modelo de evaluación de desempeño presentado en el trabajo relacionado, se define un proceso de evaluación de desempeño compuesto de un modelo determinístico. Es importante señalar que no se genera un modelo estocástico del sistema debido a que el único componente que se modela de esta forma son los tiempos de comunicación, los cuales presentan ya de manera natural este tipo de comportamiento.

- Dentro del modelo determinístico, definimos el tiempo que el sistema invierte en procesar cada una de sus tareas. La meta del experimento consistió de hacer este tiempo tan pequeño que llege a ser despreciable, esto, con el fin de conocer el impacto real de la coordinacióin dentro del sistema.

De esta forma, tenemos que el tiempo total por cada iteración se puede presentar como:

$$T_{total} = T_{procesamiento} + T_{coordinacion}$$

Posteriormente, siguiendo nuestro modelo determinístico, fue posible eleminar o hacer despreciable el tiempo que el sistema invierte en procesar. Logrando así tener el impacto de la coordinación dentro del sistema.

$$T_{total} - T_{procesamiento} = T_{coordinacion}$$

7.2. Evaluación del impacto de la coordinación

Gracias a los valores obtenidos al restar el tiempo de coordinación al tiempo total, nos es posible conocer cual fue el tiempo invertido en procesamiento a lo largo de las ejecuciones realizadas (ver cuadro 7.1).

Iter \ Work	10	20	30	40	50
1	60	63	50	35	26
2	63	62	50	46	47
3	53	60	55	46	43
4	64	62	91	45	48
5	60	61	50	46	43
6	63	60	45	45	41
7	64	56	52	44	39
8	50	20	50	45	41
9	54	39	51	45	40
10	63	39	50	45	49

Cuadro 7.1: Impacto del desempeño dentro del sistema

Como podemos observar, el comportamiento apunta hacia una constante del tiempo de procesamiento. No obstante, es de señalar que tambien existen saltos importantes entre algunas mediciones tomadas. Estos saltos pueden ser causados debido a errores de medición ajenos a la ejecución del problema.

Es entonces, que despreciando los valores antes mencionados, podemos ver que el procesamiento tiende a mantenerse constante, variando muy poco a media que se incrementan el número de workers dentro del sistema. En contraste, la figura 7.1 refleja que la proporción en la cual varia el tiempo invertido en coordinación no es el mismo que para el procesamiento. Ya que los tiempos de esta tienden a tener un impacto mayor en el desempeño total del sistema a medida que el número de workers e iteraciones dentro del sistema aumenta.

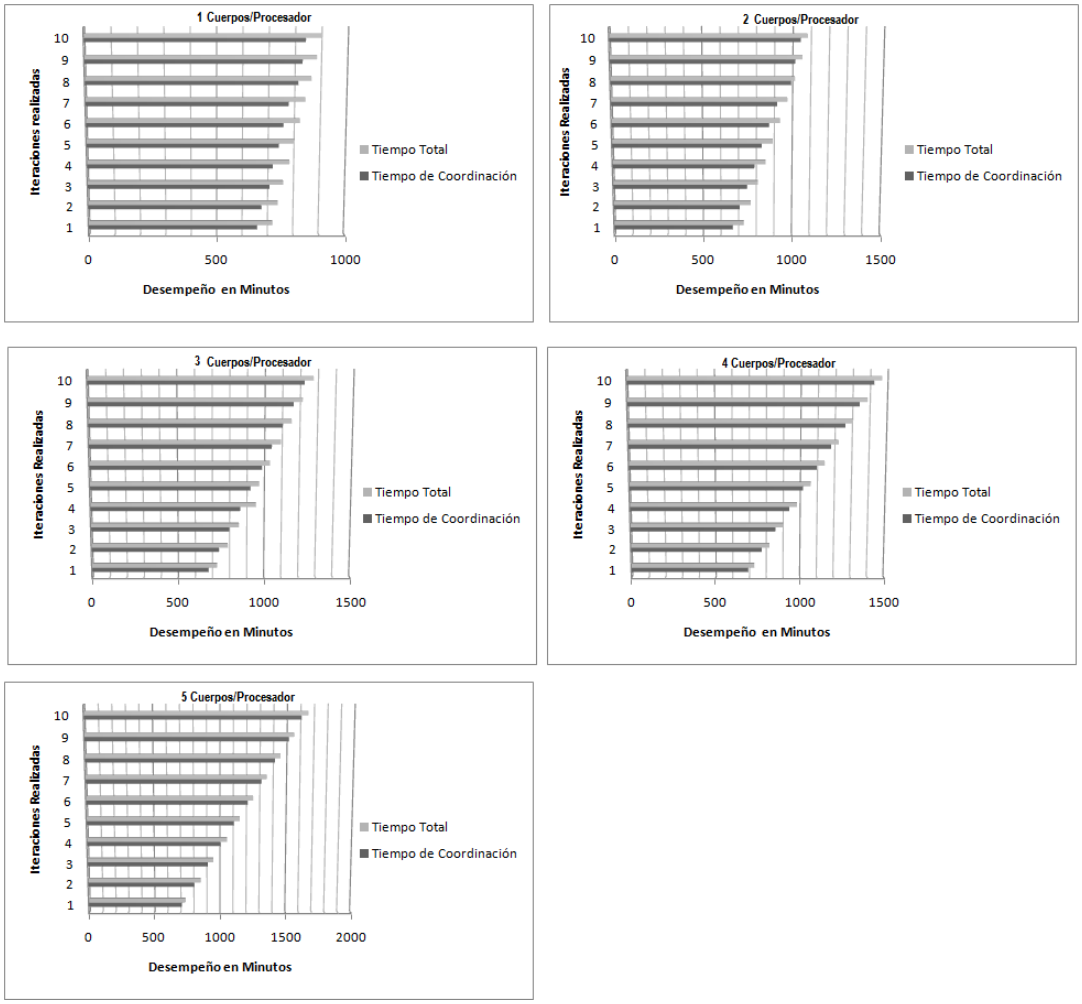


Figura 7.1: Comportamiento del desempeño del sistema

En la figura 7.1, además, podemos observar que el procesamiento se mantiene mas o menos constante a lo largo de todas las ejecuciones realizadas, variando alrededor de $11{,}23seg$ por iteración, mientras que la coordinación varia alrededor de un $177seg$ por iteración. Lo cual es un indicador claro de que la coordinación aumenta el tiempo total de ejecución del sistema en mayor proporción que el procesamiento.

Tomando el tiempo inicial obtenido por cada iteración como el $100\,\%$ del tiempo para esa iteración, es posible definir el tiempo obtenido en la segunda iteración como un porcentaje del tiempo total (ver cuadro 7.2).

Iter \ Work	10	20	30	40	50
1	92	91	93	95	96
2	91	92	94	94	94
3	93	93	94	95	95
4	92	93	90	95	95
5	92	93	95	96	96
6	92	94	96	96	97
7	92	94	95	96	97
8	94	98	96	97	97
9	94	96	96	97	97
10	93	96	96	97	97

Cuadro 7.2: Impacto de la coordinación en el desempeño total por iteración

Con estos datos podemos obtener el porcentaje promedio del tiempo total invertido en coordinación y procesamiento. (ver figura 7.2).

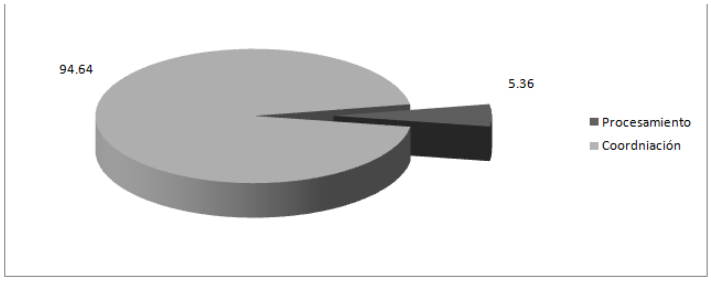


Figura 7.2: Promedio de la coordinación dentro del desempeño total del sistema

7.3. Resumen

Partiendo de un modelo híbrido de evaluación de desempeño, es posible definir de manera determinística el procesamiento y de manera estocástica la coordinación del sistema, logrando así poder controla el nivel de procesamiento de la aplicación para conocer el impacto de la coordinación dentro del sistema.

La forma de lograr este objetivo es reducir el tiempo de procesamiento, de manera que su impacto en el desempeño total del sistema sea despreciable. Permitiendo así, concocer el impacto de la coordinacióin.

$$T_{total} - T_{procesamiento} = T_{comunicacion}$$

Con base en los resultados en cada iteración antes y después de eliminar el procesamiento, fue posible conocer el impacto por iteración que tenia la coordinación dentro del sistema (ver cuadro 7.2).

adicionalmente a esto, realizando un promedio de los datos encontrados, es posible visualizar a través de la figura 7.2 el porcentaje tan relevante que tiene la coordinación dentro de nuestro sistema.

Capítulo 8

Conclusiones

Este capítulo presenta un resumen del trabajao de investigación, exponiendo de manera crítica, como las hipótesis fueron aplicadas a lo largo de un proceso de investigación, hasta llegar a los resultados presentados. Al final, se presenta un apartado de trabajo futuro, en el cual se explica el siguiente nivel al que se puede llevar este experimento para un proyecto posterior.

8.1. Resumen del trabajo de investigación

El trabajo de investigación realizado a lo largo de esta tesis, consistió en evaluar los aspectos principales que definen un sistema de cómputo paralelo:

- Coordinación
- Procesamiento

Con el fin de poder estimar el impacto que la coordinación tiene sobre el desempeño total del sistema.

Es importante señalar que para el presente trabajo, la estimación del desempeño fue resultado del tiempo que el conjunto de componentes de software requirió para procesar alguna actividad.

Es decir, el desempeño total del sistema se ve como el tiempo total de procesamiento, el cual a su vez se compone del tiempo que el sistema invierte en procesar y del tiempo que el sistema invierte en coordinación.

$$T_{procesamiento} + T_{coordinacion} = T_{total}$$

Siguiendo con esta lógica, para conocer el impacto de la coordinación dentro del sistema, únicamente se debe mantener constante el procesamiento para saber el tiempo que la coordinación aporta.

$$Tiempo_{coordinacion} = T_{total} - T_{procesamiento}$$

La motivación para este trabajo surge al saber que la mayoría de los métodos de evaluación de desempeño que existen en la actualidad solo miden el poder de procesamiento del sistema, dejando de lado la parte de coordinación.

Para comenzar a realizar las pruebas de evaluación de desempeño, se contruyó un sistema de cómputo paralelo que realiza una simulación del problema físico de los N-cuerpos (problema que consiste en encontrar las trayectorias de N cuerpos que interactúan en un espacio aislados a través de sus fuerzas gravitacionales) (ver Capítulo 4). Este sistema aportó el caso de estudio sobre el cua se realizaron nuestros experimentos.

Una vez obtenida la materia prima para este trabajo (el sistema de cómputo paralelo), fue necesario identificar que componentes del sistema aumentaban o disminuían la complejidad del sistema en términos de coordinación. Después de un análisis del sistema, se determinó que las variables que influían en la coordinación del sistema y que además podrían ser controladas y medibles, eran el número de workers que se asignan por procesador y la cantidad de posiciones a calcular para cada uno de los cuerpos del sistema.

De esta forma, se realizaron un conjunto de pruebas variando estos parámetros, pero manteniendo la integridad del sistema y sus procesos.

Como una segunda fase del experimento, se alteró el sistema para que no realizara ningún trabajo de procesamiento, logrando con esto, que el tiempo total que el sistema tarda en correr cada una de las pruebas represente el tiempo invertido en coordinación.

8.2. Reafirmación de la hipótesis

La hipótesis del presente trabjo se escribio dentro del Capítulo 1 como sigue:

> Dado un sistema de cómputo paralelo, cuán significativo es el impacto de la coordinación dentro del desempeño total de este.

8.2.1. Discusión

La forma en la cual esta pregunta se responde en el presente trabajo parte de analizar un problema específico (ver Captítulo 4), e implementar un sistema de cómputo paralelo

que lo modele (ver Capítulo 5)y sobre el cual, se realizan una serie de experimentos que ayudan a obtener un conjunto de tiempos totales de ejecución y tiempos debidos a la coordinación (ver capítulo 6). Estos tiempos al ser analizados, ofrecen una aproximación para este caso en especial, de cuánto tiempo el sistema invierte en procesamiento.

El resultado que se obtuvo fue que para este problema utilizando Manager Workers y el ambiente de software desrrollado, el sistema invierte alrededor de un $90\,\%$ del tiempo en coordinar y únicamente un $10\,\%$ en procesar (ver Cápitulo 7). Lo cual nos permite realizar las siguientes afirmaciones:

- El problema modelado es de granularidad fina. Los componentes del sistema no realizan una gran cantidad de procesamiento.
- Dada la granularidad del problema, el diseño paralelo del sistema puede ser refinado agrupando más los trabajos de procesamiento ó reconstruido definiendo un nuevo patrón de diseño paralelo que se ajuste más a las necesidades del sistema.
- Quizá uno de los puntos más sobresalientes de esto, es que el sistema construido no tiene desempeño óptimo para procesar sus cargas de trabajo. Esto debido a que pierde demasiado tiempo coordinando el trabajo que se le asigna. Sin embargo, no hubiera sido posible determinar esto si solo se hubiera tomando en cuenta el procesamiento para definir una métrica que nos ayudara a determinar el despeño del sistema. Ya que en términos de procesamiento, el sistema cumple sus tareas de forma aceptable.

Por otro lado, es importante señalar que al término de este trabajo, se contribuyó con la implementación de un sistema de cómputo paralelo que modela el problema de los N-cuerpos presentado por Newton, basado en el patrón arquitectonico "Manager Workers". A demás, se generó la documentoación detallada en leguaje UML detallando todos los componentes del sistema de manera estática y dinámica (ver capítulo 5).

Por otro lado, otra aportación que se obtuvo de este trabajo fue la implementación de un modelo de evaluación de desempeño híbrido, compuesto por dos submodelos: uno determinístico y otro estocástico, los cuales en conjunto proporcionan un mejor control de los tiempos de procesamiento y los de coordinación, facilitando su cuantificación y por ende, facilitando la aproximación de como la coordinación impacta el desempeño total del sistema.

8.3. Trabajo futuro

A partir del presente trabajo, surgen una serie de ideas que pueden ser detalladas y presentadas como complemento de esta tesis en trabajos futuros. Algunas de estas de estas ideas se presentan a continuación:

- Implementar un sistema de cómputo paralelo en distintos patrones de arquitectura y realizar pruebas de desempeño sobre estos, para validar el comportamiento del sistema y con ello determinar si la forma en que se comporta el desempeño del sistema en las distintas arquitecturas se aproxima al presentado dentro de esta tesis.

- Realizar pruebas de evaluación de desempeño sobre grupos de sistemas ya implementados para encontrar métricas generales que se sirvan para determinar el desempeño del sistema y el impacto de la coordinación en el mismo.

Las ideas anteriormente presentadas sirven de complemento al trabajo expuesto en esta tesis y ayudan a entender el impacto de la coordinación dentro los sistemas de cómputo paralelo, así como también las métricas para definirlo.

Bibliografía

[1] José Galaviz Casas, *Arquitectura de Computadoras* Vínculos Matemáicos 1er edición.

[2] Jorge Ortega Arjona, José Galavíz Casas *Programación concurrente Notas de Clase* Maestría en Ciencias de la Computación UNAM.

[3] G.S. Almasi and A. Gottlieb. *Highly Parallel Computing* Benjamin-Cummings publishers, Redwood city, CA, 1989.

[4] Wikipedia. *Problema de los dos cuerpos*

[5] Vikram S. Adve. *Analyzing the Behavior and Performance of Parallel Programs* University of Wisconsin-Madison, Diciembre 1993.

[6] Lei Hu and Ian Gorton, October 1997. *Performance Evaluation for Parallel Systems: A Suvey* Departmen of Computer Systems School of Computer Science and Engineering University of NSW, Sydney 2052, Australia.

[7] Jorge L. Ortega Arjona and Graham Roberts. *Architectural Patterns for Parallel Programming* Department of Computer Science, University College London, 1998.

[8] Jorge L. Ortega Arjona. *The Communicating Sequential Elements Pattern a Domain Paralellism Architectural Pattern for Parallel Programming* Department of Computer Science, University College London, 2000.

[9] Jorge L. Ortega Arjona. *The Shared Resouce Pattern an Activity Paralellism Architectural Pattern for Parallel Programming* Departamento de Matemáticas Facultad de Ciencias, UNAM, 2003.

[10] Jorge L. Ortega Arjona. *The Manager Workers Pattern an Activity Paralellism Architectural Pattern for Parallel Programming* Departamento de Matemáticas Facultad de Ciencias, UNAM, 2004.

[11] Jorge L. Ortega Arjona. *The Parallel Pipes and Filters Pattern a Functional Paralellism Architectural Pattern for Parallel Programming* Departamento de Matemáticas Facultad de Ciencias, UNAM, 2005.

[12] Jorge L. Ortega Arjona. *The Parallel Layer Pattern a Functional Paralellism Architectural Pattern for Parallel Programming* Departamento de Matemáticas Facultad de Ciencias, UNAM, 2007.

[13] Brinch Hansen. *Studies In Computational Science: Parallel Programming Paradigms* Syracuse University,Prentice Hall Engineering/Science/Mathematics,1995

Apéndice A

Clase Utilidades

```
1  package ncuerpos;
2
3  import java.io.*;
4  import java.util.Vector;
5  import java.io.File;
6  import org.w3c.dom.Document;
7  import org.w3c.dom.*;
8
9  import javax.xml.parsers.DocumentBuilderFactory;
10 import javax.xml.parsers.DocumentBuilder;
11 import org.xml.sax.SAXException;
12 import org.xml.sax.SAXParseException;
13 /**
14  * Clase: Utilidades.java
15  * Descripción: Clase encargada de definir mÃ©todos auxiliares
       para la suma,
16  *              resta, longitud y producto escalar de un vector
       .
17  * @author Fco. Javier Mena Barraza
18  * @version 1.1
19  */
20 public class Utilidades {
21
22     /**
23      * Funcion encargada de sumar dos vectores que seran
24      * tomados como coordenadas (x,y,z) de un cuerpo.
25      * @param double[] a primer sumando
26      * @param double[] b segundo sumando
27      * @return double[] res segundo sumando
28      */
29     public double[] suma(double[] a, double[] b){
30         /*
31          * Validamos que la longitud de los vectores, sea la
               misma
32          * para poder realizar la suma.
33          */
```

```
34          if(a.length == b.length){
35              // Vector en el cual se almacena el resultado.
36              double[] res = new double[b.length];
37              // Realizamos la suma entrada a entrada.
38              for(int i=0; i<a.length; i++){
39                  res[i] = a[i] + b[i];
40              }
41              return res;
42          }
43          else{
44              System.out.println("Longitud de los vectores
                    incompatibles");
45              double[] res = new double[a.length];
46              return res;
47          }
48      }
49
50       /**
51        * Funcion encargada de restar dos vectores que seran
52        * tomados como coordenadas (x,y,z) de un cuerpo.
53        * @param a double[] primer sumando
54        * @param b double[] segundo sumando
55        * @return res double[] segundo sumando
56        */
57      public double[] resta(double[] a, double[] b){
58          /*
59           * Validamos que la longitud de los vectores, sea la
                misma
60           * para poder realizar la resta.
61           */
62          if(a.length == b.length){
63              // Vector en el cual se almacena el resultado.
64              double[] res = new double[b.length];
65              // Realizamos la resta entrada a entrada.
66              for(int i=0; i<a.length; i++){
67                  res[i] = a[i] - b[i];
68              }
69              return res;
70          }
71          else{
72              System.out.println("Longitud de los vectores
                    incompatibles");
73              double[] res = new double[a.length];
74              return res;
75          }
76      }
77
78      /**
79       * Metodo encargado de realizar el producto escalar para un
             vector.
80       * @param vector
81       * @param escalar
82       * @return
83       */
```

```
84      public double[] productoE(double[] vector, double escalar){
85          // Creamos un vector que almacenara el resultado.
86          double[] res = new double[vector.length];
87
88          // Realizamos el producto.
89          for(int i=0; i < vector.length; i++){
90              res[i] = vector[i]*escalar;
91          }
92
93          return res;
94      }
95
96      /**
97       * Metodo encargado de obtener la norma de un vector.
98       * @param vector double[] vector a evaluar.
99       * @return res double longitud del vector.
100      */
101     public double longitud(double[] vector){
102         // Obtenemos el tamaño del vector
103         int l = vector.length;
104
105         // Creamos un entero para devolver el resultado
106         double res = 0;
107
108         // Obtenemos la longitud del vector
109         for(int i=0; i<l; i++){
110             res = res + Math.pow(vector[i], 2);
111         }
112         return Math.sqrt(res);
113     }
114
115     /**
116      * MÃ©todo encargado de construir un archivo xml. a partir
117      * de todas las iteraciones del sistema que fueron
            generadas,
118      * se crea una serie de putnos que definen la trayectoria
            para un
119      * de nuestro sistema.
120      * @param Vector sisstema: Sistema a interpretar.
121      * return void
122      */
123     public void genXML(Vector sistema){
124         // Creamos nuestro archivo
125         File file = new File("C:\\ncuerpos.xml");
126
127         // Variable que contiene el contenido del archivo
128         String result ="<?xml version='1.0' encoding='UTF-8'?>\
                n\t<sistema>\n";
129
130         // Comenzamos a validar nuestro sistema.
131         Cuerpo[] inicial = (Cuerpo[])sistema.get(0);
132         String[] trayect = new String[inicial.length];
133
134
```

```
135             // Generamos la cabecera del archivo XML para cada
                    cuerpo.
136             for(int i=0; i < trayect.length; i++){
137                 trayect[i] = "\t\t<cuerpo id=\'"+ i +"\'>\n";
138             }
139
140
141             // Agregamos las posiciones al archivo
142             for(int i=0; i < sistema.size(); i++){
143                 inicial = (Cuerpo[])sistema.get(i);
144                 for(int j=0; j < inicial.length; j++){
145                     double[] tm = inicial[j].getr();
146                     trayect[j] = trayect[j] + "\t\t\t<punto " +
147                             "x=\'"+ tm[0]/(10000*1000000) +
148                             "\' y=\'"+ tm[1]/(100000*1000000)+
149                             "\' z=\'" + tm[2]/(100000*000000)+"\' >
                                    " + "</punto>\n";
150                 }
151             }
152
153             // Cerramos las cabeceras para cada cuerpo.
154             for(int i=0; i < trayect.length; i++){
155                 trayect[i] = trayect[i] + "\t\t</cuerpo>\n";
156             }
157
158             // Agregamos todos los cuperos al sistema.
159             for(int i=0; i < trayect.length; i++){
160                 result = result + trayect[i];
161             }
162
163             // Generamos el final del archivo
164             result  = result +  "\t</sistema>";
165
166
167             // Creamos el archivo
168             try{
169                 FileWriter out = new FileWriter(file, true);
170                 out.write(result);
171                 out.close();
172             }catch(Exception e){
173                 System.out.println(e);
174             }
175         }
176
177         public Cuerpo[] parseXML(String tmp ){
178             Cuerpo [] resultado = new Cuerpo[1];
179             try {
180                 DocumentBuilderFactory docBuilderFactory =
                        DocumentBuilderFactory.newInstance();
181                 DocumentBuilder docBuilder = docBuilderFactory.
                        newDocumentBuilder();
182                 Document doc = docBuilder.parse (new File(tmp));
183
184                 // normalize text representation
```

```
185                 doc.getDocumentElement ().normalize ();
186                 System.out.println ("Root element of the doc is " +
187                      doc.getDocumentElement().getNodeName());
188
189
190                 NodeList listOfCuerpos = doc.getElementsByTagName("
                        cuerpo");
191                 int totalCuerpos = listOfCuerpos.getLength();
192                 Cuerpo[] res = new Cuerpo[totalCuerpos];
193
194                 for(int s=0; s<listOfCuerpos.getLength() ; s++){
195
196
197                     Node firstCuerpoNode = listOfCuerpos.item(s);
198                     if(firstCuerpoNode.getNodeType() == Node.
                            ELEMENT_NODE){
199
200                         Element firstCuerpoElement = (Element)
                                firstCuerpoNode;
201
202                         //-------Obtenemos el ID del cuerpo a
                                parsear.
203                         NodeList firstNameList = firstCuerpoElement
                                .getElementsByTagName("id");
204                         Element firstNameElement = (Element)
                                firstNameList.item(0);
205                         NodeList textFNList = firstNameElement.
                                getChildNodes();
206
207                         //------
208                         NodeList cuerposList1 = firstCuerpoElement.
                                getElementsByTagName("masa");
209                         Element elementmasa = (Element)cuerposList1
                                .item(0);
210                         String m = elementmasa.getAttribute("m");
211                         double masa = Double.parseDouble(m);
212                          //------
213                         NodeList cuerposList2 = firstCuerpoElement.
                                getElementsByTagName("fuerza");
214                         Element elementFuerza = (Element)
                                cuerposList2.item(0);
215                         String fx = elementFuerza.getAttribute("x")
                                ;
216                         String fy = elementFuerza.getAttribute("y")
                                ;
217                         String fz = elementFuerza.getAttribute("z")
                                ;
218                         double f1x = Double.parseDouble(fx);
219                         double f1y = Double.parseDouble(fy);
220                         double f1z = Double.parseDouble(fz);
221                         double[] fza = {f1x, f1y,f1z};
222                          //------
223                         NodeList cuerposList3 = firstCuerpoElement.
                                getElementsByTagName("velocidad");
```

```
224                         Element elementVelocidad = (Element)
                                cuerposList3.item(0);
225                         String vx = elementVelocidad.getAttribute("
                                x");
226                         String vy = elementVelocidad.getAttribute("
                                y");
227                         String vz = elementVelocidad.getAttribute("
                                z");
228                         double v1x = Double.parseDouble(vx);
229                         double v1y = Double.parseDouble(vy);
230                         double v1z = Double.parseDouble(vz);
231                         double[] vel = {v1x,v1y,v1z};
232                          //------
233                         NodeList cuerposList4 = firstCuerpoElement.
                                getElementsByTagName("posicion");
234                         Element elementPos = (Element)cuerposList4.
                                item(0);
235                         String px = elementPos.getAttribute("x");
236                         String py = elementPos.getAttribute("y");
237                         String pz = elementPos.getAttribute("z");
238                         double p1x = Double.parseDouble(px);
239                         double p1y = Double.parseDouble(py);
240                         double p1z = Double.parseDouble(pz);
241                         double[] pos = {p1x,p1y,p1z};
242                         //------
243                         // Generamos el Cuerpo
244                         Cuerpo elemento = new Cuerpo(pos, vel, masa
                                );
245                         res[s] = elemento;
246                     }//end of if clause
247                 }//end of for loop with s var
248                 resultado = res;
249             }catch (SAXParseException err) {
250             System.out.println ("** Parsing error" + ", line "
251                  + err.getLineNumber () + ", uri " + err.
                         getSystemId ());
252             System.out.println(" " + err.getMessage ());
253             }catch (SAXException e) {
254             Exception x = e.getException ();
255             ((x == null) ? e : x).printStackTrace ();
256             }catch (Throwable t) {
257             t.printStackTrace ();
258             }
259             return resultado;
260         }
261     }
```

Apéndice B

Clase Cuerpo

```
1  package ncuerpos;
2
3  /**
4   * Clase: Cuerpo.java
5   * Descripción: Clase encargada de simular un cuerpo en un
        espacio
6   *              tridimencional con una posición, una fuerza,
        una masa
7   *              y velocidad.
8   * @author Francisco Javier Mena Barraza
9   * @version 1.0
10  */
11 public class Cuerpo {
12     private  double masa;
13     private double[] fza;
14     private double[] vdad;
15     private double[] pos;
16     private Utilidades u;
17
18     /**
19      * Constructor inicializa un vector en 0.
20      */
21     public Cuerpo( ){
22         masa = 0;
23         fza = new double[3];
24         vdad = new double[3];
25         pos = new double[3];
26         u = new Utilidades();
27     }
28
29
30     /**
31      * Constructor que inicializa los valores iniciales del
           cuerpo.
32      * @param r posición inicial del cuerpo.
33      * @param v volocidad del cuerpo en el momento actual
```

```
34          * @param m masa del cuerpo.
35          */
36         public Cuerpo(double[] r,double[] v, double m){
37             u = new Utilidades();
38             masa = m;
39             fza =  u.productoE(v, m);
40             vdad = v;
41             pos = r;
42         }
43 
44         /**
45          * Constructor que inicializa los valores iniciales del
                cuerpo.
46          * @param r posición inicial del cuerpo.
47          * @param v volocidad del cuerpo en el momento actual
48          * @param f fuerza del cuerpo en el momento actual.
49          * @param m masa del cuerpo.
50          */
51         public Cuerpo(double[] r,double[] v, double[] f, double m){
52             u = new Utilidades();
53             masa = m;
54             fza =  f;
55             vdad = v;
56             pos = r;
57         }
58 
59 
60         /**
61          * Regresa la masa actual del cuerpo.
62          * @return
63          */
64         public double getm(){
65             return masa;
66         }
67         /**
68          * MÃ©todo que regresa la fuerza del cuerpo.
69          * @return fza
70          */
71         public double[] getf(){
72             return fza;
73         }
74         /**
75          * MÃ©otodo que regresa la velocidad del cuerpo.
76          * @return vdad
77          */
78         public double[] getv(){
79             return vdad;
80         }
81         /**
82          * MÃ©todo que regresa la posición del cuerpo.
83          * @return pos
84          */
85         public double[] getr(){
86             return pos;
```

```
87        }
88        /**
89         * Ingresa una nueva mas para el cuerpo.
90         * @param m
91         */
92        public void setm(double m){
93            masa = m;
94        }
95        /**
96         * Ingresa la nueva ferza del cuerpo.
97         * @param f
98         */
99        public void setf(){
100           fza = u.productoE(vdad, masa);
101       }
102
103
104       public void setf(double[] f){
105           fza = f;
106       }
107       /**
108        * Ingresa la velocidad del cuerpo.
109        * @param v
110        */
111       public void setv(double[] v){
112           vdad = v;
113       }
114       /**
115        * Ingresa una nueva posición para el cuerpo.
116        * @param r
117        */
118       public void setr(double[] r){
119           pos = r;
120       }
121
122       public String toString() {
123           String res =  "\n <Masa: ("+ masa +") \n Fuerza: ("
124                   + fza[0] +","+ fza[1] +","+ fza[2] + ") \n
                        Velocidad: ("
125                   + vdad[0]+","+ vdad[1]+","+ vdad[2]+") \n
                        Posición: ("
126                   + pos[0] +","+ pos[1] +","+ pos[2] +")> \n";
127           return res;
128       }
129   }
```

Apéndice C

Clase NbodyWorker

```
ï»¿package ncuerpos;
/**
 * Clase main.java
 * Descripcion: Clase encargada de llevar a cabo la simulacion.
 * @author Francisco Javier Mena Barraza
 * @version 1.0
 */
import java.net.*;
import java.io.*;
import java.util.Date;
import Utilities.*;
import Synchronization.*;

class NbodyWorker extends MyObject implements Runnable {

   private Utilidades u = new Utilidades();
   private int N = -1;
   private String masterMachine = null;
   private int portNum = -1;
   private EstablishRendezvous er = null;
   private int id = -1;
   private Cuerpo cp;

   public NbodyWorker(int id, String masterMachine, int portNum
       ) {
      super("NqueensWorker" + id);
      this.id = id;
      this.masterMachine = masterMachine;
      this.portNum = portNum;
      this.cp = new Cuerpo();
      er = new EstablishRendezvous(masterMachine, portNum);
      new Thread(this).start();
   }

   public NbodyWorker(int id, EstablishRendezvous er) {
      super("NbodyWorker" + id);
```

```
36          this.id = id;
37          this.er = er;
38          this.cp = new Cuerpo();
39          new Thread(this).start();
40          System.out.println("Worker "+id +" creado");
41
42      }
43
44
45      public void run() {
46          // Creamos un mensaje que contendra el resultado a
                regresar.---(Modify)
47          Message m = new Message(id, false, null, false,
48                  new Cuerpo(), -1,-1);
49
50          // Comenzamos con el trabajo.
51          while (true) {
52              Rendezvous r = er.clientToServer();
53              // Leemos algun mensae de el puerto definido en el
                    constructor.
54              m = (Message) r.clientMakeRequestAwaitReply(m);
55              // Cerramos la conexion
56              r.close();
57              // Validamos que el mensaje contenga trabajo para el
                    worker.
58              if (!m.containsWork) {
59                 if (masterMachine != null) { // remote workers
60                    System.exit(0);
61                 } else return;
62              }
63              //
                    ------------------------------------------------------------------

64              Cuerpo[] sistema = m.sistema;
65              double imto = m.incremento;
66              int ps = m.pos;
67              Cuerpo res = fuerzaT(sistema,ps);
68              res = mueveC(res,imto);
69              Cuerpo tmp = new Cuerpo();
70              tmp.setm(res.getm());
71              tmp.setr(res.getr());
72              tmp.setf(res.getf());
73              tmp.setv(res.getv());
74              //
                    ------------------------------------------------------------------

75              // Devolvemos el el trabsjo esperado
                    .---------------------(Modify)
76              m = new Message(id, true, null, false, tmp, ps, imto);
77          }
78      }
79
80
81
```

```
82
83
84
85          /**
86           * Metodo auxiliar encargado de calcular la fuerza de
                atraccion entre 2
87           * cuerpos
88           * @param pi Cuerpo 1er cuerpo a calcular.
89           * @param pj Cuerpo 2do cuerpo a calcular.
90           * @return fuerza Fuerza que ejerce el cuerpo pi sobre pj.
91           */
92          private double[] fuerza(Cuerpo pi, Cuerpo pj){
93              // Definimos la constante de gravitacion universal
94              double g = 667 * Math.pow(10,-11);
95              // Obtenemos la diferencia de las posiciones
96              double[] rij = u.resta(pi.getr(), pj.getr());
97              // Calculamos la longitud del vector anterior.
98              double rm = u.longitud(rij);
99              // Obtenemos la fuerza total.
100             double fm = (g * pi.getm() * pj.getm())/Math.sqrt(rm);
101             // calculamos un vector en la direccion del cuerpo.
102             double[] eij = u.productoE(rij,1/rm);
103             // Devolvemos el vector de la fuerza.
104             double[] fuerza = u.productoE(eij, fm);
105             return fuerza;
106         }
107
108         /**
109          * Metodo encargado de calcular la fuerza total que ejercen
110          * los n-1 cuerpos sobre un cuerpo determinado por le
                apuntador
111          * index.
112          * @param sist Cuerpo[] Arreglo de cuerpos que componen el
                sistema.
113          * @param index int Apuntador al cuarpo que deseamos
                evaluar.
114          * @return it Cuerpo con la variable fuerza actualizada.
115          */
116         public Cuerpo fuerzaT(Cuerpo[] sist, int index){
117             // Obtenemos el numero de cuerpos a evaluar.
118             int tcps = sist.length;
119             // Sacamos del sistema el cuerpo a evaluar.
120             Cuerpo it = new Cuerpo();
121             Cuerpo it2 = sist[index];
122             // Variable que elmacena los resultados parciales de
                    las fuerzas
123             // encontradas.
124             double[] fij = new double[3];
125             // Calclamos la fuerza del cuerpo it con los n-1
                    restantes.
126             for(int i=0; i<tcps; i++){
127                 if(i != index){
128                     // Para cada cuerpo  calculamos la fuerza de it
                            con el.
```

```
129                     Cuerpo sec = sist[i];
130                     double[] tmpF = fuerza(it2,sec);
131                     // Almacenamos el valor parcial de la suma.
132                     fij = u.suma(fij, tmpF);
133                 }
134             }
135             // Sumamos la fuerza del cuerpo con la fuerza de todos
                    los n cuerpos
136             // obtenidas.
137             double[] rs = u.suma(it2.getf(),fij);
138             // Actualizamos la variable fuerza del cuerpo evaluado.
139             it.setf(rs);
140             it.setm(it2.getm());
141             it.setr(it2.getr());
142             it.setv(it2.getv());
143             return it;
144         }
145 
146         /**
147          * Mï¿½todo auxiliar encargado de mover un cuerpo a la
                posiciï¿½n determinada
148          * en el instante "Incremento".
149          * @param pi Cuerpo Cuerpo a mover a una nueva posicion
150          * @param incremento double Instante al que se debe
                calcular la posicion.
151          * @return pi Cuerpo con la variable de posicion
                actualizada.
152          */
153         public Cuerpo mueveC(Cuerpo pi, double incremento){
154             double[] ai = u.productoE(pi.getf(), 1/(pi.getm()));
155             double[] dvi = u.productoE(ai, incremento);
156             double[] dri = u.productoE(u.suma(pi.getv(), u.
                    productoE(dvi, .5)),
157                         incremento);
158             pi.setv(u.suma(pi.getv(), dvi));
159             pi.setr(u.suma(pi.getr(),dri));
160             //pi.setf(new double[3]);
161             return pi;
162         }
163 
164         /*
165         public static void main(String[] args) {
166 
167           // parse command line options, if any, to override
                  defaults
168           GetOpt go = new GetOpt(args, "Ui:m:p:");
169           go.optErr = true;
170           String usage = "Usage: -i id -m master -p port";
171           int id = -1;
172           String masterMachine = "localhost";
173           int portNum = 9999;
174           int ch = -1;
175           while ((ch = go.getopt()) != go.optEOF) {
176              if      ((char)ch == 'U') {
```

```
177             System.out.println(usage);  System.exit(0);
178          }
179          else if ((char)ch == 'i')
180             id = go.processArg(go.optArgGet(), id);
181          else if ((char)ch == 'm')
182             masterMachine = go.optArgGet();
183          else if ((char)ch == 'p')
184             portNum = go.processArg(go.optArgGet(), portNum);
185          else {
186             System.err.println(usage);  System.exit(1);
187          }
188       }
189       System.out.println("NqueensWorker: id=" + id + ",
              masterMachine="
190          + masterMachine + ", portNum=" + portNum);
191       new NqueensWorker(id, masterMachine, portNum);
192    }
193    */
194 }// End class
```

Apéndice D

Clase NbodyManager

```
1
2
3   package ncuerpos;
4   /**
5    * Clase main.java
6    * Descripción: Clase encargada de llevar a cabo la simulación.
7    * @author Francisco Javier Mena Barraza
8    * @version 1.0
9    */
10  import java.net.*;
11  import java.io.*;
12  import java.util.Date;
13  import Utilities.*;
14  import Synchronization.*;
15
16  class NbodyWorker extends MyObject implements Runnable {
17
18     private Utilidades u = new Utilidades();
19     private int N = -1;
20     private String masterMachine = null;
21     private int portNum = -1;
22     private EstablishRendezvous er = null;
23     private int id = -1;
24     private Cuerpo cp;
25
26     public NbodyWorker(int id, String masterMachine, int portNum
           ) {
27        super("NqueensWorker" + id);
28        this.id = id;
29        this.masterMachine = masterMachine;
30        this.portNum = portNum;
31        this.cp = new Cuerpo();
32        er = new EstablishRendezvous(masterMachine, portNum);
33        new Thread(this).start();
34     }
35
```

```
36      public NbodyWorker(int id, EstablishRendezvous er) {
37         super("NbodyWorker" + id);
38         this.id = id;
39         this.er = er;
40         this.cp = new Cuerpo();
41         new Thread(this).start();
42         System.out.println("Worker "+id +" creado");
43
44      }
45
46
47      public void run() {
48         // Creamos un mensaje que contendra el resultado a
               regresar.---(Modify)
49         Message m = new Message(id, false, null, false,
50                 new Cuerpo(), -1,-1);
51
52         // Comenzamos con el trabajo.
53         while (true) {
54            Rendezvous r = er.clientToServer();
55            // Leemos algun mensae de el puerto definido en el
                  constructor.
56            m = (Message) r.clientMakeRequestAwaitReply(m);
57            // Cerramos la conexión
58            r.close();
59            // Validamos que el mensaje contenga trabajo para el
                  worker.
60            if (!m.containsWork) {
61               if (masterMachine != null) { // remote workers
62                  System.exit(0);
63               } else return;
64            }
65            //
                  ----------------------------------------------------------------

66            Cuerpo[] sistema = m.sistema;
67            double imto = m.incremento;
68            int ps = m.pos;
69            Cuerpo res = fuerzaT(sistema,ps);
70            res = mueveC(res,imto);
71            Cuerpo tmp = new Cuerpo();
72            tmp.setm(res.getm());
73            tmp.setr(res.getr());
74            tmp.setf(res.getf());
75            tmp.setv(res.getv());
76            //
                  ----------------------------------------------------------------

77            // Devolvemos el el trabsjo esperado
                  .--------------------(Modify)
78            m = new Message(id, true, null, false, tmp, ps, imto);
79         }
80      }
81
```

```
82
83
84
85          /**
86           * Metodo auxiliar encargado de calcular la fuerza de
                atracción entre 2
87           * cuerpos
88           * @param pi Cuerpo 1er cuerpo a calcular.
89           * @param pj Cuerpo 2do cuerpo a calcular.
90           * @return fuerza Fuerza que ejerce el cuerpo pi sobre pj.
91           */
92          private double[] fuerza(Cuerpo pi, Cuerpo pj){
93              // Definimos la constante de gravitación universal
94              double g = 667 * Math.pow(10,-11);
95              // Obtenemos la diferencia de las posiciones
96              double[] rij = u.resta(pi.getr(), pj.getr());
97              // Calculamos la longitud del vector anterior.
98              double rm = u.longitud(rij);
99              // Obtenemos la fuerza total.
100             double fm = (g * pi.getm() * pj.getm())/Math.sqrt(rm);
101             // calculamos un vector en la dirección del cuerpo.
102             double[] eij = u.productoE(rij,1/rm);
103             // Devolvemos el vector de la fuerza.
104             double[] fuerza = u.productoE(eij, fm);
105             return fuerza;
106         }
107
108         /**
109          * Metodo encargado de calcular la fuerza total que ejercen
110          * los n-1 cuerpos sobre un cuerpo determinado por le
                apuntador
111          * index.
112          * @param sist Cuerpo[] Arreglo de cuerpos que componen el
                sistema.
113          * @param index int Apuntador al cuarpo que deseamos
                evaluar.
114          * @return it Cuerpo con la variable fuerza actualizada.
115          */
116         public Cuerpo fuerzaT(Cuerpo[] sist, int index){
117             // Obtenemos el numero de cuerpos a evaluar.
118             int tcps = sist.length;
119             // Sacamos del sistema el cuerpo a evaluar.
120             Cuerpo it = new Cuerpo();
121             Cuerpo it2 = sist[index];
122             // Variable que elmacena los resultados parciales de
                    las fuerzas
123             // encontradas.
124             double[] fij = new double[3];
125             // Calclamos la fuerza del cuerpo it con los n-1
                    restantes.
126             for(int i=0; i<tcps; i++){
127                 if(i != index){
128                     // Para cada cuerpo  calculamos la fuerza de it
                            con el.
```

```
129                     Cuerpo sec = sist[i];
130                     double[] tmpF = fuerza(it2,sec);
131                     // Almacenamos el valor parcial de la suma.
132                     fij = u.suma(fij, tmpF);
133                 }
134             }
135             // Sumamos la fuerza del cuerpo con la fuerza de todos
                    los n cuerpos
136             // obtenidas.
137             double[] rs = u.suma(it2.getf(),fij);
138             // Actualizamos la variable fuerza del cuerpo evaluado.
139             it.setf(rs);
140             it.setm(it2.getm());
141             it.setr(it2.getr());
142             it.setv(it2.getv());
143             return it;
144         }
145
146         /**
147          * Método auxiliar encargado de mover un cuerpo a la
                posición determinada
148          * en el instante "Incremento".
149          * @param pi Cuerpo Cuerpo a mover a una nueva posición
150          * @param incremento double Instante al que se debe
                calcular la posición.
151          * @return pi Cuerpo con la variable de posición
                actualizada.
152          */
153         public Cuerpo mueveC(Cuerpo pi, double incremento){
154             double[] ai = u.productoE(pi.getf(), 1/(pi.getm()));
155             double[] dvi = u.productoE(ai, incremento);
156             double[] dri = u.productoE(u.suma(pi.getv(), u.
                    productoE(dvi, .5)), incremento);
157             pi.setv(u.suma(pi.getv(), dvi));
158             pi.setr(u.suma(pi.getr(),dri));
159             //pi.setf(new double[3]);
160             return pi;
161         }
162     }// End class
```

Definimosla forma en la cual se compilara

```
$ javac  Init.java
```

Apéndice E

Clase Message

```
1  package ncuerpos;
2
3  import java.net.*;
4  import java.io.*;
5  import java.util.Date;
6  import Utilities.*;
7  import Synchronization.*;
8
9  class Message implements Serializable {
10
11    public int workerID = -1;
12    public boolean containsResult = false;
13    public boolean containsWork = false;
14    public Date date = null;
15    //
          ------------------------------------------------------------(
          Modify)
16    public Cuerpo[] sistema;
17    public Cuerpo res;
18    public double incremento;
19    public int pos;
20    //
          ------------------------------------------------------------------
21
22    public Message(int workerID, boolean containsResult, Cuerpo
          [] sist,
23            boolean containsWork, Cuerpo result, int ps, double
                  dt) {
24        this.workerID = workerID;
25        this.containsResult = containsResult;
26        this.res = result;
27        this.sistema = sist;
28        this.containsWork = containsWork;
29        this.pos = ps;
30        this.incremento = dt;
```

```
31          this.date = new Date();
32      }
33
34      public String toString() {
35         return " workerID=" + workerID + ", containsResult="
36            + containsResult + ", Sistema= " + sistema + "\n
                containsWork="
37            + containsWork + ", Resultado= " +  res + ", Cuerpo="
                + pos
38            + " Momento: " + incremento + ", date=" + date;
39      }
40  }
```

Apéndice F

Clase NbodyManager

```
1  package ncuerpos;
2  import java.net.*;
3  import java.io.*;
4  import java.util.Date;
5  import java.util.Vector;
6  import Utilities.*;
7  import Synchronization.*;
8
9  class NbodyManager extends MyObject {
10
11    private static int N = 10000;
12    private static int numSolutions = 0;
13    private static int portNum = 9292;
14    private static EstablishRendezvous er = null;
15
16    public static void main(String[] args) {
17
18       // parse command line options, if any, to override
             defaults
19       GetOpt go = new GetOpt(args, "Uw:n:p:");
20       go.optErr = true;
21
22
23       String usage = "Usage: -w numWorkers -p port";
24       int ch = -1;
25       int numWorkers = 2;
26       while ((ch = go.getopt()) != go.optEOF) {
27          if      ((char)ch == 'U') {
28             System.out.println(usage);  System.exit(0);
29          }
30          else if ((char)ch == 'w')
31             numWorkers = go.processArg(go.optArgGet(),
                  numWorkers);
32          else if ((char)ch == 'p')
33             portNum = go.processArg(go.optArgGet(), portNum);
34          else {
```

```
35              System.err.println(usage);  System.exit(1);
36            }
37        }
38
39        // Generamos un vector que contencra el historico de
              todas las
40        // iteraciones realizadas.
41        Vector resultado = new Vector(N);
42
43        // Apuntador a las posiciones del vector.
44        int iterador = 0;
45
46        // Cargamos el arreglo que tendra almacenada la iteracion
              anterior.
47        Cuerpo[] hist = new Cuerpo [numWorkers];
48  //
      ------------------------------------------------------------------------
49        //Generamos la iteraciï¿½n incial
50  //
      ------------------------------------------------------------------------
51        // send out all the "work" (initial configurations)
52        if (numWorkers > 0) {
53            er = new EstablishRendezvous();
54            System.out.println("NbodyMaster: N=" + N
55                    + ", numWorkers=" + numWorkers);
56            for (int i = 0; i < numWorkers; i++) new NbodyWorker(
                  i, er);
57        } else {
58            System.out.println("NbodyMaster: N=" + N + ", portNum
                  =" + portNum);
59            er = new EstablishRendezvous(portNum);
60        }
61
62
63        Message m = null;
64        int numResultsReceived = 0;
65        // Generamos el sistema de N cuerpos
              ------------------------------
66        // Cuerpo[] rs = new Cuerpo[numWorkers];
67        Utilidades g = new Utilidades();
68        Cuerpo[] rs = g.parseXML("ncuerpos.xml");
69        // Almacenamos los resultados iniciales del sistema
70         resultado.addElement(rs);
71
72        //
              ------------------------------------------------------------------
73        for (int i = 0; i < numWorkers; i++) {
74            Rendezvous r = er.serverToClient();
75            m = (Message) r.serverGetRequest();
76            if (m.containsResult) {
77                hist[numResultsReceived] = m.res;
```

```
78                  numResultsReceived++;
79                  System.out.println("age2() =" + age() + " m:" + m
                        );
80              }
81              // Modificamos el mensaje enviado
                    -------------------------------
82              r.serverMakeReply(new Message(-1, false, rs, true,
                    null,i,i));
83              //
                    ------------------------------------------------------------------

84              r.close();
85          }
86
87
88          // tally up the returning counts
89          while (numResultsReceived < numWorkers) {
90              Rendezvous r = er.serverToClient();
91              m = (Message) r.serverGetRequest();
92              if (m.containsResult) {
93                  hist[numResultsReceived] = m.res;
94                  numResultsReceived++;
95                  System.out.println("age()=" + age() + " m:" + m.
                        toString());
96              }
97              r.serverMakeReply(new Message(-1, false, null, false,
                     null,0, 0));
98              r.close();
99          }
100
101
102         // Almacenamos la primera iteracion en el historial
103         resultado.addElement(hist);
104
105         // Nos movemos a la siguiente posicion del vector.
106         iterador++;
107
108 //
        ------------------------------------------------------------------------

109         // Comenzamos a iterar (n-1)-veces el sistema.
110 //
        ------------------------------------------------------------------------

111
112         for(int it = 0; it < (N-1); it++){
113             // Limpiamos la variable historial
114             rs = new Cuerpo[numWorkers];
115
116             er = new EstablishRendezvous();
117             System.out.println("NbodyMaster: N=" + N
118                     + ", numWorkers=" + numWorkers);
119             for (int i = 0; i < numWorkers; i++) new NbodyWorker(
                    i, er);
```

```
120
121             // send out all the "work" (initial configurations)
122             m = null;
123             numResultsReceived = 0;
124             // Generamos el sistema de N cuerpos
                    ------------------------------
125             rs = new Cuerpo[numWorkers];
126
127             for(int i = 0; i< numWorkers; i++){
128
129                //Obtenemos los vector de  posicion del estado
                       anterior del sistema
130                double[] r1 = hist[i].getr();
131
132                //Gbtenemos el vector de velocidad del estado
                       anterior del sistma
133                double[] v1 = hist[i].getv();
134
135                // Obtenemos el vector de fuerza del estado
                       anterior del sistema
136                double[] f1 = hist[i].getf();
137
138                // Obtenemos la masa del cuerpo i-esimo
139                double masa = hist[i].getm();
140                Cuerpo tmp1 = new Cuerpo(r1,v1,f1,masa);
141
142                // Colocamos el cuerpo creado en el sistema nuevo
                       a iterar.
143                rs[i]= tmp1;
144             }
145
146
147             // Limpiamos la variable historial
148             hist = new Cuerpo[numWorkers];
149             //
                    ------------------------------------------------------------

150             for (int i = 0; i < numWorkers; i++) {
151                 Rendezvous r = er.serverToClient();
152                 m = (Message) r.serverGetRequest();
153                 if (m.containsResult) {
154                     hist[numResultsReceived] = m.res;
155                     numResultsReceived++;
156                     System.out.println("age()=" + age() + " m:" +
                            m);
157                 }
158                 // Modificamos el mensaje enviado
                        --------------------------------
159                 r.serverMakeReply(new Message(-1, false, rs, true
                        , null,i,i));
160                 //
                        ------------------------------------------------------------

161                 r.close();
```

```
162             }
163
164
165             // tally up the returning counts
166             while (numResultsReceived < numWorkers) {
167
168                 Rendezvous r = er.serverToClient();
169                 m = (Message) r.serverGetRequest();
170                 if (m.containsResult) {
171                     hist[numResultsReceived] = m.res;
172                     numResultsReceived++;
173                     System.out.println("age()=" + age() + " m:" +
                            m.toString());
174                 }
175                 r.serverMakeReply(new Message(-1, false, null,
                        false, null,0, 0));
176                 r.close();
177             }
178
179              // Almacenamos la primera iteracion en el historial
180             resultado.addElement(hist);
181
182             // Nos movemos a la siguiente posicion del vector.
183             iterador++;
184
185
186             // System.out.println("age()=" + age()
187               //      + ", NbodyMaster: Iteraciones=" +
                      numSolutions);
188         }
189         er.close();
190         Utilidades file = new Utilidades();
191         file.genXML(resultado);
192         System.exit(0);
193     }
194 }
```