Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Computación de alto rendimiento en el estudio de reacciones químicas en solución a través de métodos híbridos clásico-cuánticos.

por
Pablo Listingart

Director de Tesis
Lic. Esteban Mocskos
Co-Director de Tesis
Dr. Mariano Camilo González Lebrero

Tesis para optar al grado de
Licenciado en Ciencias de la Computación

Mayo de 2006

Índice general

Agradecimientos

A mis padres, Marta y Horacio.

A Eli.

A Gabi.

A mi familia.

A Esteban.

A Nano y Darío

A mis amigos.

A la gente del LSC.

A los jurados.

A todos los que de alguna forma u otra contribuyeron para que yo llegue a este momento.

We would also like to thank the CHEPREO-FIU OSG Tier3 Center, CHEPREO / AMPATH Engineering Group for sharing its computational resources, enlarging our testbed.

”Imagination is more important than knowledge”

Albert Einstein

Resumen

En este trabajo se presenta un software de cómputo de alto rendimiento aplicado a la simulación de reactividad química en solución. El método utilizado para realizar la simulación se encuadra dentro de un esquema híbrido cuántico-clásico (QM-MM) en el cual el solvente es tratado clásicamente (como masas y cargas puntuales) mientras que el soluto es tratado con rigurosidad cuántica (DFT, en nuestro caso), de esta manera se logra optimizar el costo computacional ya que se calcula la estructura electrónica sólo de la porción del sistema en la que es realmente relevante. El trabajo consistió en la construcción de un software paralelo de simulación numérica a partir de una versión serial preexistente y su implementación en una computadora SMP y en un cluster Beowulf para el estudio y predicción de reacciones químicas en solución. La combinación del hardware y software mencionado, y el análisis de distintas estrategias de paralelización (incluyendo diversas modalidades de pasaje de mensajes) permitieron la resolución de problemas más realistas y previamente inabordables, obteniendo un algoritmo escalable.

Key Words: *Beowulf*, *Cálculo paralelo*, *Cluster*, *DFT*, *Métodos híbridos*, *n-body*, *Perfomance*

Capítulo 1

Introducción

La simulación en química es una herramienta cada vez más utilizada. Cuando nos referimos a *simulación* hablamos del uso de tecnología digital para resolver ecuaciones matemáticas que definen una teoría particular o un modelo [1]. El desarrollo de técnicas refinadas, así como el aumento geométrico del poder de cómputo son dos de las razones más importantes de su creciente uso. Ya sea como complemento o como reemplazo de experimentos, la simulación aporta, entre otras cosas, una visión microscópica inalcanzable por otra vía.

Por otra parte, la simulación numérica ha crecido de forma considerable en los últimos años debido a que, desde la perspectiva económica, el desarrollo teórico ha demostrado ser una forma viable de reducir los costos o enfocar más eficientemente la experimentación.

Las metodologías utilizadas en simulación en química varían según el problema a tratar. En general, se dividen en dos grandes ramas, la llamada rama *clásica* y la llamada *cuántica*.

La mecánica que rige el comportamiento de partículas pequeñas (como los electrones) es la llamada mecánica cuántica, que fue desarrollada a principios del siglo pasado por físicos ilustres como Einstein, Planck, Bhor y otros.

Lamentablemente, no existe una forma exacta de resolver las ecuaciones asociadas a esta mecánica, y las soluciones aproximadas accesibles son muy demandantes desde el punto de vista computacional. Esta elevada demanda hace prohibitivo el uso de esta técnica a sistemas con un número elevado de átomos.

Por otro lado la mecánica clásica, menos demandante, es incapaz de representar electrones, por lo que solo puede utilizarse si la distribución de los mismos no cambia considerablemente. No es posible, entonces, tratar reacciones químicas.

Si se quieren tratar reacciones en solución, se encuentra el problema de que el número de átomos necesarios para representar correctamente el sistema es demasiado grande como para tratarlo con rigurosidad cuántica, sin embargo, al haber reacción química, no se puede utilizar una metodología totalmente clásica. Por ello se han desarrollado las metodologías híbridas[2, 3, 4], donde a una pequeña porción del sistema (la parte reactiva), se la trata con la metodología cuántica (con detalle electrónico), mientras que al resto se lo trata como un campo de fuerzas clásico.

1.1. Teoría de los funcionales de la densidad (DFT)

1.1.1. Introducción a la Teoría del Funcional de la Densidad

En esta tesis se utiliza la teoría del funcional de la densidad (DFT) para el cálculo de la estructura electrónica en el esquema híbrido cuántico-clásico, por este motivo surge la necesidad de realizar una pequeña reseña de la misma [1].

La teoría de los funcionales de la densidad permite obtener la estructura electrónica (o cualquier otra propiedad deseada) de un sistema de electrones y núcleos. La idea original, postulada por Thomas y Fermi en la década del 20, es contemporánea al nacimiento del método Hartree-Fock [1, 5]. Sin embargo debido a su inexactitud a la hora de tratar átomos y moléculas, el método DFT permaneció relegado hasta mucho después. Recién en la década del 60 Hohenberg y Kohn propusieron dos teoremas [6] que posibilitaron un tratamiento riguroso de la estructura electrónica como funcional de la densidad.

Hasta ese momento, la teoría examinada vinculaba la energía y la densidad electrónica, sin embargo no preveía la manera de calcular una u otra. En el año 1965 Kohn y Sham [7] introdujeron una aproximación para la forma funcional de la energía cinética a través de la cual se hizo posible evaluar la densidad y obtener la energía total. Este método, denominado de Kohn-Sham, es el que permitió utilizar a DFT como un instrumento de cálculo de estructura electrónica eficiente. La enorme importancia que adquirió la teoría del funcional de la densidad en química se vio reflejada en el otorgamiento del premio Nobel al profesor Kohn en 1998.

1.1.2. Descripción de la DFT

Los métodos tradicionales en la teoría de estructura electrónica, en particular Hartree-Fock y los derivados del mismo, están basados en una complicada función de onda para varios electrones [8, 9, 10]. El objetivo principal de la DFT es reemplazar esta función de onda por la densidad electrónica como base. Mientras que la función de onda depende de $3N$ variables (tres variables espaciales por cada uno de los N electrones), la densidad es solo función de tres variables que es mucho más simple para trabajar tanto conceptual como prácticamente.

A pesar de que la DFT tiene sus raíces en el modelo de Thomas-Fermi, como hemos mencionado con anterioridad, no se utilizó realmente hasta que Hohenberg-Kohn propusieron dos teoremas. El primero de ellos demuestra la existencia de una relación biunívoca entre la densidad del estado fundamental (de menor energía) del sistema y la función de onda del estado fundamental de un sistema con múltiples partículas. Esta relación consiste en que la función de onda (y por lo tanto la energía y cualquier otro observable) se hallan unívocamente determinados por la densidad electrónica. Esta densidad da cuenta de la probabilidad de encontrar un electrón en un lugar dado del espacio. El segundo teorema prueba que la densidad del estado fundamental minimiza la energía electrónica total del sistema. Los teoremas originales se mantienen solo para el estado fundamental en ausencia de un campo magnético, aunque los mismos han sido generalizados. El primero de los teoremas solo garantiza que la relación entre densidad y función de onda del estado fundamental existe pero no provee de herramientas para calcular ninguna de las dos.

DFT se ha convertido en un método muy popular desde la década del 70 para cálculos de estado sólido debido a su relación de costo-calidad sobre otros métodos

de resolver el problema *n-body* en mecánica cuántica. Sin embargo no fue considerado suficientemente preciso en química cuántica hasta la década del 90 cuando se refinaron las aproximaciones de los mapeos para mejorar el modelado de las interacciones.

DFT es actualmente uno de los métodos más utilizados para estructuras electrónicas [11, 12, 13]. A pesar de las mejoras que ha ido sufriendo todavía existen dificultades para describir adecuadamente algunas interacciones intermoleculares.

1.1.3. Aplicaciones de DFT

La teoría de Kohn-Sham se puede aplicar de dos formas distintas, las cuales dependen de lo que se quiera investigar. En el estado sólido, se usan bases de onda plana con condiciones de contorno periódicas. Además, se pone un gran énfasis en mantener la consistencia con el modelo idealizado de gas electrónico uniforme, que exhibe un comportamiento similar al del sólido infinito.

En los estados líquido y gaseoso, el gas de electrones es un mal modelo para el comportamiento de átomos y moléculas discretas. Para tratar estos casos se ha desarrollado una enorme variedad de funcionales para aplicaciones químicas. Algunos de estos funcionales son totalmente inconsistentes con la aproximación del gas electrónico uniforme. A pesar de que los resultados obtenidos con estos funcionales suelen ser suficientemente precisos para la mayoría de las aplicaciones, no existe forma de mejorarlos. Por lo tanto no es posible estimar el error de los cálculos sin realizar comparaciones con otros métodos o experimentos, lo cual significa que igualmente deben utilizarse otras técnicas para comparar con el consiguiente costo que esto significa.

1.2. Métodos Clásicos

Describir con detalle electrónico un sistema químico es, muchas veces, imposible y/o innecesario. Existen muchos fenómenos que pueden simularse con modelos más crudos, en particular en procesos en los cuales la estructura electrónica no cambia considerablemente. En estos modelos las moléculas (o átomos) pueden ser representadas por masas puntuales, generalmente provistas de carga, que interactúan a través de un campo de fuerza clásico [14]. Para construir este campo de fuerza suelen utilizase los Potenciales Coulómbicos (de interacciones entre cargas, dipolos y, eventualmente, multipolos), para describir la interacción entre sitios cargados, de Lennard-Jones para describir las interacciones dispersivas y repulsivas de corto alcance, y potenciales armónicos para describir los términos intramoleculares (distancia y ángulos de enlace)[5]. En el sistema utilizado para la realización de esta tesis se emplean fundamentalmente modelos clásicos para describir el solvente en soluciones acuosas.

1.2.1. Modelos de agua

Un gran número de las reacciones químicas de interés ocurren en solución acuosa. Dentro de los ingredientes utilizados para modelar al agua se encuentran cargas puntuales fijas, cargas fluctuantes, dipolos puntuales, potenciales tipo Lennard-Jones, potenciales armónicos de enlace y otros. La utilización de dichos

ingredientes, así como la parametrización de los mismos, dependen en buena medida de qué propiedad interesa reproducir y con qué nivel de precisión en relación al costo computacional asociado [14].

Los modelos de agua utilizados en el código a paralelizar son el TIP4P y el TIP4P-FQ.

- En el modelo TIP4P se representa a la molécula de agua como un cuerpo rígido (sin grados de libertad internos) con cuatro centros, uno centrado en cada átomo y uno extra en el eje C_{2v} situado en la bisectriz del ángulo H-O-H (ver figura 1.1). Las moléculas de agua TIP4P interactúan entre sí mediante interacciones Coulómbicas entre cargas situadas en los hidrógenos y el sitio `M` y mediante un potencial de tipo Lennard-Jones centrado en el átomo de oxígeno.
- El modelo TIP4P-FQ es muy similar al TIP4P con la diferencia que las cargas parciales pueden fluctuar alrededor de un valor de equlilibrio, adaptándose así al campo eléctrico local. De este modo se puede modelar moléculas de agua cuyo momento dipolar depende del campo eléctrico local.

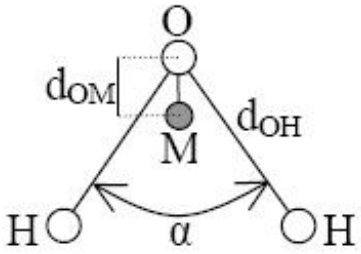

Figura 1.1: Esquema de distribución de sitios para los modelos de agua TIP4P y TIP4P-FQ.

La elección de los mencionados modelos de agua se debe a su utilización ampliamente difundida y a que reproducen correctamente geometrías y energías de enlace de dímeros y trímeros de agua con solutos.

1.3. El problema *n-body*

El problema *n-body* es aquel de encontrar, dada la posición inicial, masas y velocidades de n cuerpos, sus movimientos siguientes determinados por las leyes de la mecánica clásica [1, 15, 16].

Este problema puede referirse tanto a encontrar los movimientos de cuerpos celestes por medio de la gravedad como los de átomos y moléculas debido al potencial eléctrico. Usualmente, se cuenta con un estado inicial, y el problema consiste en ir determinando los estados subsiguientes, para observar cómo interactúan estos cuerpos en el tiempo.

Puede observarse parte de la dificultad del problema *n-body* analizando inicialmente los casos más sencillos del mismo. El problema, teniendo dos cuerpos, consiste en estudiar cómo interactúan los mismos a través del tiempo.

En el caso de estudio de dos cuerpos celestes que se encuentran bajo acción gravitatoria, si se considera que el centro común de masa de los dos cuerpos esta en reposo, cada cuerpo viaja en una sección cónica cuyo foco está en el centro de masa del sistema.

Si ambos cuerpos están ligados, ambos describirán elipses en sus trayectorias. La energía potencial relativa al estar alejados (cuyo valor siempre es negativo) tiene un valor absoluto menor que el total de energía cinética del sistema (no esta siendo tenida en cuenta la energía de rotación sobre sus ejes de los cuerpos).

Si ambos se están alejando, lo harán describiendo parábolas o hipérbolas. En el caso de la hipérbola, la energía potencial tiene un valor absoluto menor que el total de energía cinética del sistema, por lo tanto la suma de ambas energías es positiva. En el caso de la parábola, la suma de las energías es igual a cero. Las velocidades tienden también a cero a medida que los cuerpos se alejan. El hecho de que la trayectoria parabólica no tenga energía surge de la presunción de que la energía potencial gravitacional tiende a cero a medida que los cuerpos se alejan infinitamente.

A continuación se analiza como es el comportamiento del sistema con tres cuerpos. El mismo es bastante más complicado que el visto anteriormente. Su solución puede ser caótica. En general este problema no puede ser resuelto de forma analítica (en forma de fórmula cerrada con constantes conocidas y funciones elementales). Sin embargo existen soluciones aproximadas que pueden ser calculadas por métodos numéricos o de perturbación.

Un caso especial del problema de tres cuerpos es el llamado *restringido* que asume que la masa de uno de los mismos es despreciable. Este problema es un caso especial en el cual dos de los tres cuerpos estan en órbitas circulares o elípticas.

El problema restringido fue abordado extensamente por muchos matemáticos y físicos famosos, como Lagrange en el siglo 18 y Henri Poincaré al final del siglo 19.

El trabajo de Poincaré en el problema restringido de tres cuerpos resultó ser la base de la teoría del caos determinístico.

El problema que se presenta en este trabajo es similar al *n-body*. La simulación de un sistema de partículas es uno de los mayores problemas computacionales tanto en física como en química debido al tiempo que insume. Para analizar los movimientos de partículas deben primero calcularse los potenciales del sistema en su totalidad. El potencial en cada punto del sistema es igual a la suma de los potenciales determinados por las partículas.

A pesar de que la problemática con la que se trabaja es muy similar computacionalmente a la que se ha mencionado en esta sección, no se optó en esta etapa del trabajo por aplicar estas estrategias debido a que el programa serial ya estaba utilizando otra técnica de resolución y se optó por no cambiar totalmente la misma debido a la incertidumbre sobre los resultados que esto generaría. Sin embargo, existe la necesidad de mencionar que existen otras formas de resolución para situaciones similares. Pueden observarse distintos aspectos del problema en [1, 15, 16, 17]. En la literatura este problema se lo conoce, también, como *many-body problem*.

1.4. Métodos Híbridos QM/MM

Simular reacciones químicas en solución requiere la representación de un sistema con un gran número de partículas. En nuestro caso, como se verá más adelante, se han realizado pruebas con alrededor de 500 moléculas. Por razones de costo computacional y eficiencia, generalmente estos sistemas son tratados a nivel de mecánica molecular (MM)[14, 18, 19]. Sin embargo, el mayor problema que tiene la teoría de MM es que no puede ser aplicada al modelado de procesos que incluyen la formación o la ruptura de enlaces covalentes. Para modelar correctamente este tipo de procesos deben utilizarse métodos basados en la mecánica cuántica (QM)[20, 21].

La región del espacio a lo largo de una reacción en la cual ocurren cambios significativos en la estructura electrónica generalmente es muy pequeña comparado con el sistema en su totalidad. Por lo tanto, lo que se busca es hacer uso de las herramientas de mecánica cuántica en una región limitada para modelar el problema, mientras que el resto es modelado mediante mecánica molecular (ver figura 1.2) para reducir su complejidad. La base de esta técnica consiste en combinar cálculos de estructura electrónica con la mecánica molecular clásica a través de un Hamiltoniano híbrido (H_{qm-mm}). A estas metodologías híbridas se les da el nombre QM/MM (Quantum Mechanics/Molecular Mechanics) [1, 5].

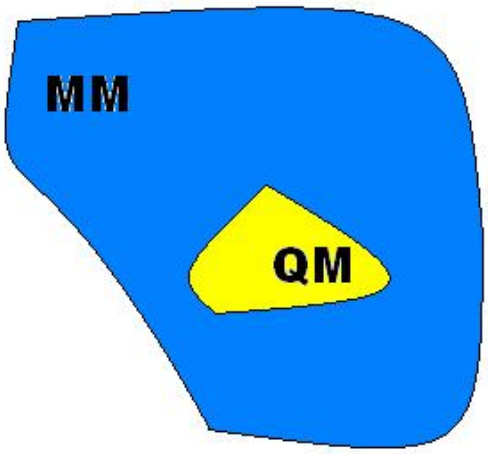

Figura 1.2: En esta figura se muestra como en sistemas con un gran número de partículas la generación y ruptura de enlaces puede estudiarse mediante la representación cuántica de una pequeña parte del mismo (Quantum Mechanics, QM), utilizando mecánica molecular (Molecular Mechanics, MM) para representar el resto.

En la versión serial del programa modificado en esta tésis se utilizó un método híbrido *QM/MM* [22]. Esta combinación de métodos permite el estudio de sistemas reactivos (que no se pueden tratar con Hamiltonianos clásicos) con un elevado número de átomos (que no pueden ser tratados con metodologías puramente cuánticas).

Este método propone que es posible escribir la energía como suma de contribuciones:

$$E[R_i, \tau_a] = E_{QM} + E_{QM-MM} + E_{MM}$$

Donde E_{QM} denota la energía del subsistema cuántico, E_{MM} la del subsistema clásico y E_{QM-MM} es la energía introducida por la interacción de ambos subsistemas. R_i y τ_a son las coordenadas de los núcleos clásicos y cuánticos respectivamente.

La energía correspondiente al subsistema cuántico puede calcularse con cualquier método de estructura electrónica. De la misma manera cualquier campo de fuerzas clásico puede aplicarse al subsistema correspondiente.

El esquema de aproximación implementado en la versión serial está basado en la teoría del funcional de la densidad (DFT) por su buena relación costo-calidad, en la cual la variable fundamental es la densidad electrónica.

Esta variable, ρ, es la relación entre densidad electrónica y la función de onda al cuadrado (este es uno de los postulados de la mecánica cuántica), donde la función de onda total se escribe como un producto antisimetrizado de orbitales.

La densidad queda entonces determinada de la siguiente manera:

$$\rho = \sum_{i=1}^{N} \mid X_i^{KS} \mid^2$$

Donde X_i^{KS} representa al orbital i de Kohn y Sham el cual está expresado como combinación lineal de gaussianas [7], por lo que la densidad queda determinada por una sumatoria de productos de las mismas.

La energía total del sistema puede describirse como:

$$E[\rho] = -\frac{1}{2}\langle X_i^{KS} \mid \sum_i \nabla_i^2 \mid X_i^{KS} \rangle + \int v(r)\rho(r)dr + \frac{1}{2}\int\int \frac{\rho(r_1)\rho(r_2)}{r_{12}} dr_1 dr_2 + E_{xc}[\rho]$$

Donde:

- $\rho(r)$ es la densidad que corresponde a los N electrones del sistema interactuantes sometidos al potencial real.
- ∇ es el operador laplaciano.
- r_i son las coordenadas del electron i.
- r es una generalización de las coordenadas.

El primer término de la fórmula corresponde a la energía cinética del sistema, el segundo es la atracción entre núcleos y electrones, el tercer término es la interacción de la densidad consigo misma y el último es la energía de intercambio y correlación. En la teoría de DFT, si se conociese el valor exacto de este último término, se podría conocer la estructura electrónica de una molécula sin problemas. Sin embargo, la dependencia de éste con la densidad es desconocida y la calidad del cálculo dependerá de la aproximación utilizada para obtener este término.

Finalmente, una clasificación importante para estos métodos híbridos es si el límite que separa QM de MM corta algún enlace covalente. Si esto no ocurre el modelado es simple, la unión de la mecánica molecular y la cuántica puede ser realizada sin mayores inconvenientes. Si, por ejemplo, se presenta una situación en la cual existen al menos un par de átomos, uno en la parte representada cuánticamente y otro en la mecánica, que poseen un enlace covalente entre ellos, la representación puede convertirse en algo intrincado para unir ambos subespacios.

Mas información sobre diversas simulaciones realizadas utilizando métodos QM/MM puede encontrarse en [23, 24, 25, 26, 27]

Capítulo 2

Simulación numérica paralela

2.1. Medición de performance de algoritmos paralelos

Al desarrollar programas de cálculo paralelo, se tiene como objetivo no solo alcanzar la mayor performance posible, sino que la misma se mantenga en proporción a medida que se agregan procesadores al cálculo. Para poder tener una noción de la medida de performance de los algoritmos paralelos, se definen las métricas de speedup, eficiencia, overhead y escalabilidad[28, 29, 30], que se introducen a continuación.

2.1.1. Speedup

El Speedup de un programa paralelo, es la relación entre la versión paralela de un algoritmo, para una cantidad dada de procesadores, y el tiempo de ejecución de la versión serial del mismo algoritmo. Para poder calcularlo, necesitamos definir primero:

- $T_1(n)$: Tiempo necesario para la ejecución de un programa serial, n corresponde a la entrada del programa.

- $T_{\mathcal{P}}(n, p)$: Tiempo necesario para la ejecución de la versión paralela del mismo programa con p procesadores.

Se define entonces como speedup de un programa paralelo ejecutándose en p procesadores a:

$$S(n, p) = \frac{T_1(n)}{T_{\mathcal{P}}(n, p)}$$

Esta medida sirve para comparar el rendimiento del algoritmo paralelo con respecto al serial. Cuando $S(n, p) = p$, se dice que se cuenta con un programa de *speedup lineal*.

El speedup lineal es el óptimo desde el punto de vista de la paralelización, puesto que indica una proporción directa y sin desperdicios entre el tiempo de ejecución y la cantidad de procesadores que intervienen. Es así que $0 < S(n, p) \leq p$.

Cuanto mayor sea el speedup de un algoritmo paralelo, mejor será la performance del mismo en la ejecución con p procesadores.

2.1.2. Eficiencia

La eficiencia de un programa paralelo, como su nombre lo indica, expresa el aprovechamiento de los recursos de procesador, por parte de dicho programa. Se define como:

$$E(n,p) = \frac{S(n,p)}{p} = \frac{T_1(n)}{p * T_{\mathcal{P}}(n,p)}$$

de donde se deduce que, para el caso óptimo de speedup lineal $E(n,p) = 1$. Para el caso de la eficiencia, las cotas son: $0 < E(n,p) < 1$. A medida que nuestro valor de eficiencia se acerque a 1, el programa tendrá mejor performance.

2.1.3. Overhead

Hay muchos factores que influyen negativamente en los tiempos de ejecución de un algoritmo paralelo, y son inherentes a la paralelización (no existen en un algoritmo serial). Algunos de estos factores son:

- Intercambio de mensajes entre los procesos
- Cálculos y decisiones adicionales requeridos en la versión paralela (identificación de ID, partición en subdominios, etc.)
- Tiempos de espera de un proceso (cuando necesita datos de un proceso vecino para poder continuar)
- Partes inherentemente seriales del algoritmo (No siempre se puede paralelizar la totalidad del algoritmo)

Es por esto, que existe el concepto de *overhead* de un algoritmo paralelo. Antes de definirlo, necesitamos definir el concepto de *trabajo* realizado por un algoritmo. En el caso de un algoritmo serial, el trabajo realizado por el mismo, es simplemente el tiempo que se necesita para su ejecución:

$$W_1(n) = T_1(n)$$

Cuando se trata de un algoritmo paralelo, el trabajo realizado por el mismo es la suma del trabajo de cada uno de los procesadores:

$$W_{\mathcal{P}}(n,p) = \sum_{i=1}^{p} W_i(n,p)$$

Donde $W_i(n,p)$ es el trabajo realizado por el programa ejecutado en el procesador i. Puesto que los tiempos de espera en cada procesador forman parte de su trabajo, resulta que $W_i(n,p) = T_{\mathcal{P}}(n,p)$, es decir que cada uno de los procesos tarda el tiempo total en terminar (haciendo algo útil o no). Con lo cual:

$$W_{\mathcal{P}}(n,p) = \sum_{i=1}^{p} W_i(n,p) = \sum_{i=1}^{p} T_{\mathcal{P}_i}(n,p) = p * W_i(n,p)$$

Con estas definiciones, decimos entonces que el overhead de un programa paralelo es el trabajo adicional que dicho programa realiza respecto a la solución serial equivalente:

$$T_{\mathcal{O}}(n,p) = W_{\mathcal{P}}(n,p) - W_1(n) = p * T_{\mathcal{P}}(n,p) - T_1(n)$$

El overhead, que está presente en todos los algoritmos paralelos, es el responsable de no poder lograr un *speedup lineal*. Al diseñar algoritmos paralelos, se trabaja para minimizar el overhead y poder lograr así un aumento del speedup.

2.2. Límites teóricos de la performance de un algoritmo paralelo

Si bien la paralelización de un algoritmo provee un aumento de la performance con respecto al algoritmo serial, este aumento no es necesariamente ilimitado. Inicialmente, Amdahl [29] evaluó que la máxima eficiencia que un algoritmo paralelo puede lograr, con respecto a su solución serial, está sujeta a la proporción paralelizable del algoritmo serial y no depende de la cantidad de procesos que intervengan en el cálculo. Esto es conocido como *Ley de Amdahl*.

Posteriormente, Gustafson [30] re-evaluó la ley de Amdahl, considerando que un problema paralelo puede *agrandarse* para mantener la performance a medida que se aumentan los procesadores. A continuación se mencionan ambos estudios.

2.2.1. Ley de Amdahl

Al analizar cualquier algoritmo paralelo, se encontrará que al menos una parte del mismo es inherentemente secuencial. Esto limita el speedup que se puede obtener por medio de la utilización de una computadora paralela. Si la fracción del problema que es inherentemente secuencial es f, entonces el máximo speedup que se puede obtener en una computadora paralela es $1/f$. Esto es, el tiempo de ejecución de la solución paralela será:

$$T_{\mathcal{P}}(n,p) = f * T_1(n) + \frac{(1-f)T_1(n)}{p}$$

Y por lo tanto, el factor de speedup será:

$$S(n,p) = \frac{T_1(n)}{T_{\mathcal{P}}(n,p)} = \frac{T_1(n)}{f * T_1(n) + \frac{(1-f)T_1(n)}{p}} = \frac{1}{\frac{p*f+(1-f)}{p}} = \frac{p}{1+(p-1)f}$$

Luego, aún contando con infinitos procesadores, el máximo speedup que se puede obtener está limitado por $1/f$.

$$\lim_{p\to\infty} S(n,p) = \frac{1}{f}$$

A su vez, la eficiencia de un programa paralelo se acercará a cero a medida que se incorporen nuevos procesadores, puesto que

$$\lim_{p\to\infty} E(n,p) = \lim_{p\to\infty} \frac{S(n,p)}{p} = 0$$

De acuerdo con la Ley de Amdahl y a modo de ejemplo, resulta que si un problema es solo 5 % serial, el máximo speedup que se puede obtener con una solución paralela es 20, independientemente de la cantidad de procesadores que se utilicen.

2.2.2. Ley de Gustafson

Gustafson, presenta un argumento para mostrar que la ley de Amdahl no es tan significativa como se pensaba con anterioridad en limitar el speedup potencial de una solución paralela. Este argumento se basa en la observación de que una computadora paralela de mayor tamaño, permite ejecutar un problema de mayor tamaño también, en un tiempo razonable.

De esta forma, se propone mantener fijo el tiempo de ejecución, pero incrementar el tamaño del problema y la cantidad de procesadores de la computadora paralela. Es así que, manteniendo constante el tiempo de ejecución, entonces el factor de speedup resultará diferente al que define Amdahl y se denomina *factor de speedup escalado*.

La ley de Gustafson se expresa de la siguiente forma: Sea s la fracción inherentemente serial de un algoritmo paralelo, y sea p la fracción paralelizada. Si se fija de forma constante en $s + p$ el tiempo de ejecución en un procesador, y para simplificar la explicación, se asume que $s + p = 1$. entonces, la ley de Amdahl establece que:

$$S(n) = \frac{s+p}{s+p/n} = \frac{1}{s+(1-s)/n}$$

(Nota: En este caso, la cantidad de procesadores está denotada por n).

Para utilizar el factor de speedup escalado que define Gustafson, se considera constante el tiempo de ejecución en paralelo en lugar del tiempo de ejecución serial. Ahora $s + p$ (la parte serial y paralela del programa, respectivamente) será el tiempo de ejecución en la computadora paralela y, nuevamente, para simplificar la explicación, se asume que $s + p = 1$.

Entonces, el tiempo que requiere la ejecución en una única máquina para resolver el mismo problema es $s + np$, puesto que la parte paralela debe ejecutarse en forma secuencial. El speedup escalado será:

$$S_s(n) = \frac{s+np}{s+p} = s + np = s + n(1-s) = n + (1-n)s$$

2.3. Arquitecturas paralelas

A continuación haremos un breve repaso de los paradigmas de programación paralela existentes al momento de encarar este trabajo. Luego, explicaremos más detalladamente el paradigma de *message-passing*, que es el que utilizamos para la implementación de la solución paralela de nuestro problema.

2.3.1. Arquitectura MIMD

Multiple-instruction-multiple-data es la clasificación en la taxonomía de Flynn para un procesador paralelo(una computadora con más de una unidad de procesamiento central usada para procesamiento paralelo) donde varias unidades funcionales(subsistemas de la CPU) realizan diferentes operaciones sobre datos distintos. Un ejemplo es una red de estaciones de trabajo.

La taxonomía de Flynn define las siguientes categorías

- Single instruction/single data stream (SISD) - Computadora secuencial.
- Multiple instruction/single data stream (MISD) - Inusual debido a que no tiene mucho sentido pues generalmente múltiples instrucciones requieren de múltiples datos para ser efectivas. Sin embargo, esta arquitectura se utiliza en algunas máquinas en las que se necesita paralelismo redundante.
- Single instruction/multiple data streams (SIMD) - Por ejemplo un *array processor*.
- Multiple instruction/multiple data streams (MIMD) - múltiples procesadores autónomos ejecutando simultáneamente diferentes instrucciones sobre diferentes datos.

2.3.2. Message passing (Pasaje de mensajes)

Message-passing consiste en un grupo de procesos que se comunican entre sí por medio del intercambio de mensajes para realizar una tarea determinada. En esta implementación hemos utilizado la biblioteca standard `MPI` (Message Passing Interface) [28, 31] sobre una arquitectura Beowulf [32].

Un cluster Beowulf es un conjunto de computadoras personales, no necesariamente homogéneas, interconectadas en red que comparten sus recursos de procesamiento y almacenamiento y pueden funcionar cooperativamente y en forma sincronizada para la ejecución de algoritmos paralelos diseñados especialmente. Estas computadoras, en comparación con las supercomputadoras de alta performance, tienen un costo mucho más accesible, ampliando las posibilidades de equipamiento de los investigadores e instituciones educativas. La posibilidad de coexistencia de diferentes modelos de procesadores y hardware, simplifican la escalabilidad de un cluster Beowulf.

En un cluster Beowulf, cada computadora o procesador se denomina $nodo$. De esta forma, se puede abstraer la arquitectura del mismo, aún en los casos de contar tanto con computadoras monoprocesador como multiprocesador en el mismo cluster. Aquí se asume que en cada procesador físico se corre un solo proceso.

Existen diversas aplicaciones de administración de procesos y aplicaciones en clusters Beowulf, muchas de ellas gratuitas y de código abierto. Las más importantes que se utilizan en el Laboratorio de Sistemas Complejos[33] son:

- `PBS`: Administrador de recursos de un cluster. Se utiliza para establecer políticas sobre la asignación y utilización de los recursos (principalmente procesadores) de un cluster. Provee colas de procesos, herramientas para encolar, visualizar y desencolar procesos, herramientas de monitoreo, etc.[34]
- `MAUI Scheduler`: Encolador avanzado de procesos. Se encarga de administrar la cola de procesos, para permitir la aplicación de prioridades, límites de ejecución, permisos, etc. [35]

Capítulo 3

Metodología

El desarrollo de un programa paralelo tiene como objetivo no sólo optimizar una única métrica, como puede ser el tiempo de ejecución total. Un buen diseño debe, además, optimizar una función dependiente del problema a resolver donde se tienen en cuenta el tiempo de ejecución, los requerimientos de memoria, los costos de implementación y mantenibilidad de la solución, entre otros. Estas optimizaciones en el diseño conllevan una tensión entre simplicidad, performance, portabilidad y otros factores.

Partiendo de un programa serial pre existente [36] y para poder decidir cómo encarar estas optimizaciones, se debió realizar un estudio de la complejidad y del comportamiento del programa serial. Para ello se realizó un profiling utilizando diversas herramientas provistas por el sistema operativo Linux y por el compilador `g77` utilizado (son descriptas más adelante).

Como ya fue mencionado, en la versión serial se utilizó un método híbrido *QM/MM* [22]. Mediante la combinación de métodos cuánticos y clásicos es posible el estudio de sistemas reactivos (que no se pueden tratar con Hamiltonianos clásicos) con un elevado número de átomos (no pueden ser tratados con metodologías puramente cuánticas).

Se puede observar en el capítulo 1 que el método QM/MM describe la energía como suma de contribuciones

$$E[R_i, \tau_a] = E_{QM} + E_{QM-MM} + E_{MM}$$

Esta fórmula indica que el total de energía del sistema es equivalente a la suma de energía del subsistema cuántico (E_{QM}), la del subsistema clásico (E_{MM}) y la energía introducida por la interacción de ambos subsistemas (E_{QM-MM}).

La energía correspondiente al subsistema cuántico puede calcularse con cualquier método de estructura electrónica. De la misma manera cualquier campo de fuerzas clásico puede aplicarse al subsistema correspondiente.

Como se mencionó con anterioridad, el esquema de aproximación implementado en la versión serial está basado en la teoría del funcional de la densidad (DFT) por su buena relación costo-calidad, en la cual la variable fundamental es la densidad electrónica. También se menciona en el capítulo 1 que la energía total se puede describir como se encuentra en la ecuación 1.4

La parte más costosa del cálculo esta relacionada con los dos últimos términos de esta expresión, que corresponden a la energía de repulsión de Coulomb, y el término de intercambio y correlación.

El primer término es calculado en forma analítica, pero su costo es del orden de N^3, donde N está directamente relacionada con el tamaño del sistema. Este término consiste de N^3 integrales analíticas que se pueden calcular en forma independiente, por lo que se puede paralelizar con facilidad.

El segundo término se debe evaluar en forma numérica, empleando un esquema de integración en una grilla centrada en los átomos y también escala como N^3. Este término es paralelizable entre los distintos procesos separando el dominio sobre el que se calcula la integral.

En el profiling realizado veremos que el cómputo de estos dos términos representa más del 90 % del costo computacional total, ya que el costo de los dos primeros términos escala con N^2.

3.1. Profiling

Para obtener un profiling de un programa se cuenta con distintas herramientas, a nivel sistema operativo y a nivel compilador. Detallamos a continuación aquellas que hemos utilizado.

- Comando *time*

  Este es un comando provisto por los sistemas operativos basados en Unix que recolecta datos básicos sobre la performance y la utilización de recursos de un programa.

  Corriendo un programa con el comando *time*, a su terminación se obtiene una línea con información de los tiempos y recursos utilizados.

  Por ejemplo:

  ```
  demo% time miprograma
  ```

  a la terminación del programa se obtiene por ejemplo:

  ```
  demo% 451.48 user 0.53 system 7:49.20 elapsed 96%CPU
  (0 avgtext + 0 avgdata 0 maxresident) k 0inputs + 0
  outputs (1109 major + 3084 minor) pagefaults 0 swaps
  ```

  Donde la interpretación es la siguiente:

  - *User*: 451.48 segundos de CPU ejecutando código en modo de usuario, no necesita permisos especiales para lo que debe realizar.
  - *System*: 0.53 segundos de CPU ejecutando código en modo de sistema, (en modo de kernel). Ejecución de instrucciones de proceso.
  - *Wallclock*: 7:49.20 minutos para completar el procesamiento completo.
  - *Recursos*: 96 % de consumo de CPU en la corrida.
  - *Memoria*: (0 avgtext + 0 avgdata 0 maxresident)k. Promedio de memoria total utilizada en el procesos dividida en datos, stack y texto.
  - *I/O*: 0 lecturas, 0 escrituras

- *Paginación*: (1109 major + 3084 minor)pagefaults, 0 swaps. Número de páginas no encontradas durante la ejecución del proceso. Éstas faltas ocurren cuando la página requerida ya no se encuentra en memoria principal (major) o cuando las páginas encontradas no son válidas (minor). El swap se refiere a la cantidad de veces que el proceso ha sido quitado de memoria principal.

- Comando `gprof`

  El comando `gprof` provee un análisis postmortem de tiempos a nivel de subrutinas, incluyendo cuántas veces una subrutina fue llamada, por quién fue llamada, y cuánto tiempo fue insumido por cada subrutina y las llamadas por ésta.

  Para activar la opción de este tipo de profiling se debe compilar el programa con los flags `-gp`.

  ```
  demo% f77 -o miprograma -pg miprograma.f ...etc
  ```

  Luego, se ejecuta el programa normalmente. Esta ejecución va a generar automáticamente un archivo `gmon.out` con todos los datos de profiling.

  Luego se ejecuta:

  ```
  demo% gprof miprograma
  ```

  y se obtiene como resultado un reporte con el análisis completo.

- Comando *tcov*

  El comando *tcov* produce un profiling de un nivel más interno, ya que recolecta información sentencia por sentencia del programa fuente, mostrando como resultado cuántas veces se ejecutó cada sentencia durante una corrida completa. También produce información sobre la estructura básica de los bloques del programa.

  Para activar la opción de este tipo de profiling se debe compilar el programa con los flags *-a, -xa* o-*xprofile=tcov*.

  ```
  demo% f77 -a -o miprograma miprograma.f ...etc
  ```

  Luego, se ejecuta el programa normalmente. Esta ejecución va a generar automáticamente un archivo `.d` por cada archivo `.f`. Para ver la información de profiling se ejecuta el comando `tcov` sobre los archivos fuentes:

  ```
  demo% tcov miprograma.f
  ```

  Como output de este comando se obtiene, finalmente, un archivo `.tcov` por cada archivo fuente. Estos archivos son una versión anotada del código donde adelante de cada sentencia se especifica la cantidad de veces que fue ejecutada.

A continuación observaremos los datos obtenidos de la realización del profiling del programa serial. En la figura 3.2 se detallan los porcentajes de procesamiento que insume cada subrutina en una corrida.

Con estos resultados pudimos identificar cuáles eran las rutinas que convenía analizar para evaluar su posible paralelización.

Se puede observar en la figura 3.2 que si bien la rutina `dnsg` insume un $62{,}5\,\%$ del tiempo de procesamiento total, esto se debe a la cantidad de veces que es invocada, ya que en cada invocación su tiempo de procesamieto es casi cero, es

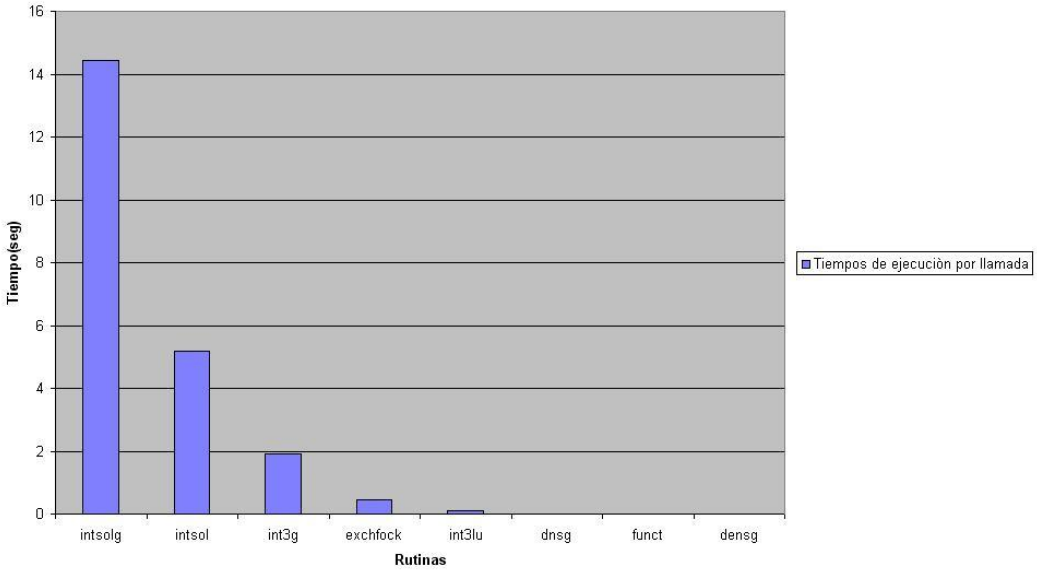

Figura 3.1: Tiempo consumido por las rutinas.

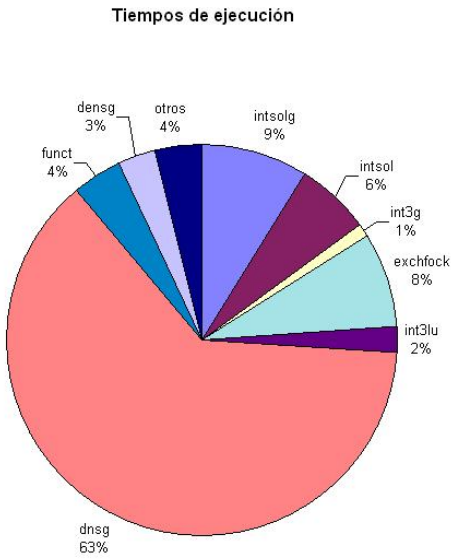

Figura 3.2: Porcentajes de tiempo consumidos por las rutinas.

decir despreciable respecto de otras. Lo mismo ocurre con las rutinas `exchfock`, `funct` y `densg`.

Siguiendo con los resultados del profiling realizado, se buscaron las llamadas correspondientes a las rutinas que calculan la densidad electrónica ya que las mismas en las corridas son las que insumen mayor cantidad de tiempo de procesamiento. Al realizar esta búsqueda se encontraron las rutinas `exchfock`, `exhnum`, `exhnumop`, `exchnum2` y `exchnum2op`.

Es en estas cinco últimas rutinas es en las que se centra nuestro análisis de paralelización. Se podrá observar más adelante que las cinco efectúan el mismo tipo de cálculo, variando sólo en algunos detalles internos como ser utilización o no de una matriz resultado, o la cantidad de parámetros de salida que tienen.

3.2. Implementación

En primera instancia se reorganizó el código de manera de eliminar vicios de programación y lograr evitar ciertos problemas que pueden traer aparejadas algunas reglas de utilización del fortran. Por ejemplo, al utilizar declaración implícita de variables si el programador declara que las variables que empiecen con letras de la `A` a la `H` sean del tipo `REAL*8` pero luego ante un descuido declara otra variable `INTEGER` con algún nombre que empiece entre esas letras podrán darse errores o diferencias en los resultados. En etapas posteriores se fueron eliminando las declaraciones implícitas de variables y se constató mediante un *dump* del programa que no se hubiera modificado nada respecto a la versión serial.

La paralelización del algoritmo se realiza a través de la descomposición del dominio de la integral presente en el tercer término de la ecuación 1.4 (página 11).

Cada uno de los procesos participantes, tiene en memoria el dominio completo y realiza sobre el subdominio correspondiente sus cálculos en forma local. Finalmente, como es necesario para las operaciones subsiguientes contar con el total de los datos en cada uno de los procesos, se realiza la sincronización de cada uno de ellos contra todos los restantes.

Como se mencionó con anterioridad al realizar el profiling se observó que entre un $65\,\%$ y $75\,\%$ del tiempo era consumido por las rutinas de tipo `densg` que son las encargadas de calcular los gradientes necesarios para obtener el valor en cada punto de la grilla de trabajo.

Esta rutinas en sí consumen muy poco tiempo de ejecución, pero el problema es que son llamadas en el orden de los millones de veces.

Por este motivo, se buscaron las rutinas desde donde `dnsg` y las otras rutinas similares eran llamadas. Esto ocurre en las funciones de tipo `exch` (`exchfock`, `exchnum`, `exchnum2`, `exchnum2op`). En las mismas se observó que las llamadas se encontraban dentro de tres ciclos, con los cuales se armaban los puntos de la grilla para cada átomo del sistema como puede observarse en la figura 3.3.

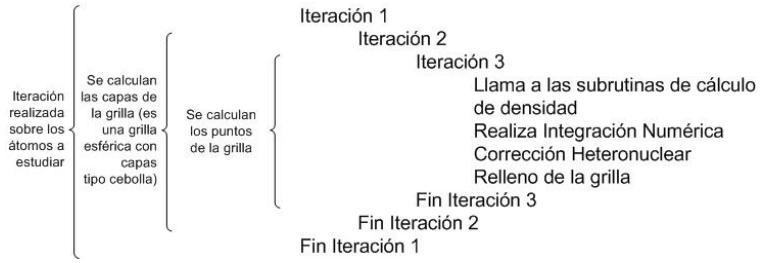

Figura 3.3: Iteraciones ejecutadas para completar la grilla RMM. Se optó por paralelizar el código a partir de la Iteración 1, la cual ejecuta los loops internos para cada uno de los átomos a analizar.

Se decidió paralelizar el algoritmo por el ciclo más externo, que justamente

es el que itera por cada átomo, para evitar sobrecarga de comunicación entre los nodos.

En este caso, si el número de procesos utilizados es menor o igual que el número de átomos con los que se trabaja se dividen los mismos de manera uniforme. En cambio, si hay más procesos que átomos, todo el procesamiento va a un solo nodo, lo cual produce pérdidas con respecto al modelo original por la introducción de instrucciones de la librería `MPI`, operaciones que no son utilizadas a ningún fin práctico con un solo proceso (como ser inicialización de acumuladores) y la inicialización del sistema paralelo.

En figura 3.4 se muestra como se paraleliza el sistema en caso de tener una entrada con 7 átomos y dos procesos paralelos. La matriz `RMM` es utilizada para calcular la integral y los subsiguientes `Atomo` i corresponden a matrices de datos calculadas en la iteración perteneciente a cada átomo y que se le van sumando al valor original de `RMM` correspondiente. Las flechas azules indican la transición de estados y las verdes indican el intercambio de mensajes y datos entre procesos.

En esa figura se ve como, en vez de realizar la sumatoria del vector RMM original más los siete átomos se toman dos conjuntos, se realizan las sumatorias parciales y luego se suman los resultados para obtener el valor final. Se puede observar que, por un lado, se hacen las sumatorias parciales: por ejemplo en el proceso 0 se suman RMM más los tres primeros átomos y por otro lado en un acumulador se guarda solamente la sumatoria de los átomos. Esto es así para poder luego enviar el contenido de los acumuladores a los otros procesos y que RMM no sea sumado N veces (con N igual al número de procesos), sino una única vez.

En cuanto a las modificaciones realizadas al programa, se agregaron sentencias de `MPI` [28, 37] para la inicialización y finalización de la utilización de la librería que lo implementa. Se utilizó la librería `MPICH`[38] para las pruebas realizadas.

En cada uno de los archivos `exch` se agregaron instrucciones para calcular el número de procesadores a utilizar de acuerdo a los átomos presentes en el sistema, y otras para compartir los resultados parciales obtenidos con el resto de los procesos para luego poder construir el resultado final.

Las llamadas a rutinas de salida debieron ser eliminadas para los procesos distintos al *root* para que el mismo fuera el único en realizar la escritura de datos de salida. Esto fue realizado incluyendo rutinas de `MPI` en todas las subrutinas que realizaban actividades de I/O para poder preguntar que número de proceso era sobre el que se corrían estas rutinas. En caso de que el proceso fuera distintos del *root* (proceso 0) no se ejecutan las instrucciones de entrada/salida.

3.3. Problemas surgidos durante la implementación

Uno de los primeros problemas surgidos durante la implementación del software paralelo fue la configuración y utilización de la librería `MPI` bajo Fortran. En el archivo main.f se agregaron las instrucciones de inicialización de `MPI`, sin embargo, en las subrutinas del programa las contantes definidas en la librería `MPIF` no eran reconocidas, lo cuál producía errores al ejecutar el software.

Un ejemplo de este caso se produjo con el comunicador `MPI_COMM_WORLD` que se utiliza para enviar mensajes a todos los procesos involucrados en la ejecución. Al no ser reconocido, no había forma de enviar y recibir mensajes entre procesos lo cuál producía un error que produciía el fin de la ejecución. Esta situación

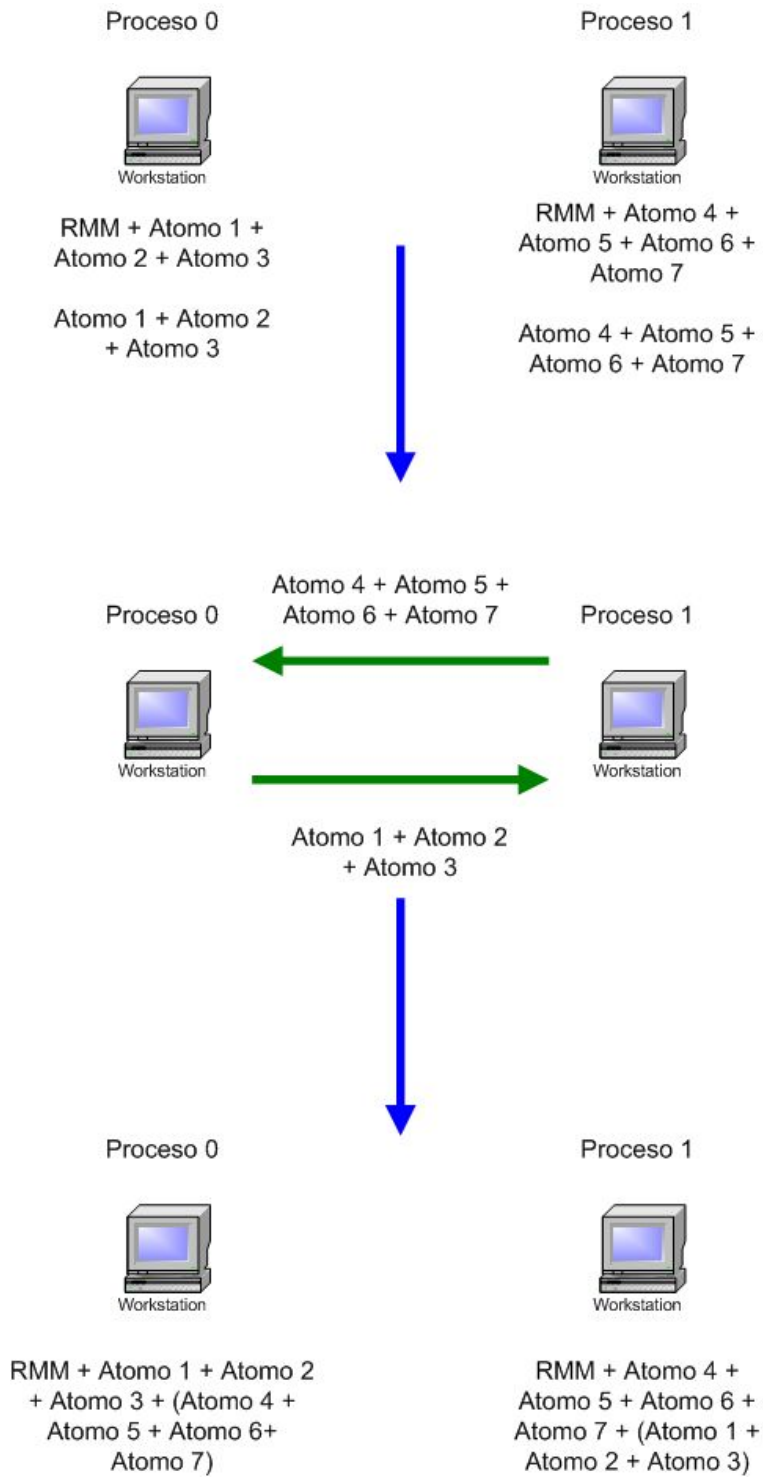

Figura 3.4: Forma en que es paralelizado el dominio de 7 átomos en dos procesos.Las matrices `RMM` y `Atomo i` son utilizadas para calcular la integral.Las flechas azules indican la transición de estados y las verdes indican el intercambio de mensajes y datos entre procesos.

se solucionó, inicialmente, pasando el comunicador como parámetro a todas las rutinas que pudieran llegar a utilizarlo. Luego, se descubrió que el sistema reconocía al valor 91 (`MPI_COMM_WORLD`) como variable global pero no así a su nombre de alias. Por lo tanto se quitaron los pasajes del comunicador como parámetro y se

utilizaron directamente los valores integer correspondientes.

Ocurrieron problemas similares con otro tipo de constantes definidas en `MPIF` pero una vez encontrado el error fue simple solucionarlo en situaciones posteriores. Finalmente el problema fue solucionado incluyendo la sentencia de preprocesamiento `include mpif.h` en todos los archivos donde se utilizan instrucciones de `MPI`. De esta forma el compilador reconoció todas las constantes con sus alias y no fue necesario utilizar los integer correspondientes.

El motivo por el cual se optó por incluir la sentencia de pre-procesamiento `include mpif.h` (que inserta el contenido de `mpif.h`) en lugar de continuar utilizando las constantes numéricas correspondientes fue para mejorar la portabilidad del programa. Distintos proveedores de implementaciones de librerías `MPI` como ser `MPICH`[38] o `LAM`[39] le asignan distintos valores a las constantes, por lo cual en caso de pasar de uno a otro en vez de utilizar el alias y reemplazarlo por el valor correspondiente automáticamente el compilador, uno debería recorrer todo el código poniendo el número correcto, lo cual no solo no es práctico sino que es una práctica aberrante de programación que no debería ocurrir.

Otro problema surgido fue con el pasaje de resultados mediante mensajes al proceso *root* (el proceso 0) para que el mismo continúe realizando los cálculos pertinentes. En los archivos modificados se deben pasar vectores al proceso root para que el mismo continúe con los cálculos y luego el mismo debe enviar también vectores con información al resto de los procesos. Sin embargo, en una primera instancia no se lograba pasar un vector completo mediante directivas `Send/Receive` por lo cual se enviaba por mensajes de a un elemento del vector por vez. Esto estaba lejos de ser lo óptimo debido al overhead generado por el envío de mensajes. Se comprobó que el problema se debía a una mala configuración y utilización de `MPICH` que luego de ser solucionado permitió pasar los vectores en forma completa.

Por otra parte, en busca de una mejora en la performance del programa, se probaron diversas formas de realizar el intercambio de información entre procesos. Como se mencionó con anterioridad la primera implementación incluía el envío de cada uno de los valores pertenecientes a los vectores por forma separada mediante mensajes bloqueantes utilizando el par `Send/Receive`. Esto producía que el tiempo de ejecución del programa paralelo excediera por mucho los tiempos del serial. Mandar los vectores completos por medio de las instrucciones `Send/Receive` redujo los tiempos de ejecución, pero aún así debido a la cantidad de procesos y topologías formadas entre ellos para el envío de mensajes, la performance estaba lejos de los esperado. Finalmente se utilizaron instrucciones de comunicación colectivas para reducir los tiempos de comunicación. Fueron utilizadas las instrucciones `MPI_Bcast` y `MPI_Allreduce`.

La instrucción `MPI_Bcast` realiza una comunicación colectiva donde un proceso envía los mismos datos a todos los procesos presentes en el comunicador. De acuerdo a la cantidad de procesos presentes en el comunicador `MPI` optimiza el envío de acuerdo a la topología que puede haber. Por ejemplo, puede envíar los mensajes con una topología de árbol binario, lo cual reduce el tiempo de envío de lineal a logarítmico.

Lo mismo sucede con la instrucción `MPI_Allreduce`, que genera una topología especial entre procesos para realizar una operación determinada entre términos que están presentes en cada uno de los procesos, y deja el resultado de la operación presente en todos los procesos del comunicador. Esto permite por ejem-

plo que si los términos de una sumatoria están repartidos entre todos los procesos, por medio de esta única instrucción se le puede indicar que realice la sumatoria y deje el resultado en todos los procesos en una variable de destino. Estas instrucciones se comportan en forma mucho más eficiente que si uno realizara las mismas operaciones por medio de pares `Send/Receive`.

Por medio de la comunicación colectiva se logró bajar considerablemente el tiempo de ejecución de manera de que la performance del programa paralelo fuese mejor que la del serial, lo cual era el objetivo de este trabajo. Los resultados se verán más adelante en la sección correspondiente.

Por último, cuando se creía que se había logrado el objetivo de desarrollar un programa paralelo que se comporte de manera similar al serial pero con una mejor performance, se tuvo un problema que fue determinante en la realización de este trabajo. El mismo surgió cuando se observó que los resultados obtenidos en la matriz `RMM` diferían en el programa paralelo con respecto al serial. Esto pudo ser observado por medio de la realización de una rutina que permite guardar mediante un archivo de texto una cantidad de variables que creímos convenientes analizar para determinar la correctitud de los cálculos, lo que denominaremos como *dump* o *estado* del programa.

En un principio se pensó que el problema era debido a un error en la manera de realizar el cálculo, pero luego de un análisis se confirmó que el mismo era correcto. Para tal efecto se fueron guardando los diversos términos participantes en las sumatorias y los mismos fueron sumados “a mano”. A partir de la confirmación de la correctitud de cálculo se intuyó que el problema podría corresponder al redondeo realizado sobre números de punto flotante, debido a que la diferencia era de unos pocos decimales. Para corroborar si el problema era de redondeo se implementaron y realizaron una batería de tests. En todos los casos de test se utilizó un entorno con 4 átomos.

- Se corrió el algoritmo serial y el paralelo, y se realizó una comparación entre los dumps generados por ambos. Así se detectó una inconsistencia en los valores del vector RMM.
- Se guardaron los valores de RMM calculados por el algoritmo serial pero separados en los atomos 1-2 y 3-4. O sea, inicialmente se calcularon con el programa serial las primeras dos iteraciones. Luego se realizó la corrida con los 4 átomos pero solo guardando el resultado de los átomos 3 y 4.
- Se guardó el estado calculado por el algoritmo paralelo (corrido en 2 procesos) correspondiente al proceso 0 que calcula los atomos 1-2 y al proceso 1 que calcula 3-4.
- Se compararon los resultados correspondientes y se observó que los calculos de RMM eran iguales.

De esta manera se concluyó que el problema radicaba en la unión de los resultados obtenidos en los distintos procesos del programa paralelo.

Una vez reducido el área donde podía ocurrir el problema se procedieron a realizar otros tests para determinar finalmente el problema. Para tal efecto se realizó una modificacion al programa serial para que calcule en RMM el resultado de los 2 primeros átomos, y los últimos dos los guarde en un acumulador. Finalmente se unió el vector RMM con el acumulador por medio de sumas. Esto hizo

que el programa serial diera el mismo resultado que el paralelo. De esta manera se demostró que el problema se debe a la limitación generada por la finitud de los números representados en la computadora, lo cual es un problema ampliamente conocido en los métodos numéricos. Ver figura 3.5

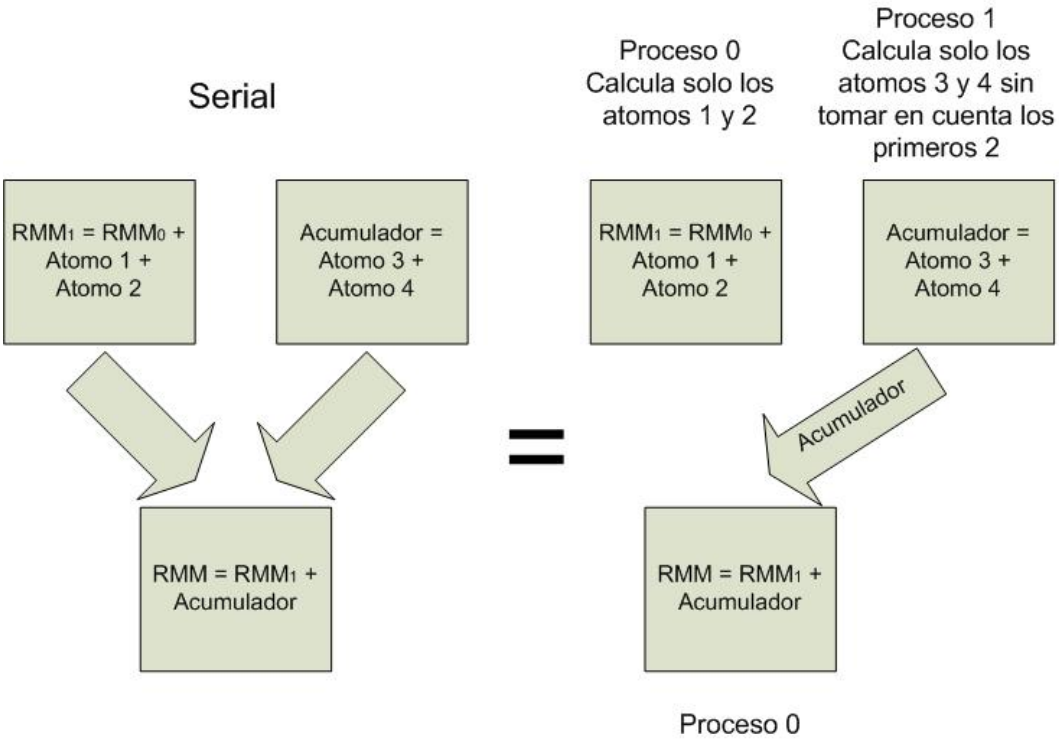

Figura 3.5: El algoritmo serial, separado en dos fases y unido por medio de sumas otorga el mismo resultado que realizando la cuenta en paralelo, demostrando que el error no se debe a que los valores de los últimos átomos dependen de los primeros, sino que son independientes y el problema es de redondeo.

3.3.1. Errores de punto flotante surgidos durante la implementación

En el mundo matemático tradicional puede haber números con una cantidad infinita de dígitos no periódicos. La aritmética que empleamos en este mundo define $\sqrt{3}$ como el único número positivo que cuando se multiplica por sí mismo produce el entero 3.

Sin embargo, en el mundo de la computación todo número representable tiene sólo un número fijo y finito de dígitos. Puesto que $\sqrt{3}$ no tiene una representación de dígitos finitos, en el interior de la computadora se le da una representación aproximada cuyo cuadrado no será exactamente 3, aunque estará lo bastante cerca a 3 para resultar aceptable en la generalidad de los casos. Casi siempre la representación y aritmética computacionales son satisfactorias y pasan inadvertidas o sin problemas. Pero no siempre es así, y en este caso la diferencia producida es amplificada de forma tal de que finalmente el resultado obtenido sea inutilizable[40].

El error de redondeo se da cuando usamos una calculadora o computadora para efectuar cálculos con números reales. El error surge porque las operaciones aritméticas realizadas en una máquina incluyen exclusivamente números finitos de dígitos, de manera que los cálculos se llevan a cabo con representaciones aproxima-

das de los números reales. En una computadora común, sólo un conjunto relativamente pequeño de números reales se utiliza para representar a todos estos números. El subconjunto que contiene únicamente números racionales, tanto positivos como negativos, y almacena una parte fraccionaria es denominada mantisa, junto con una parte exponencial conocida con el nombre de característica. La composición de un número de punto flotante según el IEEE es un dígito binario para el signo, ocho dígitos para el exponente y 23 para la mantisa [41]. Un número de doble precisión (como los utilizados para los nuestros cálculos) está compuesto por un dígito para el signo, once para el exponente y 52 para la mantisa.

No solo la representación de los números es poco exacta, la aritmética efectuada en una computadora también lo es. Esta aritmética requiere el manejo de los dígitos binarios con varias operaciones lógicas o de traslación. Si una representación o cálculo con dígitos finitos introduce un error, éste crece aún más cuando lo dividimos entre un número de pequeña magnitud (o equivalentemente multiplicando por un número de gran magnitud).

3.4. Resoluciones a los errores de punto flotante en la implementación

En esta sección se plantean dos soluciones ampliamente conocidas en la literatura para solucionar el tipo de problema presentado anteriormente. Luego en la sección 4 se explicará porqué la primera solución presentada no resultó satisfactoria y se avanzó sobre la segunda.

3.4.1. Ampliación de la precisión

Para solucionar los problemas de precisión en punto flotante inicialmente se optó por duplicar el tamaño de las variables que presentaban este inconveniente en la implementación.

Para realizar esto se optó por cambiar de compilador ya que los compiladores de Fortran 77 basados en `gcc`[42] no aceptan números de punto flotante cuadruples (llamados `long double` o `Real*16`). Por esto es que se migró todo el programa de manera de que pueda ser compilado mediante el Intel Fortran Compiler para Linux[43]. Esta migración acarreó los inconvenientes que cualquier traspaso de un lenguaje a otro puede acarrear, especialmente debido a que este compilador realiza chequeos mucho más estrictos que los realizados por el compilador usado anteriormente.

Luego de sortear los inconvenientes de migración surgió la necesidad de obtener y utilizar una librería de Message Passing que permita el envío y recepción de datos de tipo `Real*16`. Ninguna librería de las revisadas (`Intel`[44], `LAM`, `MPICH`) tenía implementado este tipo de datos para `Fortran`. Sin embargo, la librería `MPICH` tenía implementado el tipo `long double` para `C/C++`.

El inconveniente surgido es que en las máquinas IA-32 y AMD utilizadas ese tipo de datos es de 80 bits y no de 128 como en realidad es el tipo `REAL*16`. Esto quiere decir que el tipo de datos `long double` para `C/C++` que sí se encuentra definido en `MPICH` y el tipo `REAL*16` de `Fortran` no son iguales, y por lo tanto el primero no podía ser utilizado para reemplazar al segundo.

Finalmente, se optó por utilizar el tipo correspondiente a long double de `MPICH` corriéndolo en una máquina SMP Xeon Dual, *Papita*, la cual fue cedida en préstamo por Intel Argentina, en una AMD Opteron y en una Itanium ya que en las mismas este tipo de dato y el `Real*16` son equivalentes (128 bits). Se realizaron diversas pruebas para corroborar que luego de esta solución al problema de punto flotante la diferencia entre los resultados del programa serial y el paralelo no eran significativos.

3.4.2. Reordenamiento del Cálculo

Para resolver finalmente el problema de precisión en punto flotante se intentó realizar un reordenamiento de los cálculos para que el orden de las operaciones en la sumatoria con la cual se calcula la integral presente en la ecuación 1.4 sea similar al del programa original.

Para realizar esta solución se debieron hacer diversos análisis. En primera instancia se observaron los exponentes de los términos participantes en la sumatoria a realizar. Para realizar esto se modificó el programa para guardar en archivos cada uno de los vectores participantes en la sumatoria. Luego estos archivos fueron importados en una planilla de cálculo para poder observarlos en conjunto. A continuación se realizó una clasificación de los datos de acuerdo a los exponentes de los mismos, categorizándolos entre 1×10^{-50} y 1×10^{4}. Por otro lado fueron separados los ceros absolutos, ya que de otra forma contarían dentro de la categoría 1×10^{0} lo cual es incorrecto. Este análisis nos permitió conocer la distribución aproximada de los datos con los que se estaba trabajando.

Inicialmente, la idea planteada era reducir la pérdida de dígitos significativos en la operatoria, suponiendo que de esa manera el sistema se iba a estabilizar arrojando resultados similares a los generados por el programa original. De esta manera fue que se pensó en ordenar las sumatorias de forma tal que los términos se sumen de menor a mayor para minimizar la pérdida. Sin embargo, esta posibilidad fue desechada ya que al alterar el orden de la suma, como el programa era tan inestable desde el punto de vista numérico, los resultados obtenidos no eran los esperados.

Por estos motivos se implementó otra solución: analizar las pérdidas ocasionadas en cada operación del sistema serial de manera de intentar reproducir las mismas de la forma más precisa posible en el programa paralelo.

Con este objetivo en mente el primer análisis realizado fue hallar la diferencia absoluta entre los exponentes de los términos de cada suma realizada, tanto en el programa serial como en el paralelo. De esta forma fue realizada una planilla en donde si, por ejemplo, se sumaba un número con exponente 1×10^{-10} y otro con 1×10^{-5}, se anotaba una pérdida de cinco posiciones, ya que el resultado tendrá por lo menos exponente 1×10^{-5}. Uno de los aspectos a tener en cuenta nuevamente, fue la inclusión de ceros en la planilla. La diferencia entre un cero y cualquier otro número al momento de realizar la sumatoria es nula, porque no se pierden dígitos como sí ocurre entre dos números distintos de cero con diferentes exponentes. Los ceros poseen exponentes 1×10^{0} por lo cual debieron ser diferenciados de los valores distintos a cero con igual exponente para no cometer errores en los cálculos.

Luego de obtener estos datos se analizaron las distribuciones de estas pérdidas para poder observar de forma gráfica cuál era el problema que se planteaba.

Suponiendo, finalmente, que los dígitos significativos en nuestros cálculos fueran siete, se realizó un análisis de cuáles eran las operaciones en las cuales se

perdían menos de 7 dígitos y en cuáles más, y finalmente se graficó las dispersión de estos datos.

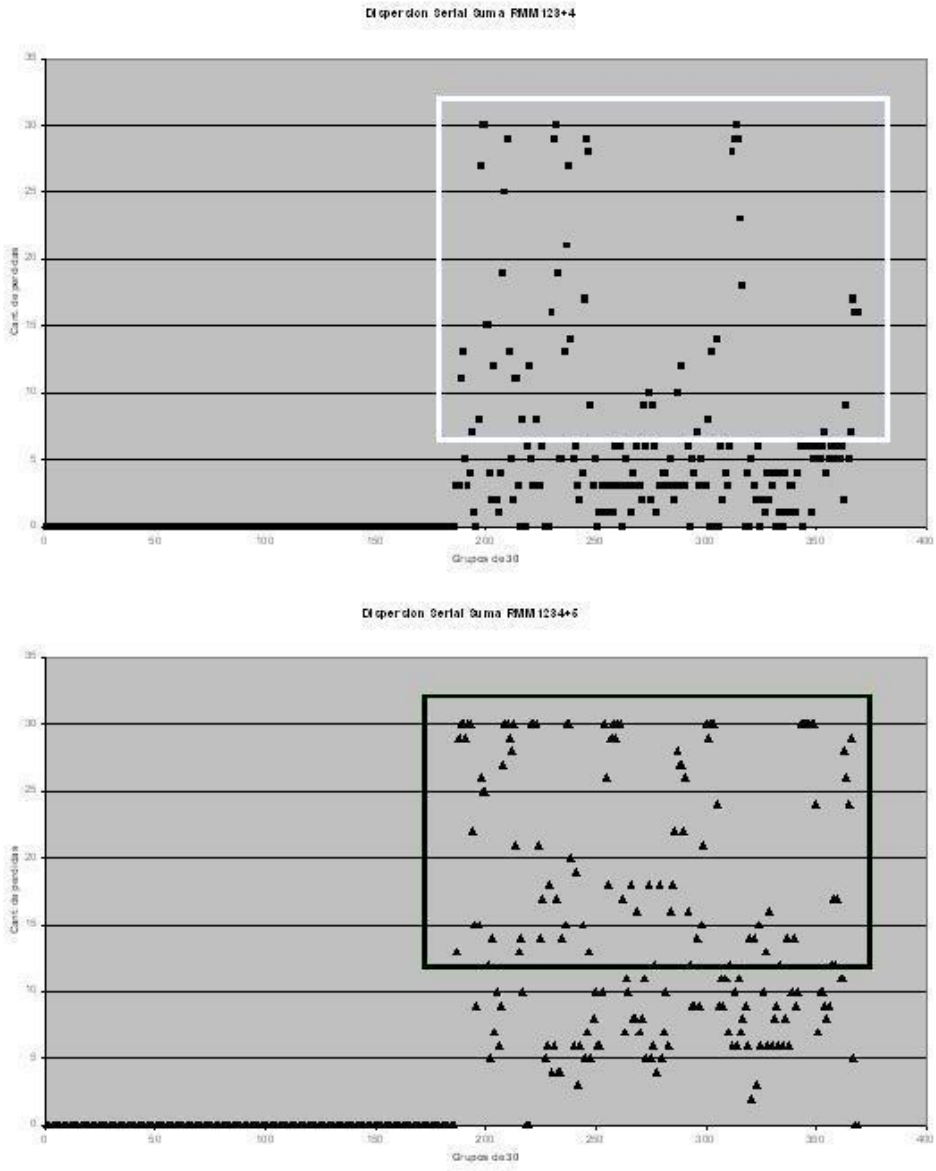

Figura 3.6: Dispersión de pérdida de dígitos significativos en la cuarta y quinta suma del algoritmo serial.

No fueron graficados los resultados tanto del proceso 0 como las primeras tres sumatorias del sistema serial debido a que estos datos eran idénticos (y así se suponía que debía ser). Los datos de las figuras 3.6 y 3.7 correponden a las sumatorias del programa serial (RMM123+4, RMM1234+5, RMM12345+6 y RMM123456+7 respectivamente) y los de las figuras 3.8 y 3.9 al paralelo(4+5, 45+6, 456+7 y Proceso0+4567 respectivamente), donde cada número indica los valores del átomo correspondiente (1 es átomo 1, y así sucesivamente).

Puede observarse que hay una cierta cantidad de datos en el programa paralelo que no son perdidos y en el serial sí. Esto se debe principalmente a que en el proceso 1 la diferencia con los resultados parciales no son tan grandes con respecto a

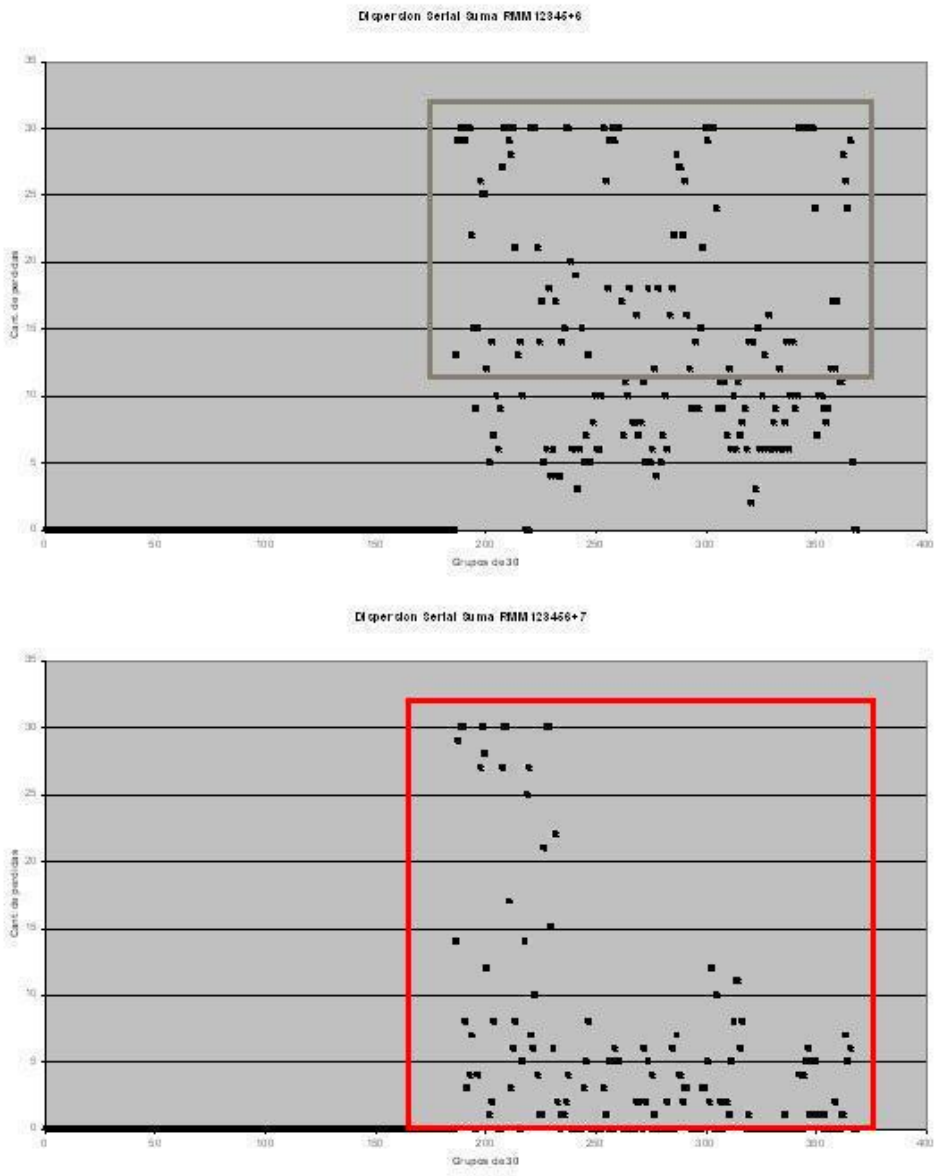

Figura 3.7: Dispersión de pérdida de dígitos significativos en la sexta y septima suma del algoritmo serial.

los nuevos valores que se van sumando como para que se produzcan tales pérdidas. Observando los datos se vió que la mayor diferencia se daba generalmente contra el valor inicial de RMM, que en muchas ocasiones poseía un valor mayor al de los átomos que se suman posteriormente.

Finalmente de la observación de los datos se llegó a una solución alternativa al problema. La misma consistió en agregar en cada uno de los procesos un valor determinado de forma tal que en cada sumatoria se ocasione una pérdida similar a la ocurrida en el programa serial, y luego finalmente restarle ese valor al total. Por ejemplo, si en el programa serial se realizaba una suma entre el vector RMM original más los átomos del 1 al 7, en el paralelo se suma por un lado el vector RMM más los átomos 1,2 y 3 y en el otro, aunque deberían sumarse sólo los átomos 4 a 7, se le agrega un valor inicial que luego es finalmente restado antes de unir

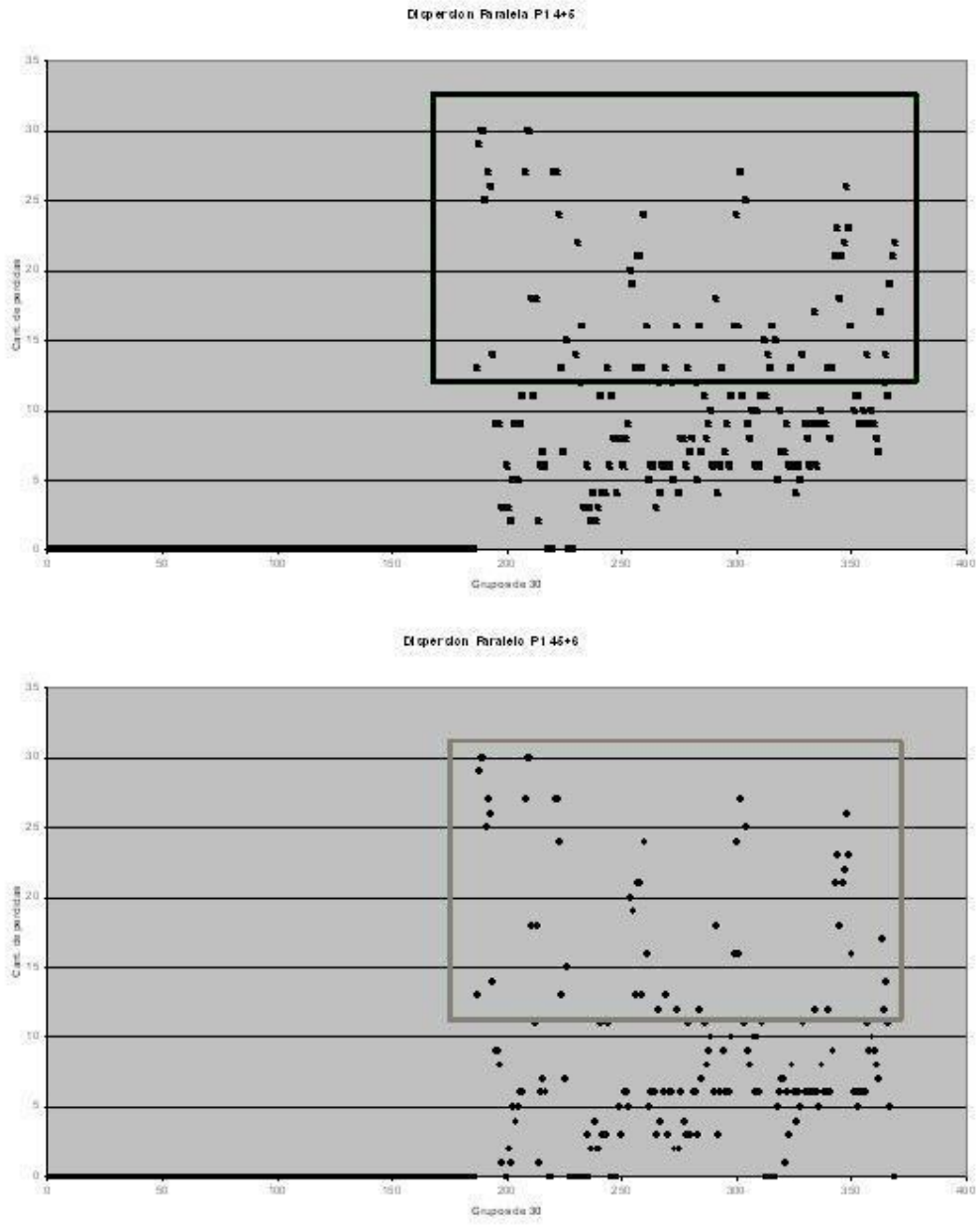

Figura 3.8: Dispersión de pérdida de dígitos significativos en las dos primeras sumas del proceso 1 del algoritmo paralelo.

ambos procesos.

Al analizar cómo eran los valores utilizados en las operaciones se observó que la mayor parte de las pérdidas se daban debido a la diferencia de magnitud entre los valores del vector original RMM y el resto de los átomos como se mencionó con anterioridad. Esto quiere decir que en una gran cantidad de casos en el programa serial al momento de sumar el cuarto átomo el orden de magnitud presentado por la sumatoria de RMM más el resto de los átomos correspondientes al proceso era igual al del RMM original. Por este motivo se optó por tomar el RMM original como número a sumar y restar en todos los procesos para producir la pérdida de información necesaria. Si bien los resultados obtenidos no fueron iguales a los del

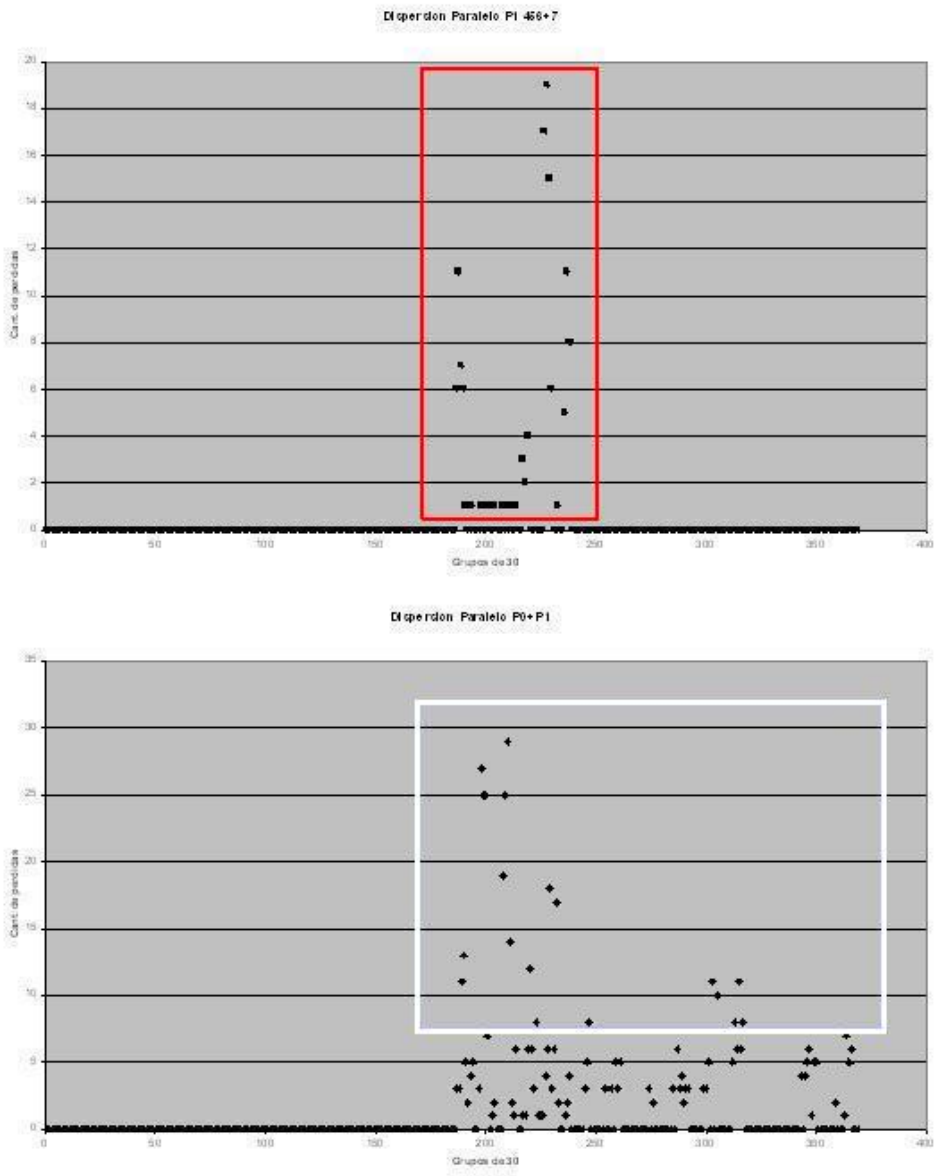

Figura 3.9: Dispersión de pérdida de dígitos significativos en la tercera suma del proceso 1 del algoritmo paralelo y la posterior unión de los dos procesos.

serial, los mismos eran muy similares, y a los efectos de la investigación química para la cual se utilizarían, eran válidos

Esta nueva solución puede observarse en el esquema de la figura 3.10

Finalmente se realizaron diversas corridas para corroborar el buen funcionamiento de esta implementación y se obtuvieron los resultados que se verán en la sección 4.

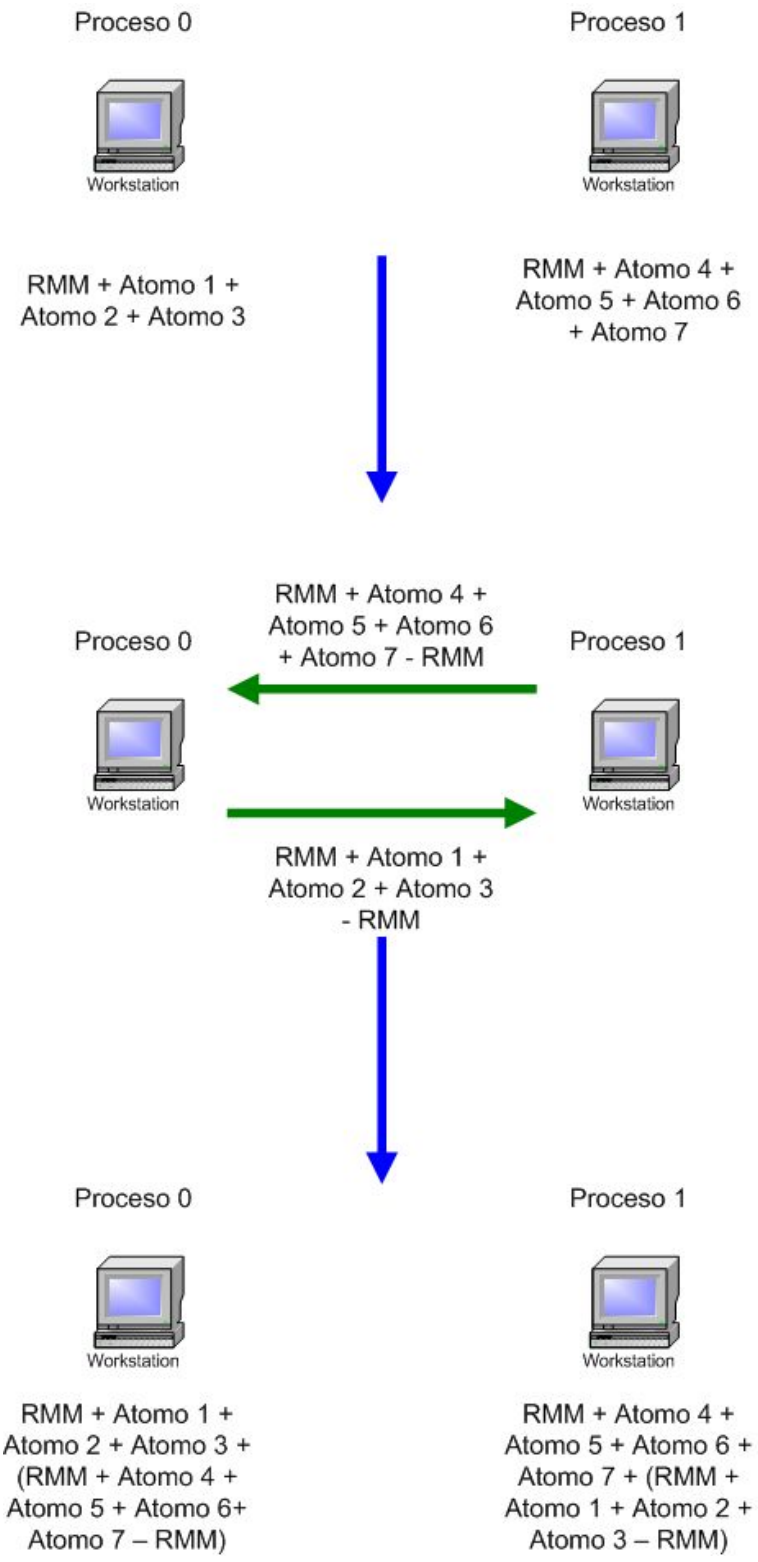

Figura 3.10: Forma en que es paralelizado el dominio de 7 átomos en dos procesos con reordenamiento de cálculo.

Capítulo 4

Resultados y Discusión

4.1. Entradas utilizadas

Se utilizaron dos tipos de entradas para realizar las pruebas de las distintas implementaciones realizadas. Una entrada se corresponde a un peroxinitrito ($OONO^-$) más un dióxido de carbono CO_2 cuánticos más 498 moléculas de agua. La segunda entrada utilizada se corresponde sólo al peroxinitrito con las 498 moléculas de agua.

El peroxinitrito ($OONO^-$) es una especie estable formada en ambientes biológicos por la reacción del anión radical superóxido (O_2^-) con el NO [45]. Es muy estable en soluciones alcalinas [20] pero no en soluciones neutras o ácidas, ya que su ácido conjugado (peroxinitroso) isomeriza a ácido nítrico. El peroxinitrito exhibe una gran toxicidad por su capacidad de formar radicales libres que oxidan grupos sulfhidrilos, fosfolípidos y ADN.

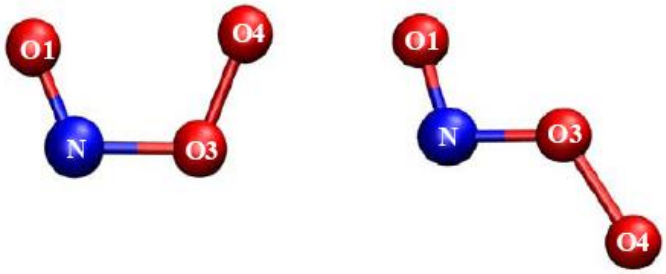

Figura 4.1: Representación esquemática de los confórmeros cis y trans del peroxinitrito.

Con la primera entrada utilizada se intenta determinar el mecanismo de reacción del peroxinitrito con dioxido de carbono. Existen dos principales razones por las cuales resulta interesante esta investigación.

- Ésta es la principal reacción del $OONO^-$ en medios fisiológicos (por la alta concentración del CO_2 y porque la misma se produce rapidamente)
- En vacío esta reacción no tiene barrera de energía potencial mientras que en solución si la tiene (hay resultados experimentales que así lo sugieren). Por

este motivo la barrera está determinada por el solvente y es ideal para este programa que simula el efecto del mismo.

Un esquema de la misma se puede ver en la figura 4.2.

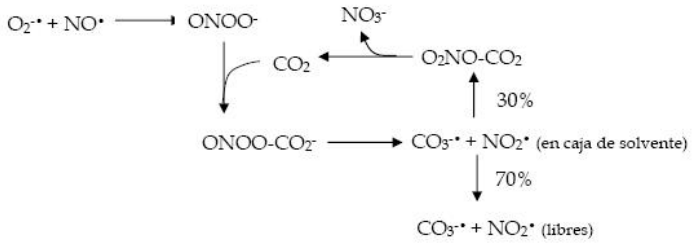

Figura 4.2: Esquema de la reacción del peroxinitrito con dióxido de carbono.

En una primera etapa, como puede observarse en la figura 4.2, el peroxinitrito se adiciona al CO_2 formando un aducto (el nitrosoperoxicarbonato). Este aducto se descompone en los radicales carbonato y NO_2 encerrados en una caja de solvente. A su vez, estos radicales pueden reacomodarse de manera tal que conforman otro aducto llamado nitrocarbonato, o bien pueden liberarse como tales.

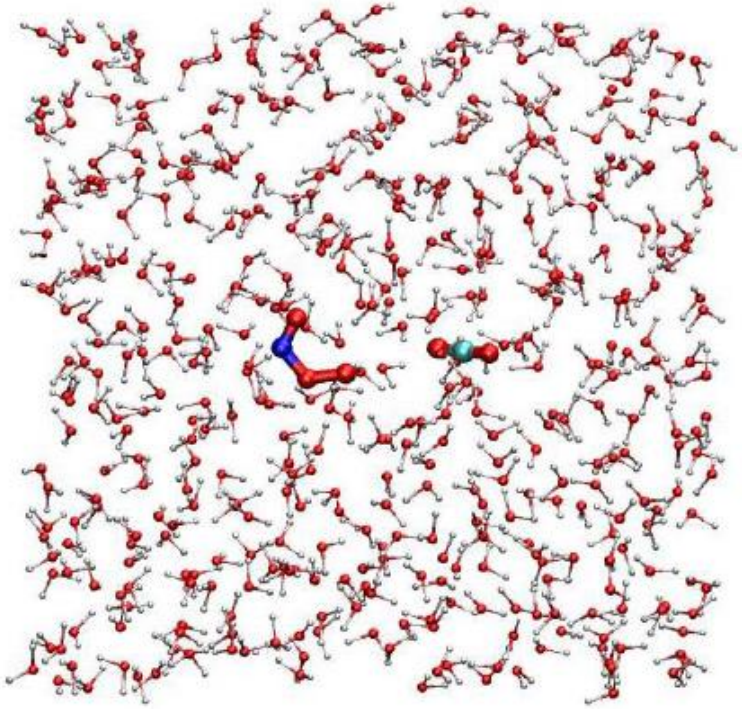

Figura 4.3: El peroxinitrito, el dióxido de carbono y las aguas en la caja de simulación.

Finalmente la investigación correspondiente a la segunda entrada se realiza a efecto de estudiar como influye un solvente (agua) sobre el espectro vibracional del peroxinitrito. Pueden observarse dimeros utilizados para observar la correctitud de los cálculos de las fuerzas y la energía en la figura 4.4.

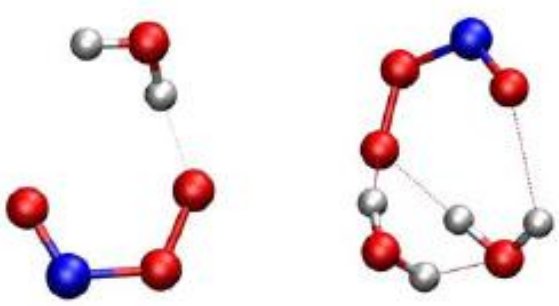

Figura 4.4: Representación esquemática de complejos de peroxinitrito con una y dos moléculas de agua.

	100 pasos	**500 pasos**	**1000 pasos**
`Serial`	24:10	128:25	251:52
`Paralelo`	22:35	107:00	215:55

Cuadro 4.1: Tiempos de ejecución total del programa en una máquina SMP Xeon Dual con 7 átomos con 128 bits de precisión. Los resultados están expresados en HH:MM.

4.2. Implementación de precisión extendida

Una vez implementado el algoritmo con precisión extendida se realizaron corridas en una máquina SMP Xeon Dual, obteniéndose los resultados de la tabla 4.1.

La misma puede observarse de mejor manera en la figura 4.5. Se utilizó la entrada de 7 átomos variando la cantidad de pasos a realizarse.

A pesar de que puede observarse cómo una vez realizada la transformación de datos de tipo `REAL*8` a `REAL*16` el programa paralelo logra una mejora de casi un $15\,\%$ con respecto al serial esta solución no es válida. Esto es así debido a que el programa serial modificado tarda un orden de magnitud más que el original. La causa de este comportamiento es que no existe aún una máquina en el mercado que trabaje en forma nativa con números de 128 bits. Tanto la SMP Xeon Dual como las Itanium de Intel, y las Opteron de AMD realizan una emulación, lo cual quiere decir que trabajan con números de esa longitud pero utilizando software y no con hardware como sí ocurre con los números de 64 bits. Es debido a esta emulación por software que la velocidad de procesamiento se ve fuertemente afectada.

Por estos motivos, a pesar de que el programa paralelo funcione mejor que el serial modificado (lo cual podría indicar que a futuro cuando haya computadoras de 128 bits nativos, la solución servirá), no se logró el objetivo de este trabajo, por lo cual se optó por proceder con la implementación de la segunda opción. A continuación se muestran gráficos de eficiencia, overhead y speedup de este método en las figuras 4.6, 4.7, 4.8 y 4.9 respectivamente.

No fueron realizadas corridas con otro tipo de entradas o en otras máquinas con esta versión debido a la limitación de performance comentada con anterioridad.

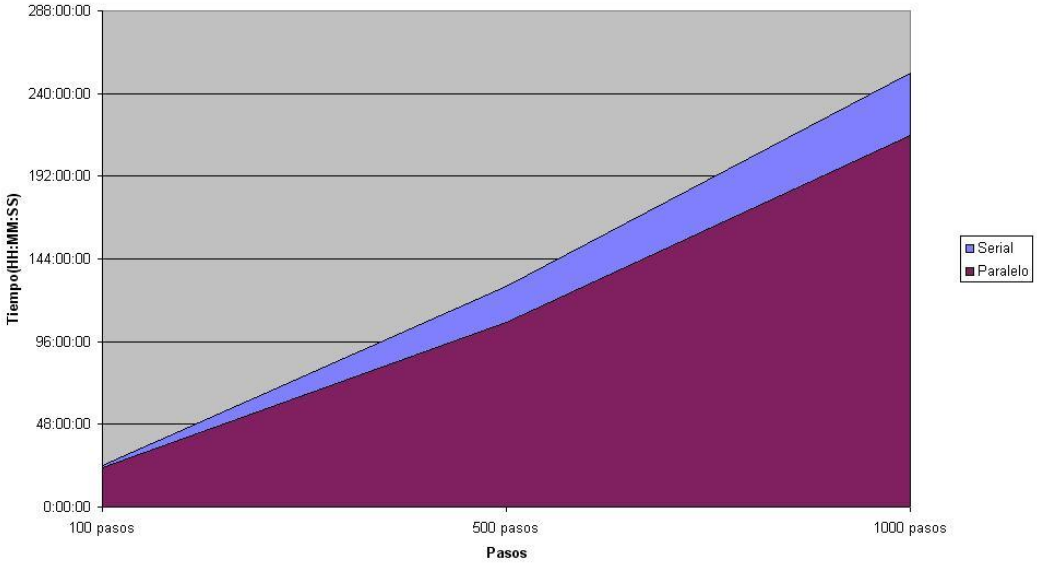

Figura 4.5: Tiempos de ejecución total del programa en una máquina SMP Xeon Dual con 7 átomos. Los resultados están expresados en HH:MM.

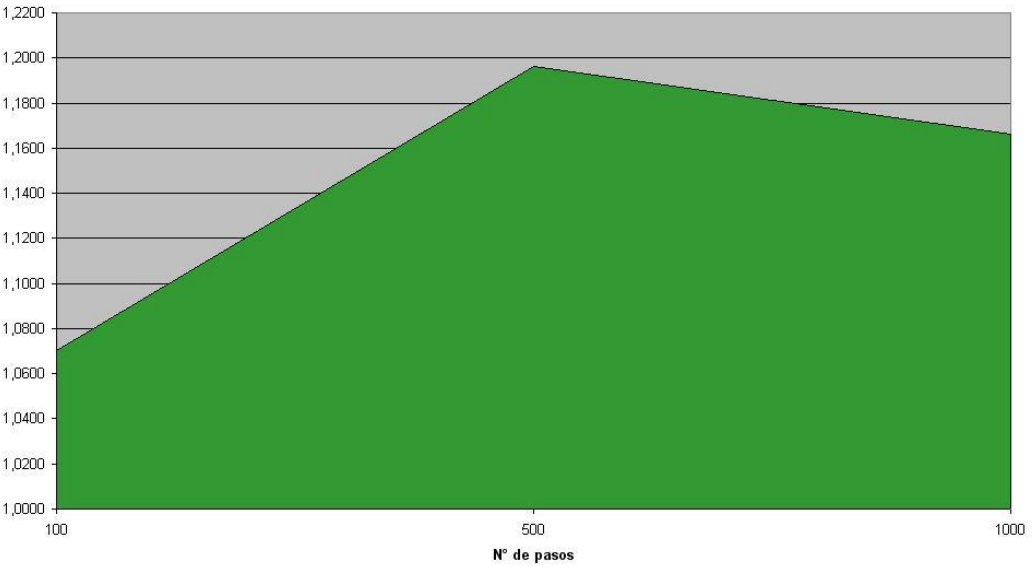

Figura 4.6: Speedup del programa paralelo con precisión extendida a 128 bits corrido en 2 procesos.

4.3. Implementación con reordenamiento de cálculo

En segunda instancia se implementó la solución de reordenamiento de cálculo mencionada en la sección 3.4.2. Para esta implementación inicialmente se realizaron diversas corridas en la máquina SMP Xeon Dual para corroborar que los resultados obtenidos se correspondan a los del programa serial. Estas corridas ini-

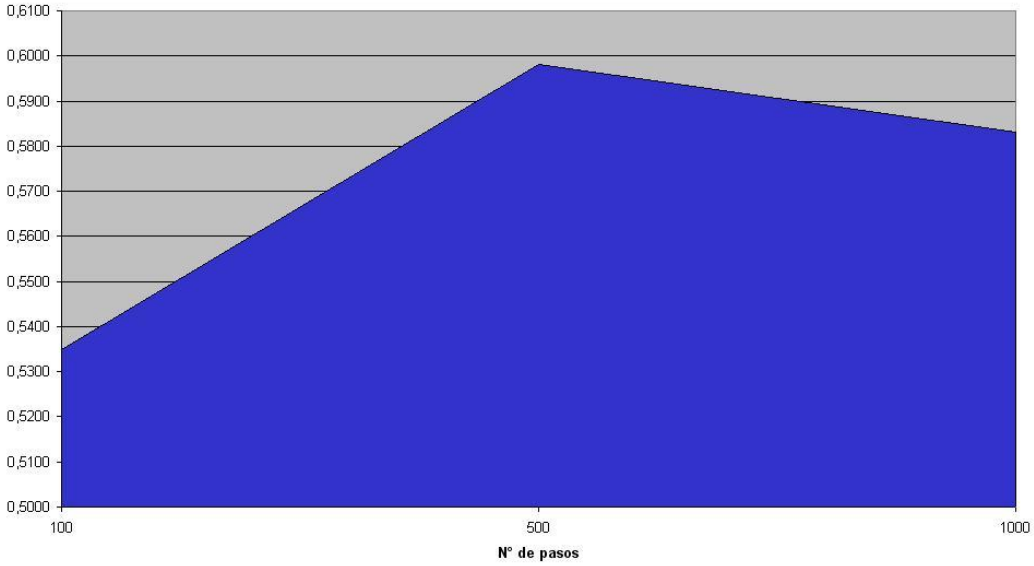

Figura 4.7: Eficiencia del programa paralelo con precisión extendida a 128 bits corrido en 2 procesos.

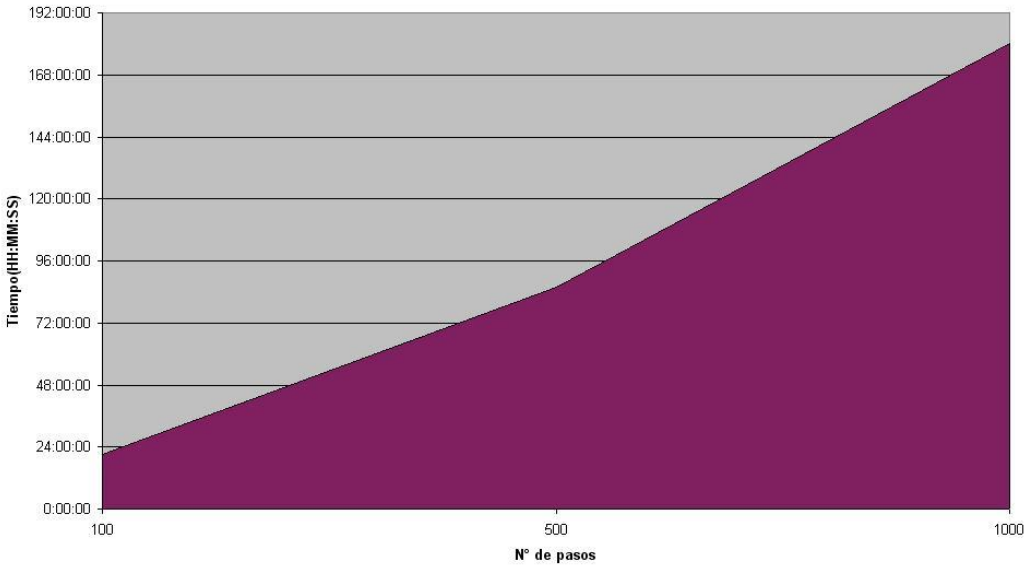

Figura 4.8: Overhead del programa paralelo con precisión extendida a 128 bits corrido en 2 procesos.

ciales fueron realizadas con una entrada de 4 átomos y variando la cantidad de pasos de ejecución. Los tiempos de ejecución de estas corridas y otros valores de performance pueden observarse en las tablas 4.2 y 4.3.

Finalmente, una vez verificados los resultados se procedió a realizar corridas más extensas con un problema más grande que constaba de un sistema con 7 áto-

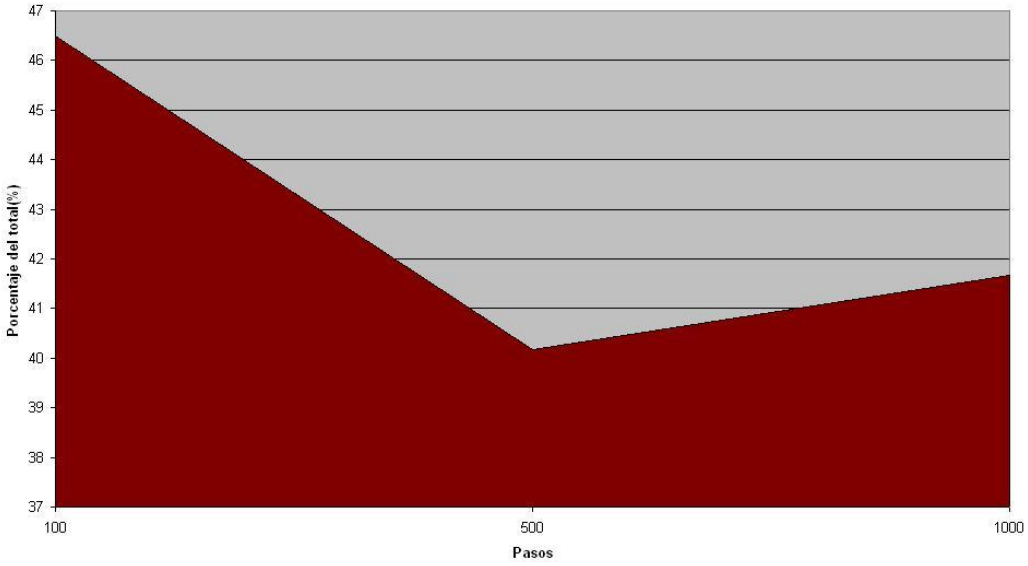

Figura 4.9: Overhead en porcentaje sobre el total de tiempo consumido por el programa paralelo con precisión extendida a 128 bits corrido en 2 procesos.

	100 pasos	**200 pasos**	**300 pasos**	**400 pasos**
`Serial`	14:30	29:15	43:10	58:20
`Paralelo`	11:30	22:45	34:30	45:40

Cuadro 4.2: Tiempos de ejecución total del programa en una máquina SMP Xeon Dual con 4 átomos con reordenamiento de cálculo en el programa. Los resultados están expresados en MM:SS.

mos. Inicialmente estas corridas fueron realizadas en la máquina Xeon Dual. Ver la tabla 4.4 y la figura 4.10.

Como puede observarse, esta implementación es mucho más rápida que la de 128 bits debido a que no se utiliza emulación por software para realizar las operaciones. También puede observarse como la brecha de tiempos entre el programa serial y el paralelo también se va abriendo conforme se agranda el dominio del problema. En las figuras 4.11, 4.12, 4.13 y 4.14 se muestran diversas variables de performance asociadas a estas corridas.

Puede observarse en los gráficos que cuando se realizaron las mediciones el overhead varió aproximadamente un 2 % cuando se hicieron 10000 pasos con res-

	100 pasos	**200 pasos**	**300 pasos**	**400 pasos**
`Speedup`	1.2608	1.2857	1.2512	1.2773
`Eficiencia`	0.6304	0.6428	0.6256	0.6386
`Overhead`	36.95 %	35.71 %	37.43 %	36.13 %

Cuadro 4.3: Speedup, Eficiencia y overhead del programa en una máquina SMP Xeon Dual con 4 átomos con reordenamiento de cálculo en el programa. El overhead se mide con respecto al tiempo total de ejecución.

	5000 pasos	10000 pasos	15000 pasos	20000 pasos
Serial	37:33	72:12	110:25	145:10
Paralelo	31:55	63:24	93:40	122:27

Cuadro 4.4: Tiempos de ejecución total del programa en una máquina SMP Xeon Dual con 7 átomos con reordenamiento de cálculo en el programa. Los resultados están expresados en HH:MM.

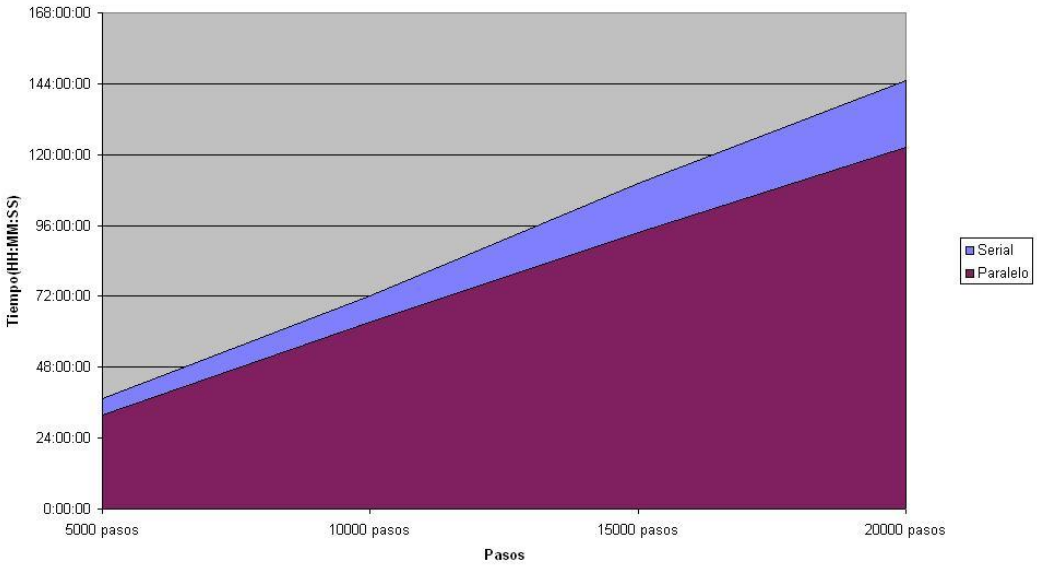

Figura 4.10: Tiempos de ejecución total del programa con reordenamiento de cálculo en una máquina SMP Xeon Dual con 7 átomos. Los resultados están expresados en HH:MM:SS.

pecto a las otras corridas. A pesar de que el gráfico produce la sensación de que esta variación es muy grande no lo es realmente. Esta situación puede deberse al hecho de que la máquina en la cual fueron realizadas las corridas también es utilizada esporádicamente para realizar otros trabajos. Si bien se intentó que la máquina no fuera utilizada por otras personas es posible que no haya sido este el caso y se haya corrido algún proceso corto que haya modificado los tiempos de ejecución del programa. Aún así, el speedup y la eficiencia medidas no son despreciables y demuestran una ganancia en performance interesante a los fines de la investigación futura de sistemas de partículas mayores. Por otro lado cabe aclarar que la utilización por parte de otros usuarios de la SMP Xeon Dual no afecta el programa serial dado que el mismo es corrido en un solo procesador, mientras que los procesos de usuarios pueden correrse en el otro. En cambio, sí afecta al programa paralelo dado que el mismo utiliza ambos procesadores, y si uno se retrasa debido al uso, el otro también lo hará debido a la mensajería bloqueante y a la interacción de ambos procesos.

Luego fueron realizadas mediciones en el cluster *FIUPG*, sito en el estado de Florida, Estados Unidos. El mismo está compuesto por 17 nodos Intel SMP Xeon duales. Las corridas fueron realizadas con dos y cuatro procesos en nodos distintos, es decir a razón de un proceso en cada nodo, obteniéndose los resultados

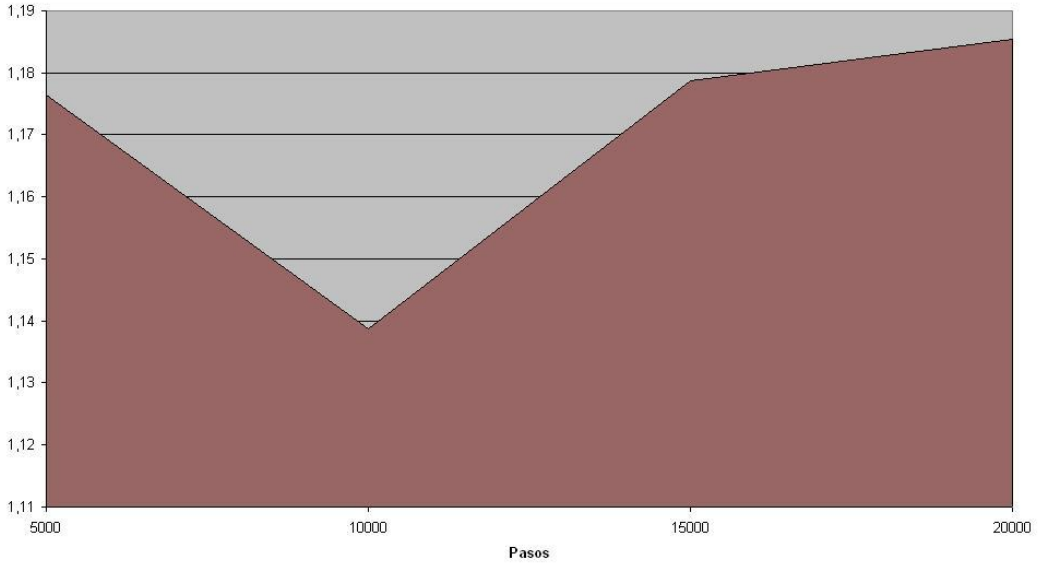

Figura 4.11: Speedup del programa con reordenamiento de cálculo en 2 procesos en una máquina SMP Xeon Dual (Papita) con una entrada de $OONO^- + CO_2$.

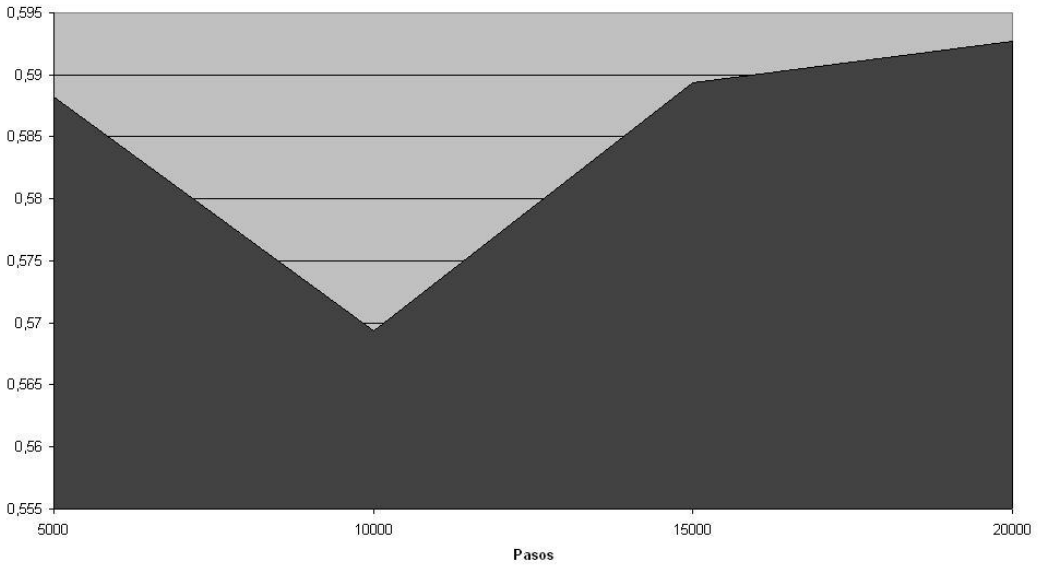

Figura 4.12: Eficiencia del programa con reordenamiento de cálculo en 2 procesos en una máquina SMP Xeon Dual (Papita) con una entrada de $OONO^- + CO_2$.

que se pueden ver en la tabla 4.5 y en la figura 4.15. No fueron utilizados más procesadores del cluster debido a que con el tama ño de las entradas con las que se realizaron las pruebas no permite exigir ejecuciones con un número de procesos mayor. Por otra parte se eligió utilizar nodos distintos para poder medir de mejor manera el overhead introducido por los buses externos de datos de baja velocidad.

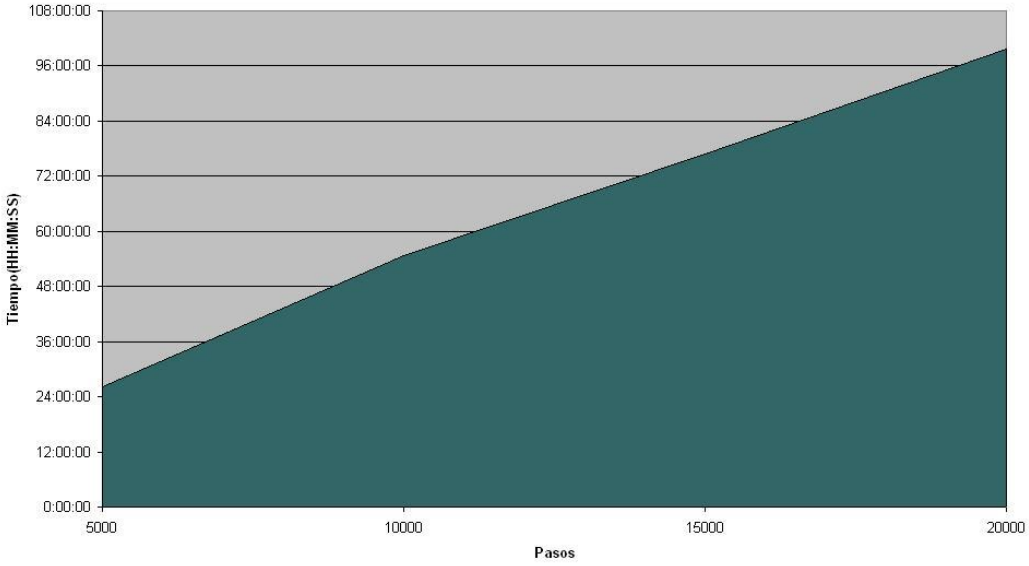

Figura 4.13: Overhead del programa con reordenamiento de cálculo en 2 procesos en una máquina SMP Xeon Dual (Papita) con una entrada de $OONO^- + CO_2$.

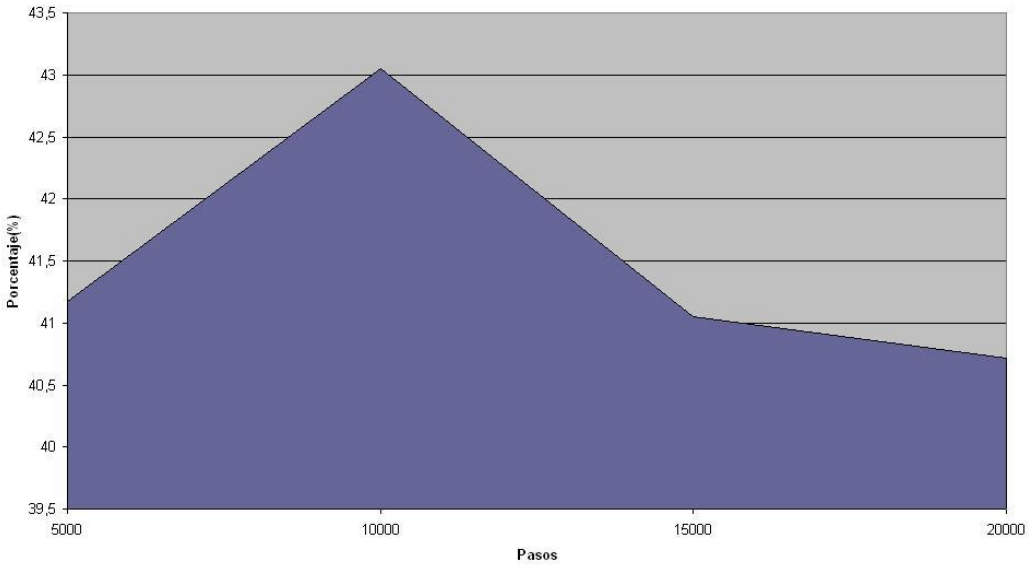

Figura 4.14: Overhead del programa con reordenamiento de cálculo en 2 procesos en una máquina SMP Xeon Dual (Papita) con una entrada de $OONO^- + CO_2$.

Una vez obtenidos los datos de las ejecuciones del sistema en distintas configuraciones y entornos se pudieron obtener datos sobre *speedup*, *overhead* y *eficiencia*, observables en las figuras 4.16, 4.17, 4.18 y 4.19 respectivamente.

Puede observarse que conforme el dominio del problema crece la brecha entre los tiempos del sistema serial y los del paralelo se vuelven mayores. Quedará a

	5000 pasos	10000 pasos	15000 pasos	20000 pasos
Serial	73:10	145:30	218:20	292:00
2 Procesos	48:50	97:42	146:40	195:30
4 Procesos	45:03	88:10	134:00	175:50

Cuadro 4.5: Tiempos de ejecución total del programa serial, y el paralelo en 2 y 4 procesos con una entrada de $OONO^- + CO_2$.

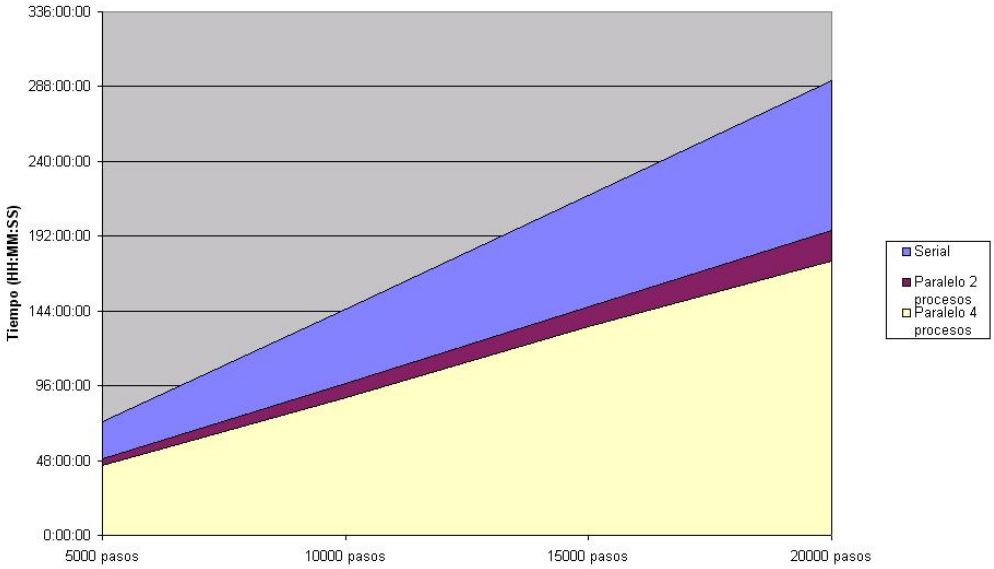

Figura 4.15: Tiempos de ejecución total del programa serial, y el paralelo en 2 y 4 procesos con una entrada de $OONO^- + CO_2$.

futuro la ejecución de un sistema de mayores dimensiones para completar el estudio sobre performance del sistema paralelizado para observar cuál es el límite de procesamiento del nuevo programa con fines prácticos. A simple vista puede observarse que para mayores dominios del problema los tiempos ahorrados por el sistema paralelo crecen significativamente con respecto al serial.

En contraparte a los resultados obtenidos puede observarse el gran overhead introducido en la versión paralela del programa debido al pasaje de mensajes, lo cual hace que el *Speedup* no sea lineal con respecto a la cantidad de procesos utilizados. Esto quiere decir que si el *Speedup* fuera lineal, utilizando dos procesos el cálculo debería tardar la mitad de tiempo. Esta situación no se esta cumpliendo debido a que, como muestran los gráficos, el overhead varía entre cerca del 25 % y el 60 % del tiempo total, lo que deja a futuro la tarea de intentar reducir el mismo. El overhead se produce no solo debido a la comunicación entre procesos sino también al agregado de operaciones utilizadas para particionar el dominio del problema y posteriormente volver a unirlo. Sin embargo los costos de comunicación son mucho mayores a los de procesamiento, por lo cual generalmente se desprecian los últimos. En este caso en particular se ha podido observar que al tratar con grandes cantidades de datos, las operaciones agregadas producían un aumento significativo en el tiempo total de ejecución del programa paralelo y es por eso que se debió analizar como reducir las mismas al mínimo posible. Por otra parte los costos

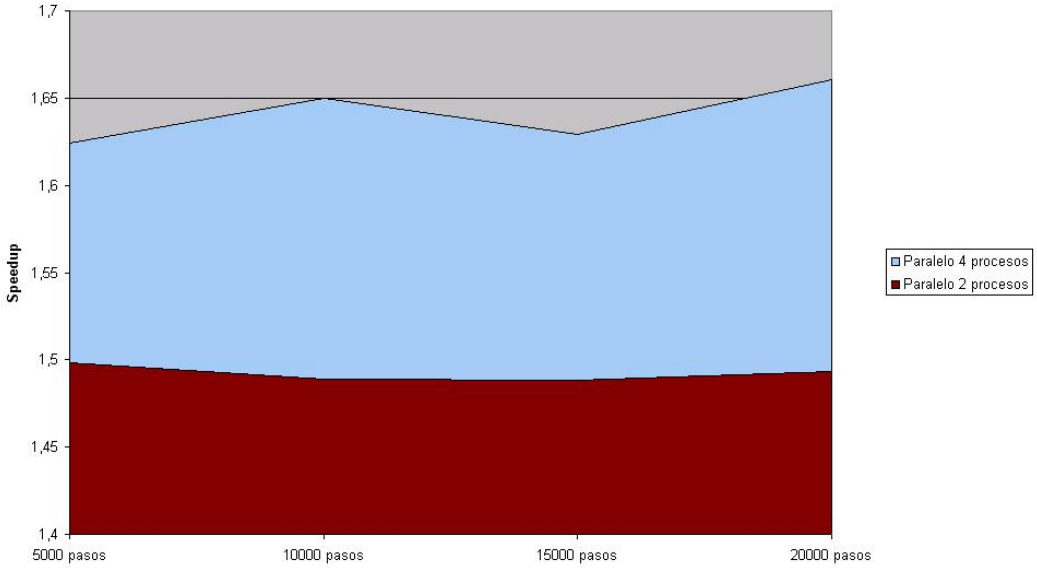

Figura 4.16: Speedup del programa con reordenamiento de cálculo en cluster de Intel SMP Xeon Dual.

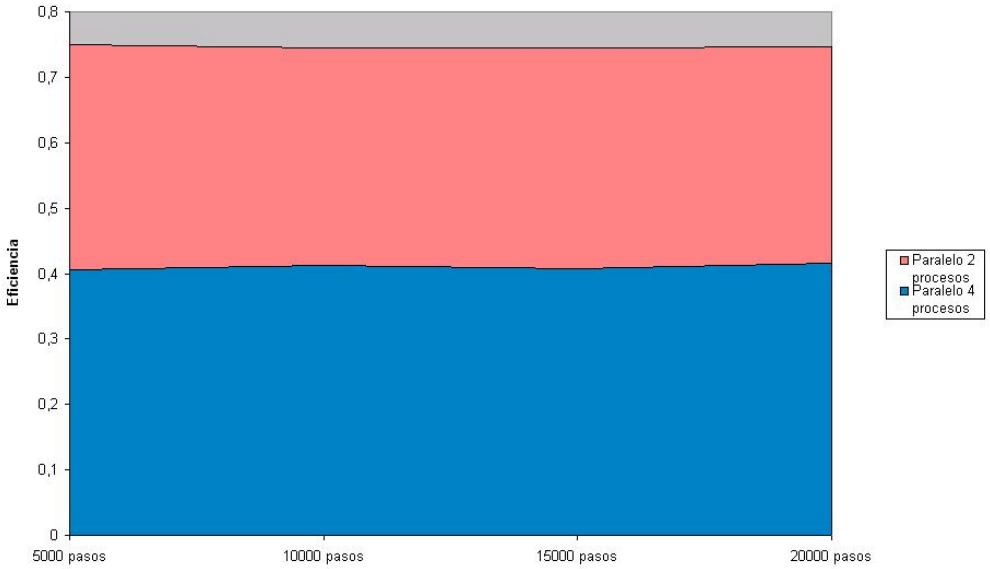

Figura 4.17: Eficiencia del programa con reordenamiento de cálculo en cluster de Intel SMP Xeon Dual.

de comunicación en un cluster como *FIUPG* son mayores a los presentes en una SMP Xeon Dual debido a que el primero posee buses externos de comunicación que son más lentos que los buses internos utilizados en la segunda.

Finalmente, para demostrar la utilidad final del programa fueron realizadas corridas totalmente nuevas en *FIUPG* y en la SMP Xeon Dual. Los parámetros para estas corridas fueron preparados por el grupo de química inorgánica que proveyó el

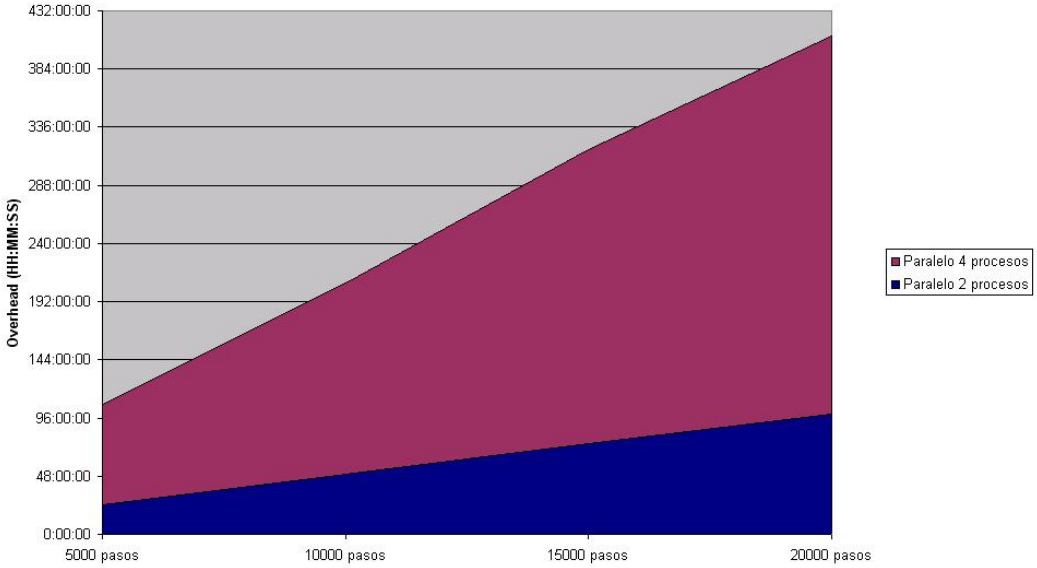

Figura 4.18: Overhead del programa con reordenamiento de cálculo en cluster de Intel SMP Xeon Dual.

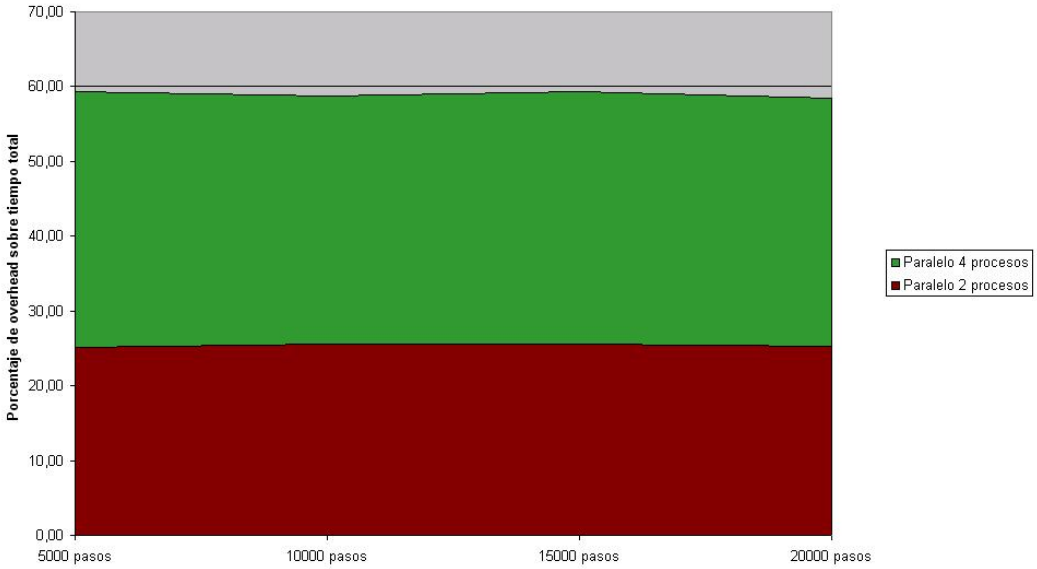

Figura 4.19: Overhead del programa con reordenamiento de cálculo en cluster de Intel SMP Xeon Dual.

programa serial y fueron supervisadas por los mismos para verificar la correctitud de los datos obtenidos. Mediante este tipo de corridas se contribuye a obtener información sobre la interacción molecular y obtener gráficos como los de las figuras 4.20 y 4.21.

Cada una de las formas en que vibra una molécula es llamada *modo normal*. En éstas figuras pueden observarse las frecuencias correspondientes a cada modo

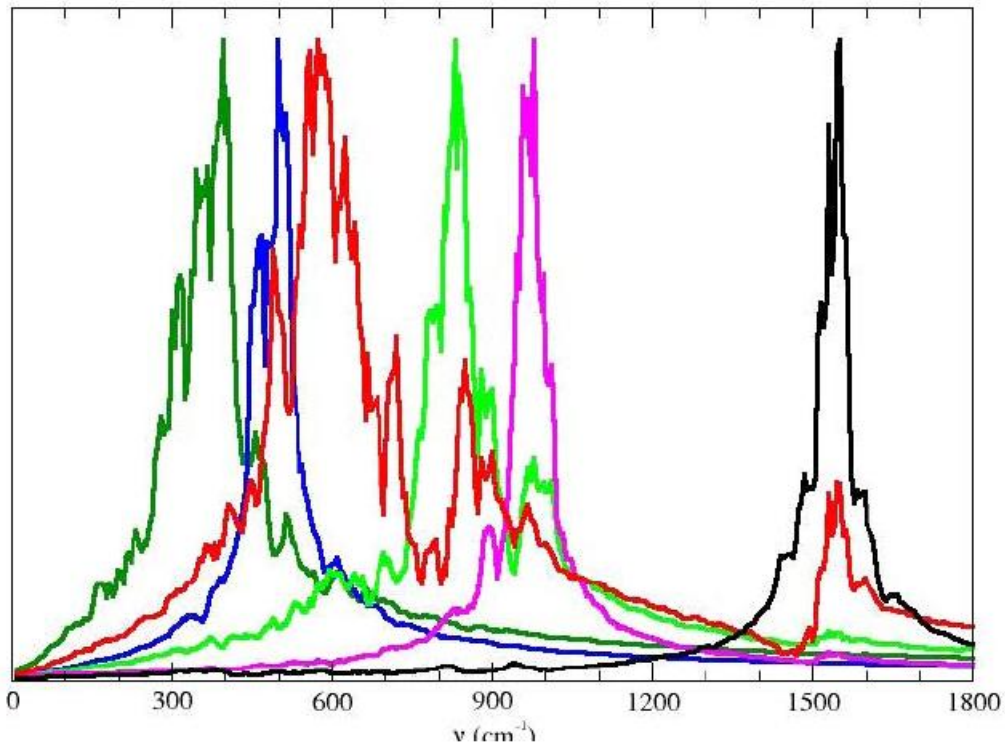

Figura 4.20: Espectro vibracional obtenido a partir de la simulación PBE-TIP4P, est. NO1 en negro, flex. O1NO3, est. O3-O4 en verde, est. NO3 en rojo, OONO tors., NO3O4 flex. en verde oscuro.

normal de la molécula de peroxinitrito en solvente. En ambas figuras se aprecia un marcado ensanchamiento en la banda correspondiente al estiramiento $N - O_3$, indicando un fuerte efecto solvente para este modo en particular. Estos resultados obtenidos coinciden con valores experimentales obtenidos[22] y mediante este trabajo se pudo reinterpretar los mismos. Los errores generales son pequeños así como también los errores con respecto a la sensible banda de estiramiento $N - O_3$. El ensanchamiento y el solvente explícito no pueden tratarse con los métodos tradicionales de simulación de estos espectros.

Este estudio sirvió para reinterpretar el espectro experimental originalmente mal asignado, así como para dar una explicación microscópica del mencionado espectro y en particular del efecto del solvente en el mismo.

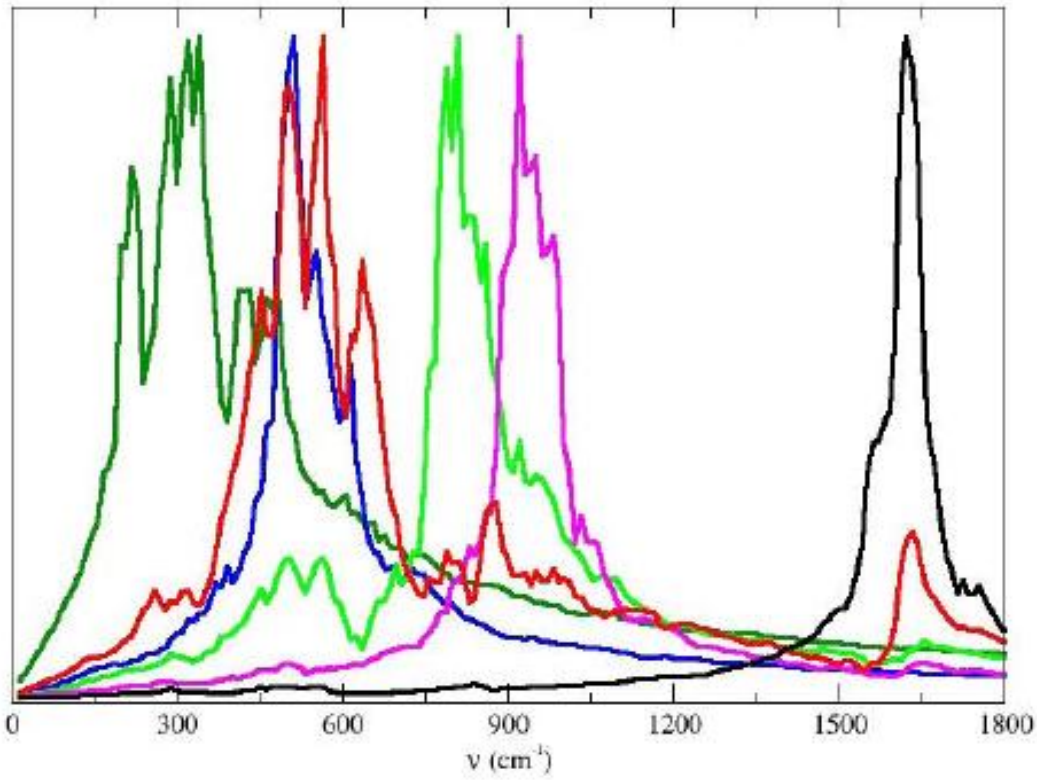

Figura 4.21: Espectro vibracional obtenido a partir de la simulación PBETIP4P-FQ, est. NO1 en negro, flex. O1NO3, est. O3-O4 en verde, est. NO3 en rojo, OONO tors., NO3O4 flex. en verde oscuro.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

De lo estudiado en el presente trabajo se desprenden diversas observaciones.

El programa serial es inestable numéricamente. Uno de los mayores problemas para realizar una solución paralela radicó en la inestabilidad numérica del programa original. Esto quedó demostrado al realizar diversos análisis en los cuales se alteraban los órdenes de distintas operaciones y los resultados cambiaban radicalmente, al punto de obtener un resultado totalmente distinto, erróneo según los parámetros utilizados o directamente ninguno (error de convergencia). A pesar de que los autores originales del programa tenían la noción de existencia de estos problemas nunca se lo había demostrado de forma tan contundente, lo cual fue un avance para la toma de conciencia y un planteo sobre la forma de trabajo y el rumbo a seguir.

Luego de consultar la problemática con distintas personas y profesionales reconocidos en el área de métodos numéricos se plantearon dos soluciones posibles y ambas fueron implementadas.

La primera versión realizada alcanzó los objetivos en cuanto a correctitud de los resultados y estabilidad numérica, sin embargo al trabajar con números de punto flotante de precisión cuádruple la performance del programa se vió afectada de forma notable. Esto se debió a que no existen en la actualidad máquinas que trabajen con esa precisión en forma nativa sino que lo hacen por medio de emulación por software. Esta primera solución será utilizable a futuro cuando se comercialicen máquinas con unidad de punto flotante que posean registros de 128 bits. En ese momento esta versión será probablemente mejor en cuanto a performance y precisión que la que finalmente fue utilizada en este trabajo. La versión de 128 bits, como ya se explicó, tiene una mayor precisión de variables, pero no solo eso, posee menos operaciones que la segunda solución implementada debido a que esta última utilizó un reordenamiento de cálculo que necesitó de una cierta cantidad de instrucciones con su consiguiente costo computacional.

Con la implementación que finalmente se utilizó en este estudio se logró una aproximación al programa serial que aunque no otorga resultados exactamente iguales, aunque fueron tomados como válidos por los expertos del área.

En esta implementación se optó por ocasionar una pérdida de precisión para poder estudiar sistemas con un programa paralelo. Este programa abre un abanico de posibilidades para poder resolver sistemas intratables hasta el momento, he

aquí su mejor característica. Adicionalmente, la performance del programa resulta razonable y el overhead producido se ve compensado por la ventaja de poder tratar sistemas cada vez más grandes.

Como pudo observarse, el algoritmo paralelo se vuelve más eficiente conforme el dominio del problema crece. Esto se debe a que el overhead producido por la comunicación entre procesos se ve compensado por la cantidad de instrucciones ejecutadas en cada uno de los nodos.

Para diversas corridas nuevas se utilizó una SMP Xeon Dual debido a que la ganancia de la misma era de un $15\,\%$ aproximadamente en el caso de Papita, y alrededor de un $50\,\%$ (esto debido a que el cluster *FIUPG* fue utilizado en exclusividad y posee máquinas mas potentes, y Papita era utilizada aleatoriamente por otros usuarios) en el caso de cada nodo del cluster *FIUPG*, lo cual reducía los tiempos de cada corrida de forma notoria. Se optó por este tipo de máquina porque los costos de comunicación mediante buses externos afectan la performance. Sin embargo esto no quiere decir que a futuro no se logre una mejor performance cuando la comunicación entre nodos de un cluster mejore y convenga más la utilización de varios nodos en vez de solo utilizar una máquina tipo SMP.

5.2. Trabajo futuro

Como trabajo a futuro se testearán dominios mayores y la ejecución de los mismos en un cluster de mayor magnitud, sin embargo esto no puede ser realizado de momento debido a falta de tiempo para realizar ejecuciones de gran envergadura.

Una estrategia analizada inicialmente, fue la utilización de mensajes no bloqueantes ya que la metodología actual (pasaje de mensajes bloqueantes) produce un overhead en el cual los procesos se encuentran inactivos. Sin embargo, la implementación de mensajes no bloqueantes requiere de un análisis exhaustivo del problema para poder seleccionar zonas del programa en las que puedan solaparse cálculo y comunicación. Dado que la función que ha sido elegida como blanco de optimización no requiere sincronización intermedia, por el momento no hemos utilizado esta modalidad. Por otro lado la alta inestabilidad numérica del programa convierte este tipo de optimización en un problema demasiado complejo para ser tratado en el presente trabajo. El motivo de esta aseveración es que al utilizar mensajes no bloqueantes el orden de operaciones se ve modificado, y como hemos explicado en secciones anteriores esto es causal de resultados no deseados e incorrectos.

Pudo observarse en el profiling que existen diversas rutinas que pueden intentarse paralelizar. Uno de los ejemplos puntuales son las rutinas `IntSol` e `IntSolg` que consumen alrededor del $15\,\%$ del tiempo total.

Otras rutinas son las del tipo de la `int3` que invocan en gran número a la función `funct` lo cual produce un consumo del $5\,\%$ del tiempo total. Por otra parte queda como trabajo de desarrollo a futuro el uso de determinadas variables en memoria en vez de el cálculo de las mismas cada vez que son solicitadas, lo cual esta produciendo una perdida de performance importante.

Finalmente, queda pendiente la prueba del sistema en otras tecnologías para observar las diversas variables de performance (speedup, overhead, eficiencia).

Apéndice A

Glosario

- **Cluster**: Un cluster es un grupo de equipos individuales que trabajan en cooperación para proporcionar una mayor capacidad de cálculo y/o garantizar la disponibilidad contínua de las aplicaciones o los servicios escenciales. La principal ventaja de un cluster radica en el multiprocesamiento masivamente paralelo (muchos procesadores, memoria distribuida y red). Ver capítulo 2.
- **Beowulf**: Cluster Beowulf no es un software especial, ni una topología de red, ni un núcleo modificado. Beowulf es una tecnología diseñada para agrupar computadores basados bajo el sistema operativo Linux para formar un supercomputador virtual paralelo. Beowulf posee una arquitectura basada en multicomputadores el cual puede ser utilizado para la computación paralela. Este sistema consiste de un nodo maestro y uno o más nodos esclavos conectados a través de una red Ethernet u otra topología de red. Esta construido con componentes de hardware comunes, como ser cualquier PC capaz de ejecutar Linux, adaptadores de Ethernet y switches estándares. Una de las diferencias principales entre Beowulf y un cluster de estaciones de trabajo (COW, Cluster Of Workstations) es el hecho de que Beowulf se comporta más como una sola máquina que como muchas estaciones de trabajo conectadas. En la mayoría de los casos los nodos esclavos no tienen monitores o teclados y son accedidos solamente vía remota o por terminal serie. El nodo maestro controla el cluster entero y presta servicios de sistemas de archivos a los nodos esclavos. Ver capítulo 2.
- **Métodos híbridos**: Uno de los principales objetivos de la Química Computacional es el de obtener medidas sobre propiedades físico-químicas (tanto macroscópicas como microscópicas) mediante la aplicación de modelos teóricos al estudio de sistemas atómicos o moleculares. Podemos agrupar estos modelos teóricos como pertenecientes a dos grandes grupos, en función de la teoría física de la que hacen uso:

  1. Métodos basados en la Mecánica Cuántica
  2. Métodos basados en la Mecánica Clásica

  Los primeros, como indica su nombre, se han desarrollado a partir de la formulación de la mecánica cuántica aplicada. Métodos híbridos QM/MM proviene del inglés Quantum Mechanics / Molecular Mechanics. Esta metodología consiste en combinar cálculos de estructura electrónica con la mecánica molecular clásica a través de un Hamiltoniano híbrido (H_{qm-mm}).Ver capítulo 3

- **Onda plana**: Se denomina así a las ondas que poseen una dimensión, o, lo que es lo mismo, que su dirección de propagación es única. Ver capítulo 1.

- **Método de Hartree-Fock**: Es un procedimiento iterativo para calcular la mejor solución a la ecuación de Schrödinger independiente del tiempo, para moléculas aisladas, tanto en su estado fundamental como en estado de excitación. Este método permite calcular la energía de intercambio de la molécula de forma exacta, pero no tiene en cuenta el efecto de la correlación electrónica. HF realiza una aproximación de la energía total de la molécula calculando la interaccin de un único electrón con el resto de los electrones del sistema mediante el promedio de una interacción entre dos cuerpos. Ver capítulo 1.

- **Ecuación de Schrödinger**: desarrollada por el físico austríaco Erwin Rudolf Josef Alexander Schrödinger en 1925, describe la dependencia temporal de los sistemas mecanocuánticos. Es de importancia central en la teoría de la mecánica cuántica, donde representa un papel análogo a las leyes de Newton en la mecánica clásica.

- **Hamiltoniano**: En mecánica cuántica es el operador que retorna el total de energía de un sistema en forma de un autovalor. Ver capítulo 3.

- **Espectro Vibracional**: El espectro vibracional representa las frecuencias características de vibración de una dada molécula en un entorno. Las espectroscopias vibracionales son herramientas experimentales que brindan, en muchos casos, información microscópica sobre un sistema dado. Estas herramientas no siempre son concluyentes y requieren a menudo de cálculos teóricos para ser interpretadas. La obtención del espectro vibracional por métodos convencionales de estructura electrónica en vacío permite obtener sólo frecuencias de vibración dentro del contexto de la aproximación armnica. Por ello se utilizan también simulaciones de dinámica molecular híbrida QM/MM para obtener el espectro vibracional, ya que con este esquema se tiene en cuenta efectos de solvente (explícito) y de anarmonicidad. Ver capítulo 4.

Apéndice B

Hardware y Software utilizados

B.1. Hardware

Papita: Intel SMP Xeon Dual, cedida gentilmente por Intel Argentina.

FIU-PG (Florida): 17 Intel SMP Xeon Dual HT 2.6GHz, 1.96Gb RAM

B.2. Software

Lenguaje de programación: Fortran77, Fortran90

Compilador: g77 3.4.4, Intel Fortran Compiler 9.0

Sistema Operativo: Gentoo Linux 3.4.4

Implementaciones utilizadas de MPI: LAM-MPI 7.0.4 y MPICH 1.2.7p1

Índice de figuras

Índice de cuadros

Bibliografía

[1] C.J. Cramer. *Essentials of Computational Chemistry, Theories and Models.* John Wiley & Sons, Ltd., West Sussex, England, 2004.

[2] T. L. Sordo N. Diaz, D. Suarez and K. M. Merz Jr. A theoretical study of the aminolysis reaction of lysine 199 of human serum albumin with benzylpenicillin: Consequences for immunochemistry of penicillins. *J. Am. Chem. Soc.*, 2001.

[3] A.V. Nemukhin; I.A. Topol; B.L. Grigorenko; S.K. Burt. On the origin of potential barrier for the reaction $oh^{-} + co_2 \rightarrow hco_3$ in water: Studies by using continuum and cluster solvation methods. *J. Phys. Chem. B.*, 2002.

[4] J. Gao L.S. Devi-Kesavan. Combined qm/mm study of the mechanism and kinetic isotope effect of the nucleophilic substitution reaction in haloalkane dehalogenase. *J. Am. Chem. Soc.*, 2003.

[5] Mariano Camilo González Lebrero. *Simulación computacional de propiedades químicas mediante técnicas QM-MM*. PhD thesis, Facultad de Ciencias Exactas y Naturales, Departamento de Química inorgánica, Universidad de Buenos Aires, 2006.

[6] P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *Phys. Rev.*, 136:864, 1964.

[7] W. Kohn and L. J. Sham. Self-consistent equations including exchange and correlation effects. *Phys. Rev.*, 140:1133, 1965.

[8] A.D. Becke. Density-functional exchange-energy approximation with correct asymptotic behaviour. *Phys. Rev.*, A38:3098, 1988.

[9] R.G. Parr and W. Yang. *Density-Functional Theory of Atoms and Molecules*. Oxford University Press, NY, USA, 1989.

[10] W. Koch and M.C. Holthausen. *A Chemist's Guide to Density Functional Theory*. Wiley-VCH, Weinheim, 2002.

[11] D.A. Estrin, L. Paglieri, G. Corongiu, and E. Clementi. Small clusters of water molecules using density functional theory. *J. Phys. Chem.*, 100:8701, 1996.

[12] J.P. Perdew. Density-functional approximation for the correlation energy of the inhomogeneous electron gas. *Phys. Rev. B*, 33:8822, 1986.

[13] A.D. Becke. Density-functional thermochemistry. the role of exact exchange. *J. Chem. Phys.*, 98:5648, 1993.

[14] A.R. Leach. *Molecular Modeling, principles and aplications*. Addison Wesley Longman Limited, 1996.

[15] Truly distribution-independent algorithms for the n-body problem. `http://www.scl.ameslab.gov/Publications/Gus/N-Body/N-Body.html`.

[16] N-body problem, multistep methods for the n-body problem (http://burtleburtle.net/bob/math/multistep.html).

[17] P. Callahan and S.R. Kosaraju. A decomposition of multi-dimensional point-sets with applications to n-nearest-neighbors and n-body potential fields. *Proc. 24th Ann. ACM Sympos. Theory Comput.*, pages 546–556, 1992.

[18] D. van der Spoel, P.J. van Maaren, and H.J.C. Berendsen. A systematic study of water models for molecular simulation: Derivation of water models optimized for use with a reaction field. *J. Chem. Phys.*, 108:10220–10230, 1998.

[19] P.J. van Maaren and D. van der Spoel. Molecular dynamics simulations of water with novel shell-model potentials. *J. Phys. Chem. B*, 105:2618–2626, 2001.

[20] A. Szabo and N.S. Ostlund. *Modern Quantum Chemistry*. Mc Graw-Hill Publishing Company, NY, USA, 1989.

[21] M.W. Mahoney and W.L. Jorgensen. Quantum, intramolecular flexibility, and polarizability effects on the reproduction of the density anomaly of liquid water by simple potential functions. *J. Chem. Phys.*, 114:363–366, 2001.

[22] M.C. Gonzlez Lebrero, L.L. Perissinotti, and D.A. Estrin. Solvent effects on peroxynitrite structure and properties from qm/mm simulations. *J. Phys. Chem. A*, 109:9598–9604, 2005.

[23] K. Kiyohara, K.E. Gubbins, and A.Z. Panagiotopoulos. Phase coexistence properties of polarizable water models. *Molecular Phys.*, 94:803–808, 1998.

[24] P.G. Kusalik and I.M. Svishchev. The spatial structure in liquid water. *Science*, 265:1219–1221, 1994.

[25] I.M. Svishchev, P.G. Kusalik, J. Wang, and R.J. Boyd. Polarizable point-charge model for water: Results under normal and extreme conditions. *J. Chem. Phys.*, 105:4742–4750, 1996.

[26] H. Yu and W. F. van Gunsteren. Charge-on-spring polarizable water models revisited: From water clusters to liquid water to ice. *J. Chem. Phys.*, 121:9549–9564, 2004.

[27] H.A. Stern, F. Rittner, B.J. Berne, and R.A. Friesner. Combined fluctuating charge and polarizable dipole models: Application to a five-site water potential function. *J. Chem. Phys.*, 115:2237–2251, 2001.

[28] Peter S. Pacheco. *Parallel programming with MPI*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[29] G.M. Amdahl. Validity of the single-processor approach to achieving large scale computing capabilities. *AFIPS Conference Proceedings*, 1967.

[30] J.L. Gustafson. Reevaluating amdahl's law. *Commun. of the ACM*, 1988.

[31] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, Cambridge, MA, 1994.

[32] Beowulf.org is a collection of resources for the expanding universe of users and designers of beowulf class cluster computers. these enterprise systems are built on commodity hardware deploying linux os and open source software.

[33] Laboratorio de Sistemas Complejos. Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires. `http://www.lsc.dc.uba.ar`.

[34] PBS Professional, altair portable batch system. `http://www.altair.com/software/pbspro.htm`.

[35] MAUI, maui cluster scheduler. `http://www.clusterresources.com/pages/products/maui-cluster-scheduler.p%hp`.

[36] M.C. González Lebrero, D.E. Bikiel, M.D. Elola, D.A. Estrin, and A.E. Roitberg. Solvent-induced symmetry breaking of nitrate ion in aqueous clusters: A quantum-classical simulation study. (http://link.aip.org/link/?jcp/117/2718/1). *The Journal of Chemical Physics*, 2002.

[37] William D. Gropp and Ewing Lusk. Installation guide for `mpich`, a portable implementation of MPI. *Mathematics and Computer Science Division*, 1996.

[38] MPICH, a portable MPI implementation. `http://www.mcs.anl.gov/mpi/mpich`.

[39] LAM MPI Local Area Multicomputer - MPI parallel computing environment. (http://www.osc.edu/lam.html).

[40] Richard L. Burden and J. Douglas Faires. *Análisis Nmérico*. International Thomson Editores, San Francisco, CA, USA, 1998.

[41] Barry B. Brey. *Los Microprocesadores Intel*. Prentice Hall, Mexico, 2001.

[42] GCC, the GNU compiler collection. `http://gcc.gnu.org/`.

[43] Intel® fortran compiler 9.0 for linux. `http://www.intel.com/cd/software/products/asmo-na/eng/compilers/index.h%tm`.

[44] Intel® MPI library 2.0. `http://www.intel.com/cd/software/products/asmo-na/eng/cluster/index.htm%`.

[45] A. Kitebataka C. Szabo and I. Sakuma. *In Recent Advances in Nitric Oxide Research*. Springer-Verlag, 1999.

[46] J.S. Newman. *Electrochemical Systems*. Prentice Hall, 1973.

[47] R.F. Probstein. *Physicochemical Hydrodynamics, An Introduction*. John Wiley & Sons, Ltd., West Sussex, England, 1994.

[48] J.P. Perdew, K. Burke, and M. Ernzerhof. Generalized gradient approximation made simple. *Phys. Rev. Lett.*, 77:3865, 1996.

[49] C. Lee, W. Yang, and R.G. Parr. Development of the colle-salvetti correlation-energy formula into a functional of the electron density. *Phys. Rev. B*, 37:785, 1988.

[50] L.A. Baez and P. Clancy. Existence of a density maximum in extended simple point charge water. *J. Chem. Phys.*, 101:9837–9840, 1994.

[51] C. Jarzinski. Nonequilibrium equality for free energy differences. *Phis. Rev. Lett.*, 78:2690, 1997.

[52] J. Liphardt, S.B. Smith, I.Tinoco Jr., and C. Bustamante. Equilibrium information from nonequilibrium measurements in an experimental test of jarzynski's equality. *Science*, 296:1832, 2002.

[53] G.L. Squadrito, X. Jin, and W.A. Pryor. Stopped-flow kinetic study of the reaction of ascorbic acid with peroxynitrite. *Arch. Biochem. Biophys.*, 322:53–59, 1995.

[54] W.G. Keith and R.E. Powell. Kinetics of decomposition of peroxynitrous acid. *J. Chem. Soc. A*, page 90, 1969.

[55] S.V. Lymar and J.K. Hurst. Co2-catalyzed one-electron oxidations by peroxynitrite: Properties of the reactive. *Inorg. Chem.*, 37:294–301, 1998.

[56] M.G. Bonini, R. Radi, G. Ferrer-Sueta, A.M.D.C. Ferreira, and O. Augusto. Direct epr detection of the carbonate radical anion produced from peroxynitrite and carbon dioxide. *J. Biol. Chem.*, 274:10802–10806, 1999.

[57] L.F. Greengard. *The Rapid Evolution of Potential Fields in Particle Systems*. The MIT Press, 1988.

[58] J.D. Lambert and I.A. Watson. Symmetric multistep methods for periodic initial value problems. *J. Inst. Maths Applics.*, 18:189, 1976.

[59] Wikipedia, la enciclopedia libre. (http://es.wikipedia.org).