Facultad de Informática

Universidad Nacional de La Plata

Análisis de Rendimiento y Optimización de una Implementación Real de un Simulador de N-Cuerpos.

por
Federico J. Díaz
<fedediazceo@gmail.com>

Trabajo Final presentado para obtener el grado de Especialista en Cómputo de Altas Prestaciones y Tecnología Grid

Director: Fernando G. Tinetti
Fecha: 3/4/19

Tabla de contenidos

1 Introducción

Actualmente en la astronomía moderna, existe una rama de estudio dedicada exclusivamente a la evolución de sistemas planetarios. La evolución de estos sistemas, transcurre a lo largo de millones de años, con lo cual el objeto de estudio real imposibilita su simple observación, como ocurre en otras ramas de la ciencia.

Para obtener los datos necesarios para avanzar el conocimiento en este campo, los astrónomos se apoyan fuertemente en integradores numéricos. Estos integradores, son programas especialmente diseñados, que resuelven las ecuaciones de interacción gravitatoria, y requieren de un alto poder de cómputo, para lograr resultados en un tiempo razonable.

En la gran mayoría de los casos prácticos, los integradores se encuentran implementados en programas, escritos en el lenguaje FORTRAN, y a pesar de que son utilizados fuertemente en el ámbito científico, muchos de ellos poseen más de 15 años de antigüedad [1]. Debido a la antigüedad que poseen, las implementaciones son principalmente secuenciales.

La evolución de las tecnologías hardware de los últimos años, está marcada por la tendencia de los fabricantes de implementar paralelismo. Por esta razón, cuando un grupo de investigación adquiere hardware más potente para ejecutar sus integradores, de naturaleza secuencial, no se logra una mejora sustancial en los tiempos de ejecución de los mismos.

En el grupo de integradores que se utilizan comúnmente en las ciencias planetarias, se encuentran Swift [2], y Mercury [3], siendo este ultimo el objeto de estudio de este trabajo. A lo largo del presente, se realizará un análisis de rendimiento e intento de optimización del integrador de N-Cuerpos Mercury.

Este integrador entra en la categoría de lo denominado "software heredado" (legacy software). Este software heredado tiene la particularidad de estar específicamente relacionado con aplicaciones científicas que podrían ser consideradas clásicas de cómputo de alto rendimiento.

Las características de los integradores mencionados, son las siguientes

1.1.1 Swift

Este conjunto de integradores numéricos, es uno de los más utilizados en simulaciones astrofísicas que no buscan simular eventos llamados "Encuentros Cercanos" (Cuerpos a una distancia menor a un radio crítico definido). Swift ha sido optimizado en GPU, con cierto éxito [4], pero no se ha propuesto una metodología concreta de optimización. Swift actualmente se encuentra integrado dentro de Mercury

1.1.2 Mercury

Este conjunto de integradores, es el más utilizado cuando lo que se buscan son "Encuentros Cercanos" y colisiones de cuerpos, ya que numéricamente contempla estos casos. Lamentablemente, al incorporar las funcionalidades correspondientes a estos eventos, el cálculo pasa a ser un problema intenso de cómputo, obteniendo tiempos de ejecución extremadamente largos.

En la siguiente sección analizaremos el caso de estudio junto con la motivación para el presente trabajo. En la sección 3 realizaremos un análisis del código del integrador Mercury, estudiando los puntos iniciales de optimización. En la sección 4 se detallarán las optimizaciones secuenciales propuestas, y en la sección 5 se hará foco en las optimizaciones paralelas propuestas. Finalmente, en la sección 6 se mostrarán los resultados obtenidos de las optimizaciones implementadas, y en la sección 7 se explicarán las conclusiones junto con las posibles líneas de investigación futuras.

2 Caso de Estudio

Partiendo de una gran cantidad de cuerpos pequeños y medianos en un volumen determinado, utilizando reglas físicas determinadas, los integradores modifican las posiciones y velocidades de estos cuerpos, a lo largo del tiempo. A su vez, se definen el tiempo de integración deseado, y el paso incremental de tiempo a utilizar.

Este último parámetro es de suma importancia, porque define la precisión con la cual se obtienen resultados, y la forma en que los integradores realizan su cómputo. Ejemplo: Si realizamos una simulación por 100 mil años, con un paso de 10 mil años, el integrador realizara solo 10 ciclos de integración, con mil años, serían 100 ciclos.

El problema que los integradores resuelven, es el denominado "problema de los N-Cuerpos" [5]. En este tipo de problemas, para cada paso de simulación se interrelacionan las posiciones y las masas de cada cuerpo, resolviendo las ecuaciones diferenciales gravitatorias de Kepler. Un ejemplo de resolución del problema de N-Cuerpos, se ve en la figura 1.

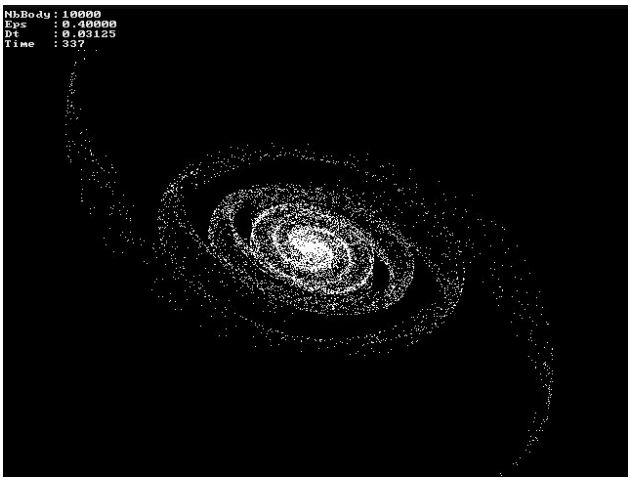

Figura 1. Resultado de una simulación de N-Cuerpos [6]

Lamentablemente, estas ecuaciones diferenciales no tienen una solución analítica determinística, cuando la cantidad de cuerpos es mayor a 2, con lo cual la utilización de métodos numéricos es la única manera de obtener un resultado. Los integradores numéricos [7] en general, son extensamente utilizados en simulaciones numéricas dentro del área de la astrofísica. Se caracterizan principalmente por requerir una gran cantidad de cómputo (operaciones de

punto flotante en su mayoría de doble precisión), y los algoritmos implementados, en muchos casos, se encuentran en complejidades computacionales de órdenes cuadráticos.

Esto ocurre porque la solución tiene que realizar, para cada paso de simulación, una interacción de cada cuerpo, contra todos los restantes. En la figura **2** se ven estas interacciones.

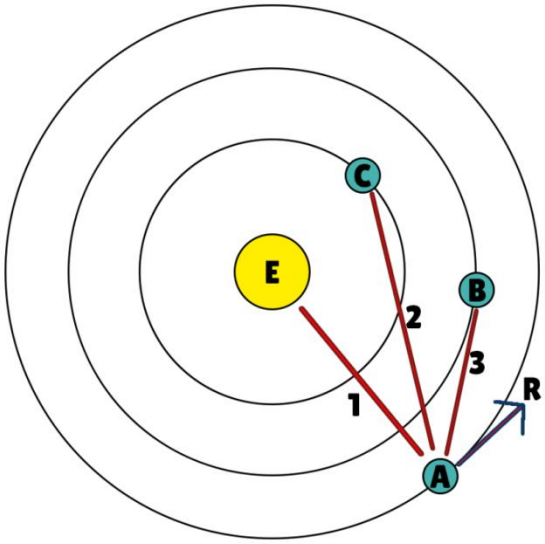

Figura 2. Interacción realizada por los integradores numéricos durante un paso de simulación

En la figura 2, El Cuerpo **A** calcula, en este caso la velocidad resultante **R**, mediante las interacciones **1**,**2** y **3** entre el cuerpo **B**, **C** y el astro central **E.** Luego en el mismo paso, se repetirá el proceso para B y C, dando por completado un paso de simulación

Los problemas utilizados en la bibliografía orientada a ciencias de la computación realizan experimentaciones sobre la expresión básica de interacción de fuerzas entre cuerpos mencionada. La fórmula que define estas interacciones, es la mostrada en la figura 3.

$$\ddot{\mathbf{r}}_i = -G \sum_{j=1;\, j\neq i}^{N} \frac{m_j(\mathbf{r}_i - \mathbf{r}_j)}{|\mathbf{r}_i - \mathbf{r}_j|^3}\,.$$

Figura 3. Ecuación de interacción gravitatoria entre masas puntuales

En la figura 3, se observa que los únicos parámetros intervinientes son la distancia entre cuerpos (r), las masas (m) y la constante gravitatoria (G). En las implementaciones comúnmente encontradas, se resuelve esta fórmula por cada paso de simulación, y esto se corresponde con el proceso denominado "integración numérica". [8]

Sin embargo, en la investigación astronómica real, se utiliza una matemática más compleja para obtener soluciones precisas, y adaptadas a las observaciones realizadas. En estos casos, se utilizan otro tipo de algoritmos, llamados "Integradores simplécticos" [9], los cuales parten de resolver el sistema Hamiltoniano, expresado a modo de referencia en la figura 4

$$\frac{dx_i}{dt} = \frac{\partial H}{\partial p_i},$$

$$\frac{dp_i}{dt} = -\frac{\partial H}{\partial x_i},$$

Figura 4. Ecuación Hamiltoniana de movimiento.

En la figura 4, los parámetros **x** y **p** representan las coordenadas y el momento de cada cuerpo, **t** el tiempo, y **H** representa el "Hamiltoniano" del sistema [10]

En la figura 5, a modo de referencia, se puede observar las operaciones realizadas por Mercury en el cálculo de encuentros cercanos. Estas son soluciones específicas al sistema Hamiltoniano mencionado previamente.

La diferencia en cómputo requerida es notable, respecto a los cálculos de masas simples, donde el término H_1, representa solamente el cálculo de la interacción gravitatoria clásica. El tratamiento de la solución en partes (H_0, H_1, H_2), es lo que caracteriza la implementación de un integrador simpléctico [11]

$$
\begin{aligned}
H_0 &= \sum_{i=1}^{N} \left(\frac{p_i^2}{2m_i} - \frac{Gm_\odot m_i}{r_{i\odot}} \right) \\
H_1 &= -G \sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{m_i m_j}{r_{ij}} \\
H_2 &= \frac{1}{2m_\odot} \left(\sum_{i=1}^{N} \mathbf{p}_i \right)^2
\end{aligned}
$$

Figura 5. Solución al Hamiltoniano de movimiento para encuentros cercanos.

Existen ciertas implementaciones paralelas de Mercury [12], pero en este caso, también la falta de una metodología clara de optimización, no nos provee una forma sencilla de extrapolar las técnicas utilizadas en estas implementaciones, para aplicar en otros integradores.

2.1 Motivación

El integrador Mercury, se encuentra en su totalidad diseñado en forma secuencial. Las tecnologías actuales, se encuentran especializadas en incrementar el número de procesadores paralelos, en lugar de aumentar la frecuencia de operación en sistemas mono-procesador.

Los problemas secuenciales se benefician principalmente del aumento de frecuencia y memoria de los procesadores físicos, no así del incremento de procesadores en forma paralela, debido a que el paralelismo automático no es eficiente.

Las optimizaciones automáticas que existen actualmente, no producen aun los mismos resultados que las optimizaciones realizadas mediante una metodología manual [13]. Esto implica que, si se quiere mejorar la eficiencia en el uso de recursos, las herramientas automáticas no darán los mejores resultados.

Teniendo en cuenta lo anterior, se determina que el objetivo general de este trabajo es el análisis de rendimiento y el intento de lograr una implementación optimizada del integrador Mercury. Con este último fin, se aplicaron técnicas de optimización secuencial, concretamente reordenamiento de instrucciones, e introducción de funciones en línea), como se detalla en la sección 4. También, se aplico una técnica de optimización de Paralelismo en memoria compartida, eliminando dependencia de datos, como se detalla, en la sección 5.

Para poder estudiar el comportamiento del integrador Mercury, el departamento de ciencias planetarias de la Facultad de Ciencias Astronómicas y Geofísicas de la Universidad de La Plata, proporcionó **dos** casos de estudio reales.

El primer caso, llamado **"Caso de cuerpos pequeños"**, consiste en una lista de 37 cuerpos de poca masa y un cuerpo de mayor masa. El integrador realiza una operación llamada "Caso de 3 cuerpos restringido".

En este caso, por cada paso de integración, se calcula la interacción mutua entre un cuerpo pequeño, el cuerpo de mayor masa, y la estrella central. Se desprecian todas las interacciones entre los cuerpos pequeños.

Este caso presenta ciertas propiedades. Entre ellas, que la solución se puede predecir numéricamente (por cada paso de simulación se resuelve un problema de 3 cuerpos reiteradas veces), con lo cual se puede obtener una validación precisa de los resultados por el grupo de investigación de ciencias planetarias.

El segundo caso presentado, llamado **"Caso de cuerpos grandes"**, es también una lista, pero de 57 cuerpos. En este caso, por cada paso de la integración, se calculan las interacciones entre todos los cuerpos intervinientes. Aquí se presentan ciertos inconvenientes:

- Los resultados del integrador no se pueden validar en forma numérica precisa, solo por análisis de error en la salida producida
- Los cuerpos interactúan todos entre sí, generando una gran cantidad de cómputo por cada paso de la integración.
- Los cuerpos grandes pueden colisionar, generando una situación específica, donde dos cuerpos se fusionan en un tercero, modificando la lista de cuerpos en el siguiente paso de integración.

Cabe aclarar, que estos dos contextos de ejecución utilizados, no son comparables entre sí. A modo de ejemplo, el caso de cuerpos pequeños, se utiliza para evaluar la evolución de pequeños asteroides muy distantes entre sí, cuya interacción mutua es despreciable.

El caso de cuerpos grandes, se utiliza para evaluar la evolución de cuerpos considerablemente grandes, cuya interacción mutua es notable. Esto se traduce también, en una diferencia sustancial en el cómputo del integrador: Tiene que calcular la interacción entre todos los cuerpos, por cada paso de integración.

Realizar una optimización paralela de este integrador, por todo lo exhibido anteriormente, puede generar un impacto significativo en los actuales grupos de investigación que lo utilizan. Al aprovechar mejor el hardware, y reducir los

tiempos de ejecución, el grupo de investigación puede realizar un eficiente uso de recursos, y obtener mejores respuestas a sus problemáticas en menor tiempo.

2.1.1 Hardware Paralelo: CPU

Existen diversos fabricantes, entre ellos se puede mencionar a AMD [14], Intel [15], ARM [16], NVIDIA [17], entre otros. Los fabricantes mencionados, se encuentran actualmente investigando y comercializando procesadores con un alto nivel de paralelismo de instrucciones y datos, que permitirían ejecutar algoritmos optimizados con un buen rendimiento de MFLOP/s. Estos procesadores son ideales para resolver problemas donde existe paralelismo a nivel de instrucción, debido a que cada unidad de ejecución es lo suficientemente compleja como para operar en forma independiente del resto.

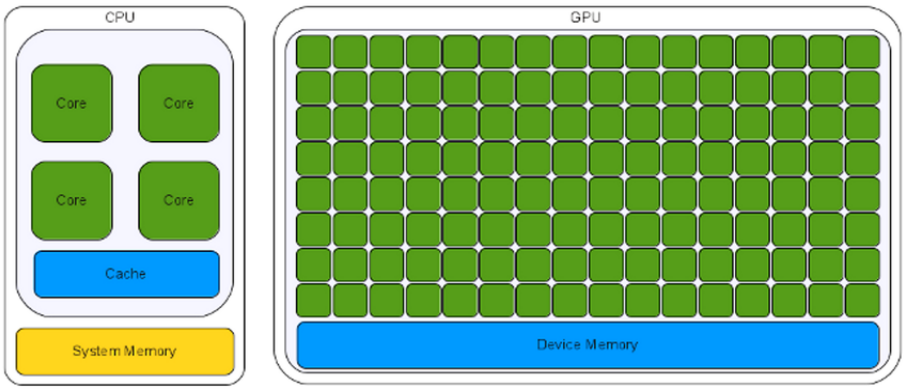

Figura 6. Arquitecturas multi-core y arquitecturas many-core

A su vez, debido a la tendencia mencionada en paralelismo otorgada por los fabricantes, a medida que las tecnologías sigan evolucionando hacia recursos que faciliten el ultra-paralelismo, el software secuencial perderá valor de utilidad cada vez más rápidamente. En la figura 6, se pueden observar las arquitecturas "multi-core", versus las arquitecturas "many-core" [18], donde la reducción en el costo de fabricación de transistores, permite una producción masiva de procesadores con una cantidad significativa de núcleos, a precios accesibles.

Los procesadores "multi-core", son dispositivos CPU compuestos por pocos núcleos, con mucha memoria cache, y pueden manejar pocos hilos de ejecución en simultaneo. En contrapartida, los procesadores "many-core" son dispositivos GPU equipados con cientos de núcleos, con la capacidad de manejar miles de hilos en forma simultánea

Los integradores numéricos mencionados, en sus implementaciones originales, no aprovechan el potencial de ejecución disponible, al utilizar una sola unidad de ejecución en forma secuencial. Replicando la metodología propuesta, se podría actualizar estos algoritmos a un uso eficiente de los recursos de CPU.

3 Análisis de Código

A continuación, se realizará un análisis general del código y de rendimiento del integrador numérico utilizado en simulaciones de N-Cuerpos, Mercury[19]. Mediante este análisis, se detallará el trabajo y la metodología de optimización, junto con los cambios propuestos, para lograr las optimizaciones deseadas

3.1 Análisis del integrador Mercury

Utilizando las dos configuraciones específicas propuestas por el grupo de investigación de ciencias planetarias de la Facultad de Ciencias Astronómicas y Geofísicas de la Universidad de La Plata, se realizó un estudio de desempeño sobre el código, analizando los resultados de ejecución con gprof [20], para detectar los puntos clave de optimización.

3.1.1 Elementos del paquete Mercury

- mercury6_x.for

Este es el programa principal de integración numérica. Contiene todas las rutinas necesarias para realizar una simulación de N-Cuerpos con una configuración determinada. Mercury produce una salida en lenguaje de máquina que es procesada con los dos programas mencionados debajo.

- element6.for y close.for

El primer programa convierte la salida de una ejecución con Mercury, en un conjunto de elementos orbitales que permite analizar el cambio de la órbita en el estado final del sistema. El segundo programa permite analizar las situaciones de encuentros cercanos entre cuerpos.

De ahora en más, nos enfocaremos en "mercury6_x.for", siendo la versión pública de Mercury la 6.2. Para ejecutar nuestros experimentos, utilizamos gfortran en forma clásica, añadiendo el flag "-pg" para habilitar gprof en el binario, y realizar el análisis correspondiente. Para compilar se utiliza:

- gfortran [–O(1,2,3)] –pg mercury6_[x].for -o mercury6

Luego, por una cuestión de preferencia a la hora de trabajar con el código, se le realizó una modificación para hacerlo compatible con Fortran (La versión utilizada es la del estándar F90), sin que esto implique cambios de ningún tipo en los tiempos de ejecución ni resultados. Esto permitirá utilizar las mejoras provistas por Fortran en esta versión, ya que el código original se encuentra escrito en FORTRAN (La versión del estándar F77).

Para tomar una métrica real de tiempo de ejecución, se utiliza lo que se llama un "wall clock". Esto es una simple marca de tiempo antes de la ejecución, y una marca de tiempo posterior a la ejecución, que toma en cuenta todo el contexto en el cual el programa está corriendo.

De esta manera se mide en forma correcta la percepción que el programa provoca en el usuario. Para tomar estas marcas de tiempo, se utilizaron las líneas de código de la figura 7

```
INTEGER t1,t2,clock_rate, clock_max
CALL system_clock ( t1, clock_rate, clock_max )

  ---Programa principal---

CALL system_clock ( t2, clock_rate, clock_max )
write ( *, * ) 'totalTime: ', real(t2-t1)/real(clock_rate)
```

Figura 7. Líneas de código utilizadas para toma de tiempos

El "wall clock" se utiliza, porque los datos obtenidos de gprof, solo miden los tiempos de ejecución de una función, sin tener en cuenta el impacto real de tiempo que percibe el usuario al ejecutar el programa. Gprof resulta útil para medir el impacto de funciones en términos relativos, es decir para comparar ejecución entre funciones, pero no para medir tiempos reales de ejecución.

3.1.2 Configuración inicial

El grupo de ciencias planetarias de la Facultad de Ciencias Astronómicas y Geofísicas de la Universidad Nacional de La plata (FCAGLP), realizó la configuración inicial del simulador para los dos problemas planteados.

En el primero se utilizo un caso de simulación en el cual existen 37 cuerpos pequeños y un cuerpo grande. Este caso, es el "**caso de cuerpos pequeños**", llamado anteriormente el "problema de 3 cuerpos restringido".

Para el segundo caso, se utilizaron 57 cuerpos grandes que interactúan entre sí, dando lugar a un problema de N-cuerpos completo, añadiendo colisiones a la ejecución. Este es el llamado "**Caso de cuerpos grandes**".

Para el primer caso se utilizo la configuración de la figura 8. Para el segundo, se utilizo la configuración de la figura 9.

```
)O+_06 Integration parameters  (WARNING: Do not delete this line!!)
) Lines beginning with `)' are ignored.
)------------------------------------------------------------------------
) Important integration parameters:
)------------------------------------------------------------------------
 algorithm (MVS, BS, BS2, RADAU, HYBRID etc) = hyb
 start time (days)= 0.
 stop time (days) = 365.25D8
 output interval (days) = 365.25D4
 timestep (days) = 1826.25
 accuracy parameter=1.d-12
)------------------------------------------------------------------------
) Integration options:
)------------------------------------------------------------------------
 stop integration after a close encounter = no
 allow collisions to occur = no
 include collisional fragmentation = no
 express time in days or years = years
 express time relative to integration start time = yes
 output precision = high
 < not used at present >
 include relativity in integration= no
 include user-defined force = no
)------------------------------------------------------------------------
) These parameters do not need to be adjusted often:
)------------------------------------------------------------------------
 ejection distance (AU)= 100.
 radius of central body (AU) = 0.1
 central mass (solar) = 1.0
 central J2 = 0
 central J4 = 0
 central J6 = 0
 < not used at present >
 < not used at present >
 Hybrid integrator changeover (Hill radii) = 3.
 number of timesteps between data dumps = 500
 number of timesteps between periodic effects = 100
```

Figura 8. Este archivo *param.inc* representa los parámetros de simulación para el "**caso de cuerpos pequeños**". No se permite la colisión entre cuerpos

En el archivo de configuración del caso de cuerpos pequeños, mostrado en la figura 8, se indica que la integración se realizará por 100 millones de años, con un paso de simulación de 5 años. Esto da un total, de 20 millones de pasos de integración.

```
)O+_06 Integration parameters  (WARNING: Do not delete this line!!)
) Lines beginning with `)' are ignored.
)------------------------------------------------------------------------
) Important integration parameters:
)------------------------------------------------------------------------
 algorithm (MVS, BS, BS2, RADAU, HYBRID etc) = hyb
 start time (days)= 0.
 stop time (days) = 1826.25E4
 output interval (days) = 365.25E4
 timestep (days) = 3.0
 accuracy parameter=1.d-12
)------------------------------------------------------------------------
) Integration options:
)------------------------------------------------------------------------
 stop integration after a close encounter = no
 allow collisions to occur = yes
 include collisional fragmentation = no
 express time in days or years = years
 express time relative to integration start time = yes
 output precision = medium
 < not used at present >
 include relativity in integration= no
 include user-defined force = no
)------------------------------------------------------------------------
) These parameters do not need to be adjusted often:
)------------------------------------------------------------------------
 ejection distance (AU)= 1000
 radius of central body (AU) = 0.01
 central mass (solar) = 1.0
 central J2 = 0
 central J4 = 0
 central J6 = 0
 < not used at present >
 < not used at present >
 Hybrid integrator changeover (Hill radii) = 3.
 number of timesteps between data dumps = 50000
 number of timesteps between periodic effects = 100
```

Figura 9. El archivo *param.inc* representa los parámetros de simulación para el "**caso de cuerpos grandes**". Se permite la colisión de cuerpos, que luego de colisionar, en el siguiente paso de simulación, se convierten en un nuevo cuerpo

En el archivo de configuración del caso de cuerpos pequeños, mostrado en la figura 9, se indica que la integración se realizará por 20 mil años, con un paso de simulación de 3 días. Esto da un total de aproximadamente 2.433.333 millones de pasos de integración.

3.1.3 Puntos de optimización

En la ejecución del caso de cuerpos pequeños, sin interacción entre sí, la salida del perfilador gprof para la ejecución inicial, es la mostrada en la tabla 1. Los resultados de la ejecución de cuerpos grandes, donde se producen encuentros cercanos, y colisiones, se muestran en la tabla 2.

Cada muestra cuenta como 0,01 segundos				
% Tiempo	**Segundos Acumulados**	**Segundos Propios**	**Llamadas a función**	**Nombre**
18,32	92,75	92,75	457097800	drift_dan_
11,99	153,48	60,72	20000000	mdt_hy_
9,02	199,17	45,69	372415975	drift_kepmd_
7,81	238,73	39,57	138907774	mfo_hkce_
7,52	276,83	38,10	20000025	mfo_drct_
7,14	312,97	36,13	4758093	mdt_bs2_
6,44	345,60	32,64	40290002	mco_dh2h_

Tabla 1. Salida de gprof para la ejecución del caso base para el "**caso de cuerpos pequeños**"

Cada muestra cuenta como 0,01 segundos				
% Tiempo	**Segundos Acumulados**	**Segundos Propios**	**Llamadas a función**	**Nombre**
66,34	773,40	773,40	6087504	mfo_drct_
10,92	900,66	127,26	6087500	mce_snif_
5,66	966,70	66,04	331355993	drift_dan_
3,71	1010,00	43,30	6087500	mdt_hy_
3,37	1049,28	39,28	331355993	drift_kepmd_
2,08	1073,50	24,22	12236126	mco_dh2h_

Tabla 2. Salida de gprof para la ejecución del caso base para el "**caso de cuerpos grandes**"

En todos los casos, en las tablas, se muestran las funciones predominantes desde el punto de vista del tiempo de cómputo. Nótese en las tablas 1 y 2, las diferencias entre las llamadas a función, correspondientes a los contextos de

ejecución diferentes definidos por los casos de entrada (caso de cuerpos pequeños y caso de cuerpos grandes)

Obsérvese además, las diferencias de cómputo que se producen entre ambos casos, mientras que en el primer caso, de cuerpos pequeños sin interacción entre sí, se puede observar que la función DRIFT_DAN es llamada un gran número de veces en comparación con el resto de las funciones mostradas en la tabla. La función DRIFT_KEPMD es llamada dentro de DRIFT_DAN, con lo cual se podría estudiar la ventaja de eliminar la sobrecarga de llamada en estas dos funciones, realizando una inserción "en línea" (del inglés "inline"), dentro de la función llamadora.

En el caso de cuerpos grandes con interacción, aparece la función MFO_DRCT, que es la encargada de realizar las interacciones entre los mismos. Esta función está ocupando más del 50% del tiempo, con lo cual es el punto directo de optimización en el cual vamos a ocupar la mayor parte del esfuerzo de optimización.

Los tiempos de ejecución medidos con el "wall clock" están detallados en la tabla 3. Es de importante consideración establecer que todos los resultados numéricos obtenidos de las ejecuciones fueron evaluados por el grupo colaborador de astronomía de la FCAGLP.

El menor tiempo en el caso de cuerpos grandes mostrado en la tabla 3, se da por una reducción significativa en el tiempo de integración mencionado. Este caso requiere de más poder de cómputo al realizar la interacción entre todos los cuerpos de la lista, pero para poder obtener resultados en un tiempo razonable, y por practicidad, se le redujo el tiempo de integración considerablemente.

Tiempos caso base: Cuerpos pequeños	2559,8511 segundos
Tiempos caso base: Cuerpos grandes	1234,9270 segundos

Tabla 3. Tiempos de partida para los casos base de ambos problemas.

La discrepancia de tiempos observada en las ejecuciones, respecto de los valores medidos con gprof, es una recurrencia común al realizar análisis de programas en cómputo de alto desempeño. Esta discrepancia ocurre, porque las mediciones de gprof, se toman como una muestra cada 0,01 segundos en este caso, y consultando en qué función se encuentra en el contexto del programa el puntero de instrucciones del procesador.

Es por eso que se mencionó que los valores obtenidos por gprof, sirven a punto comparativo entre funciones, no en forma absoluta para medir tiempos de ejecución. Gprof no mide todo lo que ocurre en el sistema por fuera del integrador, durante la ejecución del mismo.

En forma empírica, podemos realizar la siguiente comprobación para comprobar estos resultados: Para el caso de cuerpos pequeños, cuya salida está en la tabla 1, gprof nos indica que la función DRIFT_DAN ocupa un 18,32% del tiempo total de ejecución, con 92,75 segundos ocupados. Al extrapolar el 100% del tiempo de ejecución tomando estos valores, obtenemos que el tiempo total de ejecución medido por gprof, sea de unos 506,27 segundos, aproximadamente.

Sin embargo, el tiempo de reloj total medido, acumula un total de 2559,8511 segundos. Aquí es donde en nuestro análisis a priori, detectamos que el caso de cuerpos pequeños, está realizando una excesiva entrada/salida, la cual no es medida por gprof, ya que la entrada salida es realizada por bibliotecas de sistema, externas al integrador

4 Optimizaciones Secuenciales Realizadas

El primer acercamiento hacia las optimizaciones propuestas, es realizar una mejora de la ejecución del programa en forma secuencial. Las modificaciones propuestas en esta sección buscan mejorar la calidad del código previo a una optimización paralela, pero sin perder el foco en la misma.

Esto vale aclararlo, ya que cierto tipo de optimizaciones secuenciales, generan dependencia de datos, lo cual evita que se pueda realizar una paralelización en forma directa. Sin embargo, las mejoras presentadas en esta sección, se enfocan en la eliminación de entrada/salida, la re-ingeniería del código utilizando funciones "en línea" (inline), lo cual no genera esta dependencia mencionada.

4.1 Eliminación de entrada y salida redundante

Aunque esto no se ve reflejado en las salidas de gprof en cada ejecución, el programa producía salidas del tipo mostrado en la figura 10. Estas salidas por consola generaban un retraso innecesario en el cómputo, con lo cual fueron eliminadas en todos los casos para evitar demoras de entrada/salida que no son pertinentes a la hora de realizar optimizaciones en cómputo de altas prestaciones

```
Integrating massive bodies and particles up to the same epoch.
Beginning the main integration.
Time:     2500.000 years  dE/E:  2.47703E-13  dL/L:  1.04563E-13
Time:     5000.000 years  dE/E: -5.32687E-13  dL/L: -3.12967E-13
.....Continúa salidas similares.....
Time:  99997500.000 years  dE/E:  4.83925E-11  dL/L:  2.41502E-11
Time: 100000000.000 years  dE/E:  4.82842E-11  dL/L:  2.41247E-11
Integration complete.
totalTime:   2559.8511
```

Figura 10. Ejemplo de entrada/salida por consola del programa. Este tipo de salida informativa solo retrasa el cómputo y es innecesaria a la hora de producir optimizaciones sobre el mismo

4.2 Utilización de funciones intrínsecas

En la función DRIFT_KEPMD, se encontraron las líneas de código mostradas en la figura 11. Estas intentan reemplazar el cálculo de las funciones intrínsecas del lenguaje, que se encuentran optimizadas para su uso.

```
------------------------------------------------------
!         y = x*x
¡         s = x*(A0-y*(A1-y*(A2-y*(A3-y*(A4-y)))))/A0
¡         c = sqrt(1.d0 - s*s)
------------------------------------------------------
         s = sin(x)
         c = cos(x)
```

Figura 11. Ejemplo de reemplazo de cálculo manual de funciones trigonométricas por las funciones intrínsecas de Fortran. Este reemplazo se repitió en todas las secciones de la función donde se encontró un cálculo similar

4.3 Conversión de una función llamada múltiples veces a forma "en línea" (inline)

Luego de analizar la salida de gprof, se observa que la función DRIFT_DAN y DRIFT_KEPMD (siendo dos de las 3 funciones más utilizadas), se llaman una cantidad relativamente muy grande de veces, como se detalla en la tabla 4.

Cada muestra cuenta como 0,01 segundos				
% Tiempo	**Segundos Acumulados**	**Segundos Propios**	**Llamadas a función**	**Nombre**
18,32	92,75	92,75	457097800	drift_dan_
11,99	153,48	60,72	20000000	mdt_hy_
9,02	199,17	45,69	372415975	drift_kepmd_

Tabla 4. Extracto de la tabla 1, para el "**caso de cuerpos pequeños**", mostrando solo las funciones que mas cómputo realizan

El experimento siguiente consiste en convertir estas funciones a un formato "en línea", para estudiar la posible reducción de tiempos de ejecución producida por la sobrecarga de la llamada a las mismas. El estado original de las funciones se puede ver en las figuras 12 y 13.

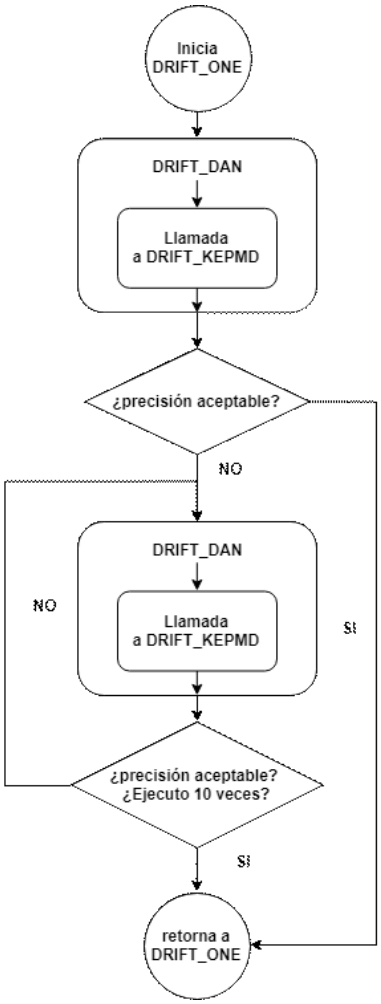

Figura 12. Abstracción del esquema de flujo de la función DRIFT_ONE sin modificaciones. La función DRIFT_ONE es llamada para cada uno de los cuerpos en el sistema, para cada paso de simulación, con lo cual la cantidad de llamadas se multiplica, provocando una posible sobrecarga en la llamada a las mismas.

```
------ CÓDIGO DE DRIFT_ONE----------------------------------------
         call drift_dan(mu,x,y,z,vx,vy,vz,dt,iflg)

         if(iflg .ne. 0) then

           do i = 1,10
             dttmp = dt/10.d0
             call drift_dan(mu,x,y,z,vx,vy,vz,dttmp,iflg)
             if(iflg .ne. 0) return
           enddo

         endif
------- CONTINUA CÓDIGO DE DRIFT_ONE -------------------------------
```

Figura 13. Código original de DRIFT_ONE, donde se observa la llamada a DRIFT_DAN, la cual a su vez contiene la llamada sucesiva a DRIFT_KEPMD

En el estado original, DRIFT_DAN realiza la llamada a DRIFT_KEPMD como se ve en la figura 12. Inicialmente lo que se implementó, fue DRIFT_KEPMD "en línea" dentro de DRIFT_DAN, para reducir las llamadas internas de la misma. Este proceso se realizo en forma directa, ya que DRIFT_KEPMD no poseía sentencias GOTO, lo cual no es el caso de DRIFT_DAN.

Esta última, utilizaba estas sentencias, para resolver un problema de ciclos en la lógica, con lo cual requirió un proceso de análisis para cambiar este flujo de ejecución que se ve en la figura 14, en uno que respete los estándares del paradigma procedural. La utilización de la sentencia GOTO se puede ver también en la figura 15

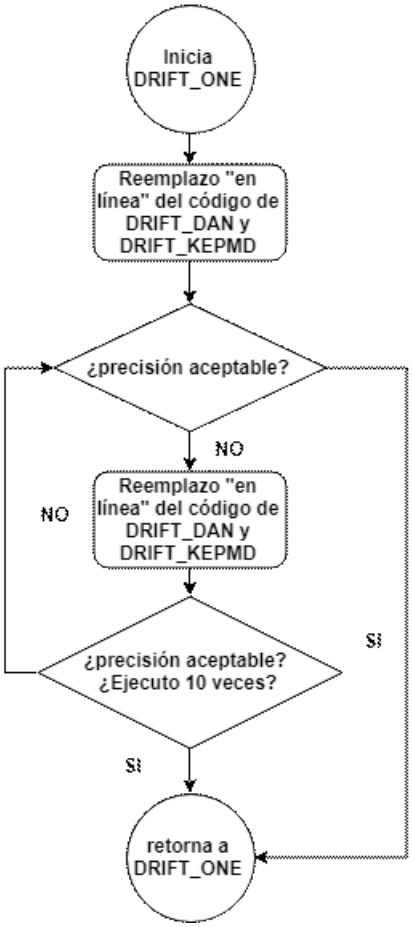

Figura 14. Abstracción del esquema de flujo de la función DRIFT_ONE con la modificación "en línea" (o "inline").

```
-------- Continúa código de DRIFT_DAN ----------------
         dm = dt*en - int(dt*en/TWOPI)*TWOPI
         dt = dm/en
         if((dm*dm .gt. 0.16d0) .or. (esq.gt.0.36d0)) goto 100
         if(esq*dm*dm .lt. 0.0016) then
             call drift_kepmd(dm,es,ec,xkep,s,c)
             fchk = (xkep - ec*s +es*(1.-c) - dm)
             if(fchk*fchk .gt. DANBYB) then
                iflg = 1
                return
             endif
-------- Continúa código de DRIFT_DAN ----------------
```

Figura 15. Fragmento de Código original de DRIFT_DAN, donde se observa la llamada a DRIFT_KEPMD, junto con una llamada a una sentencia GOTO

Una ventaja añadida respecto de la modificación "en línea", es que se eliminaron sentencias GOTO como la que se muestra en la figura 14, lo cual siempre produce un resultado de código mejor escrito, y más de acuerdo con los paradigmas modernos de Ingeniería de Software [21], (independientemente de si esta modificación produce una mejora de performance). El resultado final de esta conversión, es reducir las llamadas a DRIFT_DAN y DRIFT_KEPMD, para que solo ocurra una llamada por ciclo a DRIFT_ONE, y así estudiar si la reducción de llamadas produce una mejora aceptable en el rendimiento del integrador.

```
¡...  Inicio de DRIFT_ONE
!...  Necesario para DRIFT_KEPMD INLINE
      real*8 dxkep
      real*8 fac1kep,fac2kep,qkep,ykep
      real*8 fkep,fpkep,fppkep,fpppkep
!--------------------------------

      ¡... Inicio Código de la rutina de DRIFT_DAN
        ¡... Inicio Código de la rutina de DRIFT_KEPMD
        ¡... Fin Código de la rutina de DRIFT_KEPMD
      ¡... Fin Código de la rutina de DRIFT_DAN

          if(iflg .ne. 0) then

            do i = 1,10
              dttmp = dt/10.d0
              ¡... Inicio Código de la rutina de DRIFT_DAN
                 ¡... Inicio Código de la rutina de DRIFT_KEPMD
                 ¡... Fin Código de la rutina de DRIFT_KEPMD
              ¡... Fin Código de la rutina de DRIFT_DAN
              if(iflg .ne. 0) return
            enddo

          endif
!.............................
```

Figura 16. DRIFT_DAN y DRIFT_KEPMD en forma "en línea".

Nótese en la figura 16, que se realizo además la unión de las llamadas internas de la misma. La función consistía en operaciones matemáticas básicas, con lo cual se considero posible que la optimización "en línea" produzca algunos resultados favorables.

Por otro lado, la eliminación de sentencias GOTO, no se ha podido resolver en forma automática por los compiladores a la fecha. Lamentablemente requiere de un análisis profundo del flujo de ejecución, para el cual el analizador estático del compilador no puede encontrar una conversión directa

4.4 Reducción de operaciones redundantes y simplificación de código

En la ejecución de 57 cuerpos, el caso de cuerpos grandes, se observa la llamada a la función MFO_DRCT. A diferencia del caso de cuerpos pequeños, esta función, encargada de procesar las interacciones entre los 57 cuerpos, es llamada una gran cantidad de veces, y ocupa un 66,34% del tiempo del total del integrador, detallado en la tabla 5.

Cada muestra cuenta como 0,01 segundos				
% Tiempo	**Segundos Acumulados**	**Segundos Propios**	**Llamadas a función**	**Nombre**
66,34	773,40	773,40	6087504	mfo_drct_
10,92	900,66	127,26	6087500	mce_snif_
5,66	966,70	66,04	331355993	drift_dan_

Tabla 5. Extracto de la tabla 2, para el "**caso de cuerpos grandes**", detallando las funciones que mas cómputo realizan.

Al analizar esta función, se encuentra que existe una operación de división redundante, tal como se observa en la figura 16. Recordemos que esta función se ejecuta una gran cantidad de veces en el paso de simulación. Esta operación redundante no solo agrega tiempo a la ejecución sino que, además, introduce un potencial error en el resultado matemático, que podría afectar a largo plazo las ejecuciones del simulador

```
          if (s2.ge.rc2) then
              s_1 = 1.d0 / sqrt(s2)
              tmp2 = s_1 * s_1 * s_1
          else if (s2.le.0.01*rc2) then
              tmp2 = 0.d0
          else
----------------------------------------
              s_1 = 1.d0 / sqrt(s2) (1)
----------------------------------------
              s   = 1.d0 / s_1      (2)
----------------------------------------
              s_3 = s_1 * s_1 * s_1
              q = (s - 0.1d0*rc) / (0.9d0 * rc)
              q2 = q  * q
              q3 = q  * q2
              q4 = q2 * q2
              q5 = q2 * q3
              tmp2 = (10.d0*q3 - 15.d0*q4 + 6.d0*q5) * s_3
          end if
```

Figura 17. Al realizar la división registrada en (1), y luego realizar la división registrada en (2), existe una operación extra

En la figura 17, se puede ver en el código, que la operación realizada en (1), es la inversa de la raíz cuadrada de s2. Luego en la línea (2), se realiza la inversa de de la inversa de la raíz. Sin embargo, esta operación, da como resultado, que s, equivale a la raíz cuadrada de s2, insertando una división extra en el cálculo, totalmente innecesaria.

Eliminar una división redundante en una función crítica que es llamada una gran cantidad de veces durante el ciclo de integración, no solo reduce el tiempo de ejecución, sino que también reduce el error asociado a esa operación. Al eliminar el error asociado, los resultados del integrador, ya no serán los mismos que en el caso base. En la figura 18, se puede observar el resultado del reordenamiento de operaciones aplicado

```
          if (s2.ge.rc2) then
              s_1 = 1.d0 / sqrt(s2)
              tmp2 = s_1 * s_1 * s_1
          else if (s2.le.0.01*rc2) then
              tmp2 = 0.d0
          else
              s_1 = 1.d0 / sqrt(s2)
-----------------------------------------
              s = sqrt(s2)          (1)
-----------------------------------------
              s_3 = s_1 ** 3
              q = (s - 0.1d0*rc) / (0.9d0 * rc)
              q2 = q  * q
              q3 = q  * q2
              q4 = q2 * q2
              q5 = q2 * q3
              tmp2 = (10.d0*q3 - 15.d0*q4 + 6.d0*q5) * s_3
          end if
```

Figura 18. En la línea (1) se puede observar que ya **no se realiza una división extra para almacenar la variable s**.

En la figura 18, se procede a eliminar también asignaciones de variables redundantes, para simplificar aun más el cómputo realizado. El compilador no puede eliminar estas asignaciones innecesarias en forma automática, ya que las variables son utilizadas en líneas contiguas, y para el analizador estático del compilador, estas variables son necesarias, mientras que para el funcionamiento correcto del integrador, no lo son.

```
    if (s2.ge.rc2) then
      s_1 = 1.d0 / sqrt(s2)
      tmp2 = s_1 * s_1 * s_1
    else if (s2.le.0.01*rc2) then
      tmp2 = 0.d0
    else
      s_1 = 1.d0 / sqrt(s2)
      sqrt2 = sqrt(s2)
      s_3 = s_1 ** 3
-----------------------------------------
      q = (sqrt2 - 0.1d0*rc) / (0.9d0 * rc)                    (1)
      tmp2 = (10.d0*(q**3) - 15.d0*(q**4) + 6.d0*(q**5)) * s_3  (2)
-----------------------------------------
    end if
```

Figura 19. En las líneas (1) y (2) se muestra la eliminación de asignaciones redundantes, juntando todas las operaciones matemáticas en una sola línea.

En la figura 19, además se detalla, que al juntar todas las operaciones matemáticas en una sola línea, el compilador al procesar esta sentencia, puede proceder a realizar optimizaciones automáticas sobre la misma.

Finalmente en la figura 20, se trabaja sobre el condicional inicial, para eliminar un bloque completamente. Además, se elimino la última asignación redundante sobre la variable s_1, utilizando la operación de raíz inversa directamente en el cálculo, ya que esta no es reutilizada en ninguna otra sección del bloque.

```
  tmp2 = 0.d0
  if (s2.ge.rc2) then
      tmp2 = (1.d0 / sqrt(s2)) ** 3
  else if (s2.gt.0.01*rc2) then
      sqrtS2 = sqrt(s2)
      q = (sqrtS2 - 0.1d0*rc) / (0.9d0 * rc)
      tmp2 = (10.d0*(q**3) - 15.d0*(q**4) + 6.d0*(q**5)) * ( 1.d0 / sqrtS2 ) ** 3
  end if
```

Figura 20. Resultado final del reordenamiento de operaciones. El código queda en forma más concisa, eliminando tanto la asignación redundante, como el condicional extra.

Este tipo de optimización de reordenamiento de instrucciones, escapa a la capacidad de análisis estático que un compilador puede realizar en forma automática.

5 Optimizaciones Paralelas Realizadas

Desde el punto de vista paralelo, y teniendo en cuenta que los casos más utilizados por los astrónomos resultan ser los que utilizan interacciones entre cuerpos grandes, procedemos a analizar la función MFO_DRCT nuevamente, que posee un ciclo de ejecución potencialmente paralelizable, como se observa en la figura 21

```
!
      if (i0.le.0) i0 = 2
      !
      do i = i0, nbig
            do j = i + 1, nbod
                  dx = x(1,j) - x(1,i)
                  dy = x(2,j) - x(2,i)
                  dz = x(3,j) - x(3,i)
                  s2 = dx * dx  +  dy * dy  +  dz * dz
                  rc = max(rcrit(i), rcrit(j))
                  rc2 = rc * rc
                  !
                  tmp2 = 0.d0
                  if (s2.ge.rc2) then
                        tmp2 = (1.d0 / sqrt(s2)) ** 3
                  else if (s2.gt.0.01*rc2) then
                        sqrtS2 = sqrt(s2)
                        q = (sqrtS2 - 0.1d0*rc) / (0.9d0 * rc)
                        tmp2 = (10.d0*(q**3) - 15.d0*(q**4) +
6.d0*(q**5)) * ( 1.d0 / sqrtS2 ) ** 3
                  end if
                  !
                  faci = tmp2 * m(i)
                  facj = tmp2 * m(j)
                  a(1,j) = a(1,j)  -  faci * dx
                  a(2,j) = a(2,j)  -  faci * dy
                  a(3,j) = a(3,j)  -  faci * dz
                  a(1,i) = a(1,i)  +  facj * dx
                  a(2,i) = a(2,i)  +  facj * dy
                  a(3,i) = a(3,i)  +  facj * dz
            end do
      end do
```

Figura 21. Ciclo potencialmente paralelizable de MFO_DRCT. Existe una vinculación en serie entre los ciclos, que no permite su paralelización directa sin modificaciones

El problema que plantea el ciclo potencialmente paralelizable, es que el acceso a los datos del vector de cuerpos, no posee independencia de datos. Esto se produce porque cuando se está recorriendo el ciclo interno del bucle, se están modificando en forma recíproca los datos pertenecientes al cuerpo que se está comparando con el resto de los cuerpos del vector.

En la figura 22, en el paso 1, se muestra cómo funciona la interacción reciproca: todas las interacciones hacia la izquierda, modifican el cuerpo N1, al mismo tiempo que el integrador está evaluando, hacia la derecha, los datos pertenecientes al resto de cuerpos. Esto permite en ejecución secuencial, realizar una reducción en la cantidad de operaciones, ya que toda la evolución del cuerpo N1, se calcula en el primer ciclo, mientras que en el segundo ciclo, se calculan los de N2, y así sucesivamente, sin necesidad volver hacia atrás en el vector de cuerpos.

La dependencia de datos existe, porque como se observa en la indicación A, el cuerpo N2 es modificado en el paso 1 y en el paso 2. Si se quiere eliminar esta dependencia, es necesario que los datos de cada cuerpo, sean modificados en forma única por ciclo.

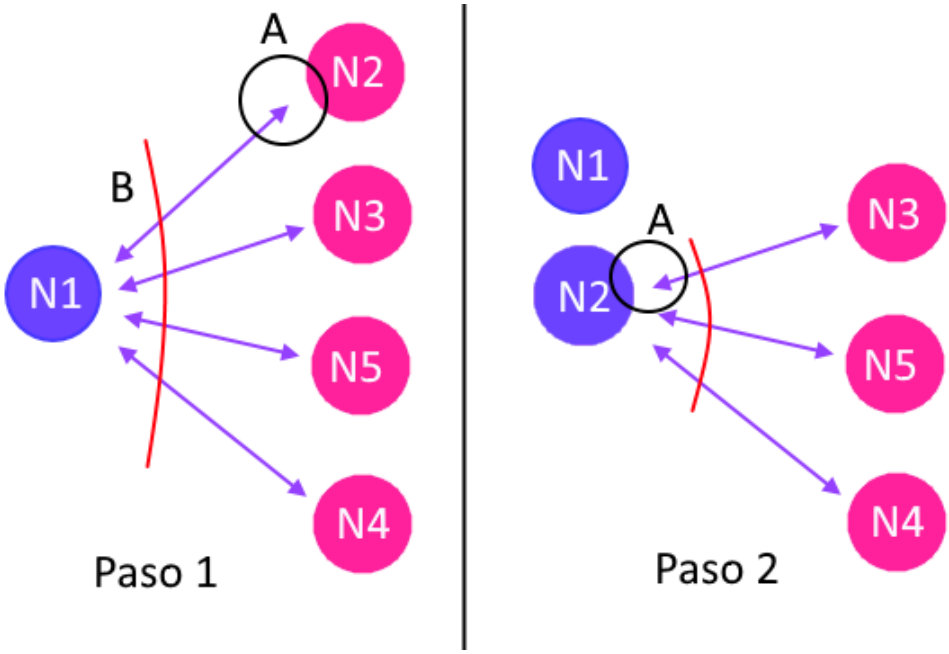

Figura 22. El ciclo original no es paralelizable por diseño.

Cuando se utiliza un solo procesador, esta dependencia de datos es beneficiosa. Pero al querer paralelizar, el ciclo debe modificarse, para evitar introducir sincronizaciones de escritura.

Lo que se realizó es una modificación en la forma de procesar los índices. Al momento de realizar esta modificación, también se dejo planteada una optimización secuencial por bloques, para permitir aprovechar el uso de memoria cache. Por el momento esta optimización, no generará ningún beneficio, ya que la cantidad de cuerpos es muy pequeña, y por cada ciclo, todos los cuerpos entran en un bloque de memoria cache.

El resultado conceptual se ve en la figura 23. Al remover la optimización secuencial que genera dependencia de datos, cada cuerpo interactúa con todos los cuerpos restantes en cada ciclo, pero solo es modificado una vez por ciclo. Esto genera una pérdida de performance en ejecuciones secuenciales, pero permite la correcta ejecución paralela, sin necesidad de introducir sincronizaciones de escritura.

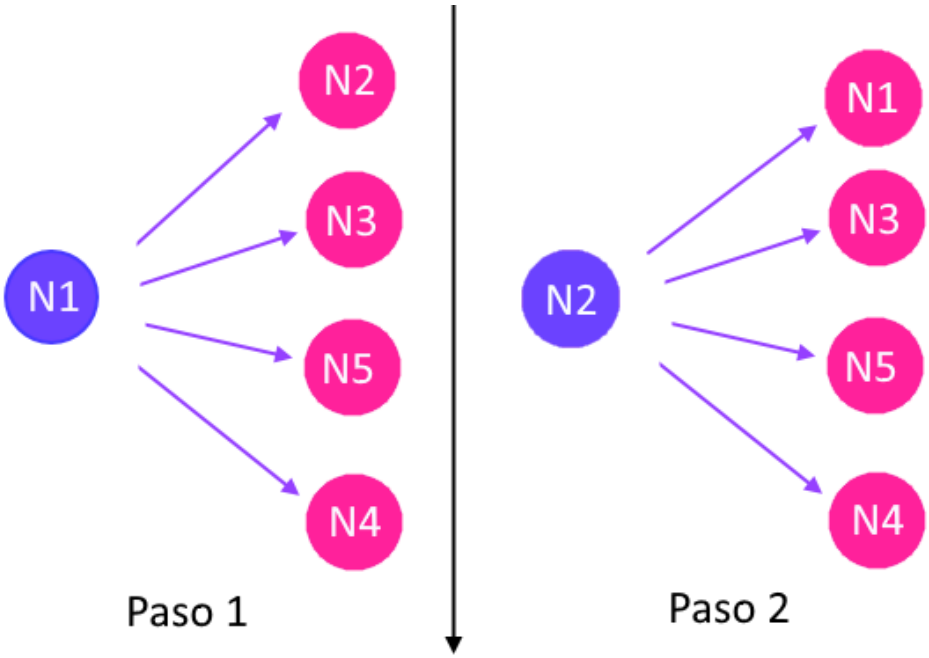

Figura 23. Ejecución del ciclo con independencia de escritura de datos

En la figura 24, se encuentra el código modificado de MFO_DRCT. Esta versión, posee independencia de datos, junto con la optimización por bloques, que se deja planteada para investigaciones futuras.

Para la optimización paralela en memoria compartida, se utiliza la biblioteca OpenMP [22], utilizando directivas al compilador para generar paralelismo. La optimización en bloques, como se mencionó, tendrá sentido, cuando en próximos trabajos, se estudie la escalabilidad del integrador optimizado, con una cantidad muy significativa de cuerpos.

```
!$omp parallel shared(m,x,a,rcrit,i0,nbod,nbig,stat)
default(private)
!(jj,i,j,dx,dy,dz,s,s_1,s2,s_3,rc,rc2,q,q2,q3,q4,q5,tmp2,faci,facj,s
qrtS2)
!$omp do
!block optimization
 do ii=i0,nbig,BLOCK_SIZE
   do jj=i0,nbod,BLOCK_SIZE

      do i=ii,min(ii+BLOCK_SIZE-1,nbig)
         do j=jj,min(jj+BLOCK_SIZE-1,nbod)
            if (j.ne.i) then
               dx = x(1,j) - x(1,i)
               dy = x(2,j) - x(2,i)
               dz = x(3,j) - x(3,i)
               s2 = dx * dx  +  dy * dy  +  dz * dz
               rc = max(rcrit(i), rcrit(j))
               rc2 = rc * rc
               tmp2 = 0.d0
               if (s2.ge.rc2) then
                  tmp2 = (1.d0 / sqrt(s2)) ** 3
               else if (s2.gt.0.01*rc2) then
                  sqrtS2 = sqrt(s2)
                  q = (sqrtS2 - 0.1d0*rc) / (0.9d0 * rc)
                  tmp2 = (10.d0*(q**3) - 15.d0*(q**4) + 6.d0*(q**5))
                         * ( 1.d0 / sqrtS2 ) ** 3
               end if
                  faci = tmp2 * m(i)
                  facj = tmp2 * m(j)
                  a(1,i) = a(1,i)  +  facj * dx
                  a(2,i) = a(2,i)  +  facj * dy
                  a(3,i) = a(3,i)  +  facj * dz
               end if
            end do
      end do

      end do
 end do
!$omp end do
!$omp end parallel
```

Figura 24. Optimización paralela utilizando OpenMP. Se deja planteada una optimización en bloques para investigaciones futuras

6 Resultados experimentales

A continuación se mostrarán los resultados experimentales de las ejecuciones de cada una de las optimizaciones propuestas las cuales fueron ejecutadas siempre en el servidor remoto de la Facultad de Informática de la Universidad Nacional de La Plata. Para las ejecuciones secuenciales se agregará también la salida de la ejecución de gprof, así se observa la evolución de las funciones a medida que se introdujeron las modificaciones

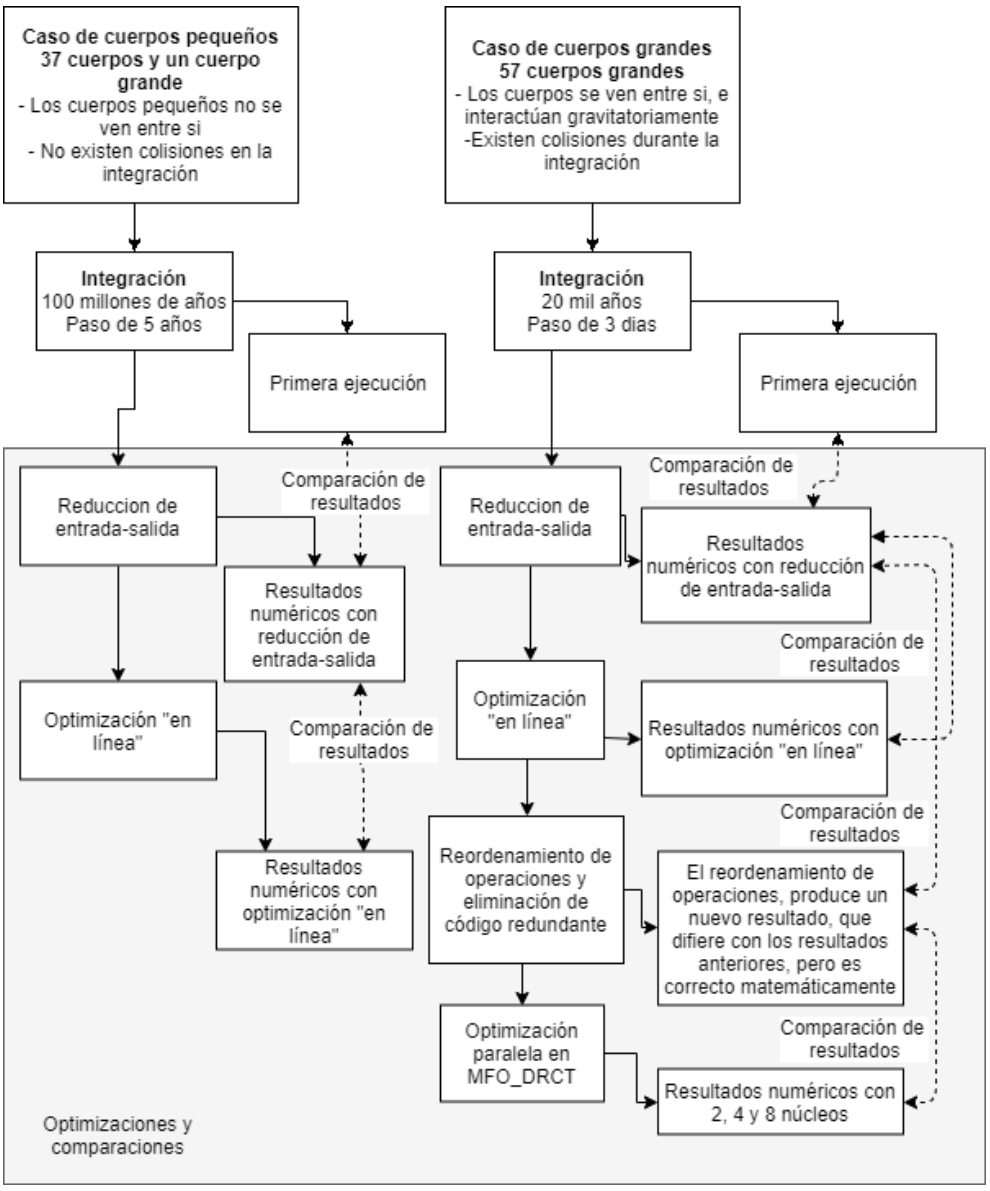

Figura 25. Metodología experimental realizada a lo largo del trabajo

En la figura 25 se detalla la metodología experimental realizada. A medida que se fue avanzando sobre optimizaciones progresivas, los resultados se compararon entre las diferentes versiones, para intentar obtener los coeficientes de mejora.

En la figura 26 se detalla la fórmula de Coeficiente de mejora utilizada para calcular la mejora de la performance para cada caso. Como lo mencionado, las técnicas de optimización son incrementales, en el sentido de que cada mejora, contiene la optimización anterior.

$$Coeficiente\ de\ mejora(C_m) = \frac{Tiempo\ de\ ejecución\ Base}{Tiempo\ de\ ejecución\ con\ Optimizaciones}$$

Figura 6. Fórmula para el cálculo del Coeficiente de mejora utilizada para el análisis de los ejemplos.

Partiendo de la base, y corroborando los resultados con los Astrónomos de la FCAGLP, se determinó que el nivel de compilación a utilizar que mantiene la coherencia en los resultados, es el nivel O2. El nivel de optimización del compilador O3, al ser un nivel que genera cambios en el código en forma más agresiva, no proporcionaba resultados coherentes.

6.1 Hardware utilizado

El hardware utilizado se detalla en la tabla 4. El mismo es el correspondiente al servidor de cómputo de altas prestaciones de la Universidad Nacional de La Plata

Modelo de procesador	CPU Intel Xeon 5405 @2.00GHz
Cantidad de procesadores	Max. \# de procesadores: 8 2 Procesadores XEON por placa madre
Fecha de lanzamiento del CPU	Q4'07

Tabla 6. Detalle del hardware utilizado para la ejecución de las pruebas

6.2 Evaluación del desempeño: Optimización del compilador -O2

A continuación evaluaremos los dos casos planteados por el grupo de investigación de ciencias planetarias de la FCAGLP. El primer caso planteado, es la utilización del flag de optimización –O2, que es la suma de todas las optimizaciones mostradas en la figura 27 [23].

```
-fauto-inc-dec                  -falign-functions  -falign-jumps
-fbranch-count-reg              -falign-labels  -falign-loops
-fcombine-stack-adjustments     -fcaller-saves
-fcompare-elim                  -fcode-hoisting
-fcprop-registers               -fcrossjumping
-fdce                           -fcse-follow-jumps  -fcse-skip-blocks
-fdefer-pop                     -fdelete-null-pointer-checks
-fdelayed-branch                -fdevirtualize  -fdevirtualize-speculatively
-fdse                           -fexpensive-optimizations
-fforward-propagate             -ffinite-loops
-fguess-branch-probability      -fgcse  -fgcse-lm
-fif-conversion                 -fhoist-adjacent-loads
-fif-conversion2                -finline-functions
-finline-functions-called-once  -finline-small-functions
-fipa-profile                   -findirect-inlining
-fipa-pure-const                -fipa-bit-cp  -fipa-cp  -fipa-icf
-fipa-reference                 -fipa-ra  -fipa-sra  -fipa-vrp
-fipa-reference-addressable     -fisolate-erroneous-paths-dereference
-fmerge-constants               -flra-remat
-fmove-loop-invariants          -foptimize-sibling-calls
-fomit-frame-pointer            -foptimize-strlen
-freorder-blocks                -fpartial-inlining
-fshrink-wrap                   -fpeephole2
-fshrink-wrap-separate          -freorder-blocks-algorithm=stc
-fsplit-wide-types              -freorder-blocks-and-partition  -freorder-functions
-fssa-backprop                  -frerun-cse-after-loop
-fssa-phiopt                    -fschedule-insns  -fschedule-insns2
-ftree-bit-ccp                  -fsched-interblock  -fsched-spec
-ftree-ccp                      -fstore-merging
-ftree-ch                       -fstrict-aliasing
-ftree-coalesce-vars            -fthread-jumps
-ftree-copy-prop                -ftree-builtin-call-dce
-ftree-dce                      -ftree-pre
-ftree-dominator-opts           -ftree-switch-conversion  -ftree-tail-merge
-ftree-dse                      -ftree-vrp
-ftree-forwprop
-ftree-fre
-ftree-phiprop
-ftree-pta
-ftree-scev-cprop
-ftree-sink
-ftree-slsr
-ftree-sra
-ftree-ter
-funit-at-a-time
```

Figura 27. Detalle de las optimizaciones activadas por la opción -O2

6.2.1 Evaluación del caso de cuerpos pequeños: Opción –O2

En la tabla 5 se ven los resultados de ejecución con solo utilizar la opción -O2 del compilador gfortran. En la tabla 7, se muestran los resultados del "wall clock" para la misma ejecución, junto con el coeficiente de mejora obtenido.

Cada muestra cuenta como 0,01 segundos				
% Tiempo	**Segundos Acumulados**	**Segundos Propios**	**Llamadas a Función**	**Nombre**
30,91	82,59	82,59	470004100	drift_dan_
15,82	124,87	42,28	395693882	drift_kepmd_
6,87	143,21	18,34	74310218	drift_kepu_new_
6,58	160,78	17,58	124183696	mfo_hkce_
5,16	174,57	13,79	20000025	mfo_drct_
4,71	187,15	12,58	20000000	mdt_hy_

Tabla 7. Detalle del perfilador gprof para la ejecución del caso de cuerpos pequeños sin colisión (3-Cuerpos restringido), utilizando el nivel de optimización del compilador O2

Tiempo para O2 con E/S: Cuerpos pequeños	2398,093 segundos
Coeficiente de mejora obtenido	1,067453

Tabla 8. Tiempos y coeficiente de mejora obtenido por utilización del nivel O2 de compilación para cuerpos pequeños

Aun al utilizar el nivel de optimización –O2, vemos que la mejora no es significativa. Esto fue un indicio, para detectar, que en realidad el problema de retraso medido en el tiempo de usuario de la tabla 8, junto con un muy bajo coeficiente de mejora, está siendo causado por factores externos al cómputo del integrador.

En el caso de cuerpos pequeños, se realiza una integración por una gran cantidad de años, con lo cual para cada paso de de la misma, se está generando un evento de escritura por consola. No solo esto, sino que por la configuración establecida, se realiza un volcado de datos a disco para almacenamiento temporal de resultados, cada cierta cantidad de pasos.

Una reducción significativa en la entrada-salida, permitirá comprobar si esta hipótesis es cierta. En contraste, esto no es lo mismo que ocurre en el caso de cuerpos grandes.

6.2.2 Evaluación del caso de cuerpos grandes: Opción –O2

En la tabla 9, se muestra la ejecución con la optimización O2, para el caso de cuerpos grandes. En la tabla 10, podemos ver los resultados del "wall clock" obtenidos.

Cada muestra cuenta como 0,01 segundos				
% Tiempo	**Segundos Acumulados**	**Segundos Propios**	**Llamadas a Función**	**Nombre**
54,55	282,86	282,86	6087504	mfo_drct_
16,48	368,29	85,43	6087500	mce_snif_
11,89	429,97	61,68	343348444	drift_dan_
6,77	465,07	35,10	343348444	drift_kepmd_
1,80	474,41	9,34	8689583	mce_box_
1,77	483,59	9,18	56644365	mfo_hkce_

Tabla 9. Detalle del perfilador gprof para la ejecución del caso de cuerpos grandes con colisión (N-Cuerpos completo), utilizando el nivel de optimización del compilador O2

Tiempo para O2 con E/S: Cuerpos grandes	591,272 segundos
Coeficiente de mejora obtenido	2,089

Tabla 10. Tiempos y coeficiente de mejora obtenido por utilización del nivel O2 de compilación para cuerpos grandes

El coeficiente de mejora en el caso de cuerpos grandes, como se muestra en la tabla 10, indica un aumento de más del doble en el rendimiento del integrador. Recordando, que en el caso de cuerpos grandes, como se menciono anteriormente, el tiempo configurado para la integración es significativamente menor, y que la entrada-salida del integrador está estrechamente relacionada con este tiempo configurado, es esperable que el caso de cuerpos grandes se vea afectado en mucha menor medida por los efectos de la entrada-salida.

La optimización –O2, aumenta al doble el rendimiento del caso de cuerpos grandes. Resta analizar los efectos de la reducción de entrada-salida en el integrador, y esto se estudiará en la sección siguiente

6.3 Evaluación del desempeño: Reducción de Entrada-Salida

A continuación se detallan los resultados obtenidos de la reducción de entrada-salida para ambos casos

6.3.1 Caso de cuerpos pequeños: Reducción de entrada-salida

Los resultados de ejecutar el integrador, reduciendo la entrada-salida ya sea por consola y por accesos a disco para almacenar resultados temporales, se observan en la tabla 11.

Cada muestra cuenta como 0,01 segundos				
% Tiempo	**Segundos Acumulados**	**Segundos Propios**	**Llamadas a Función**	**Nombre**
33,88	44,00	44,00	470004100	drift_dan_
12,61	60,37	16,38	395693882	drift_kepmd_
7,01	69,47	9,10	74310218	drift_kepu_new_
5,75	76,94	7,47	124183696	mfo_hkce_
5,61	84,22	7,28	20000000	mdt_hy_
5,42	91,26	7,04	24258838	mce_box_

Tabla 11. Detalle del perfilador gprof para la ejecución del caso de cuerpos pequeños sin colisión (3-Cuerpos restringido), sin generación de entrada/salida por consola

En la tabla 11, se ve que los cambios porcentuales de cómputo de las funciones, no se vieron afectados en gran medida por la reducción de entrada-salida. Sin embargo, al observar el tiempo medido con el "wall clock" en la tabla 12, se ve un incremento de 12 veces en el rendimiento percibido por el usuario.

Tiempo sin E/S: Cuerpos pequeños	188,074 segundos
Coeficiente de mejora obtenido	12,75079

Tabla 12. Tiempos y coeficiente de mejora obtenido por utilizar reducción de E/S para cuerpos pequeños

Este incremento en el rendimiento confirma la hipótesis, de que el problema de tiempos observado en el caso de cuerpos pequeños, está relacionado con un problema externo al cómputo del integrador. En este caso, se estaba realizando una excesiva entrada-salida, y al reducirla, el salto de rendimiento es considerable.

6.3.2 Caso de cuerpos grandes: Reducción de entrada-salida

En el caso de cuerpos grandes, dado que el tiempo configurado de integración y la cantidad de pasos de integración es menor que en el caso anterior, este se ve poco afectado por la entrada-salida. Es por esta razón, que al reducirla, se puede ver que las diferencias son mínimas respecto a simplemente utilizar la opción –O2.

De nuevo se observa que los tiempos porcentuales mostrados en la tabla 13, se mantienen similares a la ejecución con entrada-salida, mientras que el tiempo de la tabla 14, correspondiente al medido por el "wall clock", se mantiene similar al observado con entrada-salida.

Cada muestra cuenta como 0,01 segundos				
% Tiempo	**Segundos Acumulados**	**Segundos Propios**	**Llamadas a Función**	**Nombre**
54,85	284,46	284,46	6087504	mfo_drct_
15,90	366,94	82,48	6087500	mce_snif_
11,49	426,52	59,58	343348444	drift_dan_
7,08	463,26	36,73	343348444	drift_kepmd_
1,89	473,05	9,79	56644365	mfo_hkce_
1,77	482,23	9,18	6087500	mdt_hy_

Tabla 13. Detalle del perfilador gprof para la ejecución del caso de cuerpos grandes con colisión (N-Cuerpos completo), sin generación de entrada/salida por consola

Tiempo para O2 sin E/S: Cuerpos grandes	593,586 segundos
Coeficiente de mejora obtenido	2,08

Tabla 14. Tiempos y coeficiente de mejora obtenido por utilizar reducción de E/S para cuerpos grandes

No obstante estos resultados, se utilizará la versión de entrada-salida reducida en las ejecuciones siguientes para ambos casos, debido a que el estudio de la entrada-salida en profundidad escapa al análisis de computo de alto rendimiento para este caso

6.4 Evaluación del desempeño: Optimización secuencial "en línea" (inline)

A continuación evaluaremos los resultados de realizar la optimización en línea para cada caso. Cabe aclarar, que los coeficientes de mejora, a partir de este punto, serán tomados respecto del mejor caso sin entrada-salida, para evaluar correctamente mejoras de cómputo, excluyendo los factores externos.

6.4.1 Caso de cuerpos pequeños: optimización "en línea"

Como se pudo observar anteriormente, la cantidad de llamadas realizadas a las funciones DRIFT_DAN y DRIFT_KEPMD es significativa. En un intento de reducir la sobrecarga de estas llamadas se aplico la optimización "en línea", y se estudió el comportamiento del código mostrado en la tabla 15, donde se ve que la cantidad de llamadas a estas dos funciones se reduce en la llamada a DRIFT_ONE.

Cada muestra cuenta como 0,01 segundos				
% Tiempo	**Segundos Acumulados**	**Segundos Propios**	**Llamadas a Función**	**Nombre**
49,41	66,44	66,44	470004100	drift_one_
7,39	76,38	9,94	74310218	drift_kepu_new_
5,74	84,10	7,72	124183696	mfo_hkce_
4,89	90,68	6,58	20000000	mdt_hy_
4,83	97,17	6,49	24258838	mce_box_
4,77	103,58	6,41	20000025	mfo_drct_

Tabla 15. Detalle del perfilador gprof para la ejecución del caso de cuerpos pequeños sin colisión (3-Cuerpos restringido), con la conversión de DRIFT_DAN y DRIFT_KEPMD en formato "en línea" dentro de DRIFT_ONE

Tiempo para optimización "En línea": Cuerpos pequeños	189,258 segundos
Coeficiente de mejora obtenido	1,00

Tabla 16. Tiempos y coeficiente de mejora obtenido, respecto de la ejecución sin entrada-salida, con la conversión de DRIFT_DAN y DRIFT_KEPMD en formato "en línea" dentro de DRIFT_ONE para cuerpos pequeños

En principio, en la tabla 16, se puede ver que la mejora no introduce un aumento de velocidad para los casos de cuerpos pequeños. En el caso de cuerpos pequeños, no tenemos más propuestas de optimización a realizar, y a partir de ahora, nos enfocaremos en las mejoras para el caso de cuerpos grandes.

6.4.2 Caso de cuerpos grandes: optimización "en línea"

El resultado de la ejecución del integrador para cuerpos grandes con la optimización "en línea", se observa en la tabla 17. En la misma se observa que la función DRIFT_ONE ahora ocupa el 20% de la ejecución, eliminando las llamadas redundantes de DRIFT_DAN y DRIFT_KEPMD.

Cada muestra cuenta como 0,01 segundos				
% Tiempo	**Segundos Acumulados**	**Segundos Propios**	**Llamadas a Función**	**Nombre**
54,03	283,31	283,31	6087504	mfo_drct_
20,16	389,01	105,70	343348444	drift_one_
16,09	473,39	84,38	6087500	mce_snif_
1,93	483,53	10,14	56644365	mfo_hkce_
1,67	492,28	8,75	6087500	mdt_hy_
1,66	501,00	8,72	8689583	mce_box_

Tabla 17. Detalle del perfilador gprof para la ejecución del caso de cuerpos grandes con colisión (N-Cuerpos completo), con la conversión de DRIFT_DAN y DRIFT_KEPMD en formato "en línea" dentro de DRIFT_ONE

Tiempo para optimización "En línea": Cuerpos Grandes	582,65198 segundos
Coeficiente de mejora obtenido	1,02

Tabla 18. Tiempos y coeficiente de mejora obtenido con la conversión de DRIFT_DAN y DRIFT_KEPMD en formato "en línea" dentro de DRIFT_ONE para cuerpos grandes

Al integrar esa optimización en el caso de cuerpos grandes, se detecta una pequeña mejora en el tiempo de ejecución, donde el aumento del coeficiente de mejora obtenido entre este caso y el anterior mostrado en la tabla 18, es de un 2%.

Aunque este número en valor absoluto es muy pequeño, resulta interesante notar que un 2% de la ejecución del programa, se estaba malgastando en llamadas a función que se podían simplificar. No obstante, al realizar esta modificación, también se eliminaron sentencias GOTO, y esto resulta en una mejora en la lectura del código a futuro.

6.5 Evaluación del desempeño: Eliminación y reordenamiento de operaciones en MFO_DRCT

A continuación se estudiaran los resultados de ejecutar el integrador para el caso de cuerpos grandes, realizando un reordenamiento de operaciones y una eliminación de redundancias. De ahora en adelante, como se mencionó, las optimizaciones solo tendrán sentido para el caso de cuerpos grandes, porque se hacen sobre la función MFO_DRCT, que es mayormente utilizada en este caso.

6.5.1 Caso de cuerpos grandes: Eliminación y reordenamiento de operaciones

En la tabla 19 se muestran los resultados de gprof para el caso de cuerpos grandes con eliminación y reordenamiento de operaciones. Al realizar una eliminación de operaciones, el resultado obtenido por el integrador difiere del caso base, por eso las relaciones porcentuales de computo en las funciones, difiere de los casos mostrados anteriormente para cuerpos grandes.

Estos resultados fueron validados por el grupo de ciencias planetarias, para garantizar que son correctos, independientemente de que los cambios realizados, matemáticamente conservan su significado conceptual.

Cada muestra cuenta como 0,01 segundos				
% Tiempo	**Segundos Acumulados**	**Segundos Propios**	**Llamadas a Función**	**Nombre**
54,80	289,91	289,91	6087501	mfo_drct_
19,75	394,38	104,47	346987500	drift_one_
15,91	478,56	84,18	6087500	mce_snif_
1,83	488,26	9,70	6087500	mdt_hy_
1,75	497,53	9,27	8496935	mce_box_
1,62	506,09	8,56	6087500	mce_cent_

Tabla 19. Detalle del perfilador gprof para la ejecución del caso de cuerpos grandes con colisión (N-Cuerpos completo), con la conversión de DRIFT_DAN y DRIFT_KEPMD en formato "en línea" dentro de DRIFT_ONE, agregando la reducción de operaciones para MFO_DRCT

Al eliminar una división, lo que ocurre aquí es que el camino de cómputo ya no es el mismo que previamente, y el coeficiente de mejora no muestra cambios significativos, el caso no es exactamente el mismo, y la comparación se vuelve más compleja.

Tiempo para optimización "En línea": Cuerpos Grandes	587,988 segundos
Coeficiente de mejora obtenido	1,01

Tabla 20. Tiempos y coeficiente de mejora obtenido con la conversión de DRIFT_DAN y DRIFT_KEPMD en formato "en línea" dentro de DRIFT_ONE para cuerpos grandes, agregando la reducción de operaciones para MFO_DRCT

En este caso particular, se observa en la tabla 20 un incremento en el tiempo respecto de la optimización secuencial anterior "en línea", aumenta de unos 582 segundos, a 587 segundos por integración. El hecho de modificar el integrador, eliminando una división, reduce también el error asociado a esta operación.

Al tener un error menor, el resultado final obtenido por el integrador, difiere del que se generaba anteriormente, con lo cual se modifica el camino de cómputo, y el caso se convierte en un caso completamente nuevo. Es por esta razón, que el coeficiente de mejora en paralelo, deberá tomarse respecto de la ejecución de este caso, ya que consiste en un nuevo cómputo respecto de los anteriores.

6.6 Evaluación del desempeño: Optimización paralela

A continuación se estudiara la ejecución del integrador para el caso de cuerpos grandes. Los coeficientes de mejora, se tomarán respecto del caso anterior, por tratarse de un nuevo caso de cómputo al realizar la optimización de eliminación de operaciones redundantes.

6.6.1 Caso de cuerpos grandes: Optimización paralela

Lamentablemente, los resultados de gprof para evaluar programas paralelos, no resultan útiles, ya que solo mide los valores dentro de un solo procesador. Para mostrar la escalabilidad de la solución, se procede a analizar el resultado de utilizar 2, 4 y 8 núcleos, detallado en las tablas 21, 22 y 23 sucesivamente.

Tiempo de ejecución paraleliza con 2 núcleos	585,281 segundos
Coeficiente de mejora obtenido	1,004

Tabla 21. Tiempos y coeficiente de mejora obtenido al eliminar la optimización secuencial que forzaba una dependencia de datos entre ciclos, para 2 núcleos.

Tiempo de ejecución paraleliza con 4 núcleos	433,939 segundos
Coeficiente de mejora obtenido	1,355

Tabla 22. Tiempos y coeficiente de mejora obtenido al eliminar la optimización secuencial que forzaba una dependencia de datos entre ciclos, para 4 núcleos.

Tiempo de ejecución paraleliza con 8 núcleos	381,284 segundos
Coeficiente de mejora obtenido	1,54

Tabla 23. Tiempos y coeficiente de mejora obtenido al eliminar la optimización secuencial que forzaba una dependencia de datos entre ciclos, para 8 núcleos.

Eliminando la dependencia de datos del ciclo interno de MFO_DRCT, se puede lograr paralelismo en el ciclo, utilizando memoria compartida. Sin embargo, al eliminar la dependencia de datos, se elimina también la optimización secuencial.

Es por eso, que a menor cantidad de núcleos, el resultado final obtenido es próximo al valor secuencial previo. Pero cuando la cantidad de núcleos aumenta linealmente, el coeficiente de mejora también lo hace.

En el mejor caso, con 8 núcleos, cuando se realiza la comparación respecto de los tiempos obtenidos para la versión secuencial base, nos da un 54% de mejora en los tiempos. Este es el mejor caso obtenido para el caso de cuerpos grandes hasta el momento, y representa el máximo logrado en este trabajo, al incluir todas las optimizaciones planteadas en este trabajo.

De esta manera, se termina generando que se ejecuten más operaciones en forma secuencial. Pero al realizar esas operaciones en 8 procesadores en paralelo, se reduce el tiempo total de ejecución, obteniendo el mejor caso para cuerpos grandes.

6.7 Resultados de tiempos y coeficientes de mejora agrupados por caso

A continuación se detallan en forma gráfica los resultados relevantes de cada caso.

6.7.1 Resultados relevantes para el caso de cuerpos pequeños

En la figura 28, se ven los tiempos entre el caso con entrada-salida, y el caso donde se elimina la misma. Se agruparon los resultados de todos los tiempos observados en las distintas ejecuciones para el caso de cuerpos pequeños.

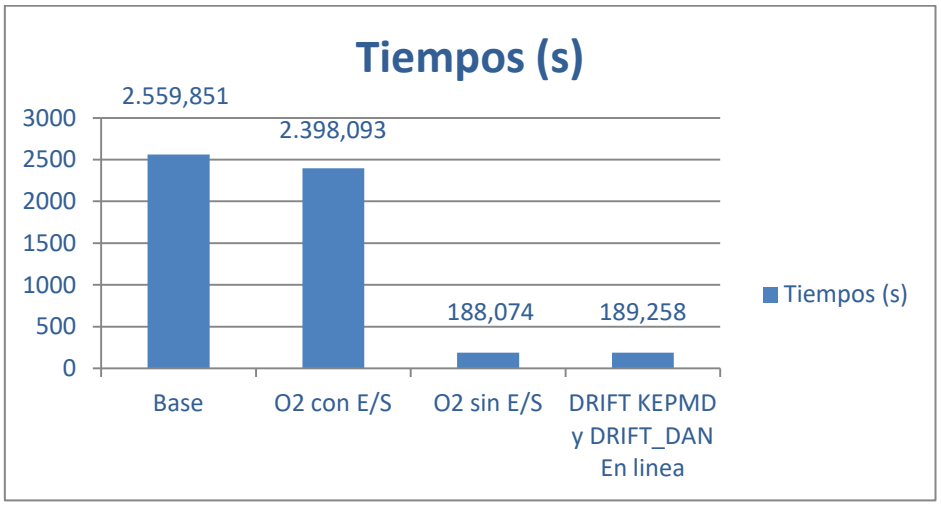

Figura 28. Tiempos para el caso de cuerpos pequeños por cada ejecución

En la figura 29, se muestra la mejora significativa en forma porcentual, a través del coeficiente de mejora. Se excluyeron las comparativas respecto de las mejoras en optimizaciones secuenciales "en línea", en el caso de cuerpos pequeños, por observarse poca diferencia notable entre las mismas.

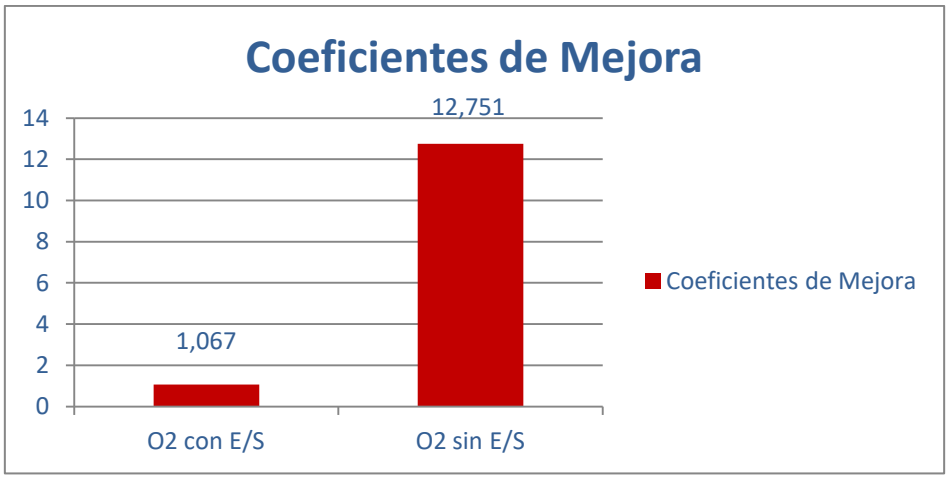

Figura 29. Mejora relativa al eliminar entrada-salida en el caso de cuerpos pequeños

6.7.2 Resultados relevantes para el caso de cuerpos grandes

En el caso de cuerpos grandes, las comparativas se realizan en dos etapas. En la primera etapa, estudiaremos las optimizaciones secuenciales excluyendo el reordenamiento y eliminación de operaciones, debido a que el resultado numérico en este caso difiere del caso base ejecutado.

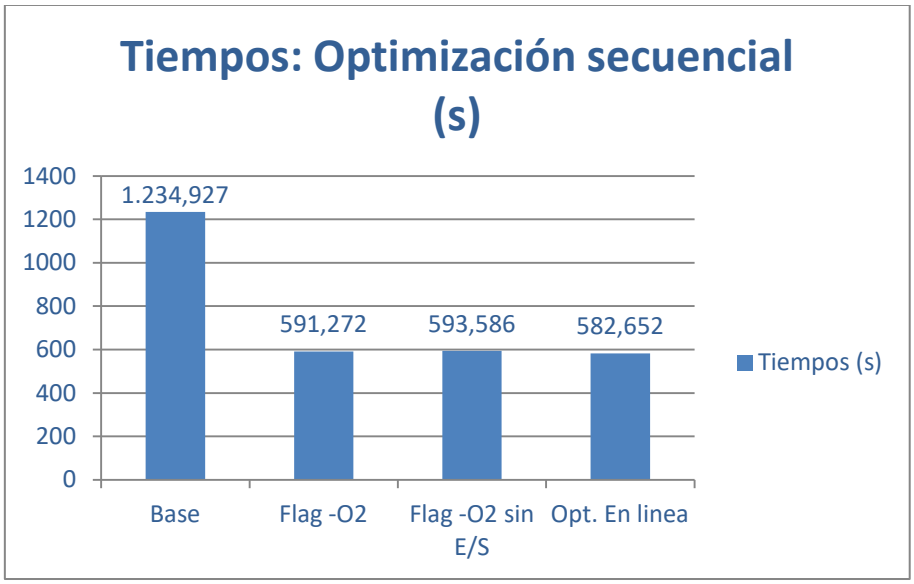

Figura 30. Tiempos medidos para optimizaciones secuenciales en el caso de cuerpos grandes.

En la figura 30, podemos observar los diferentes tiempos medidos para el caso de cuerpos grandes. A diferencia del caso de cuerpos pequeños, el caso de cuerpos grandes se ve beneficiado en gran medida por la utilización del flag –O2, mientras que la eliminación de entrada-salida, por la naturaleza del integrador en este caso, como explicado anteriormente, no provee mejoras.

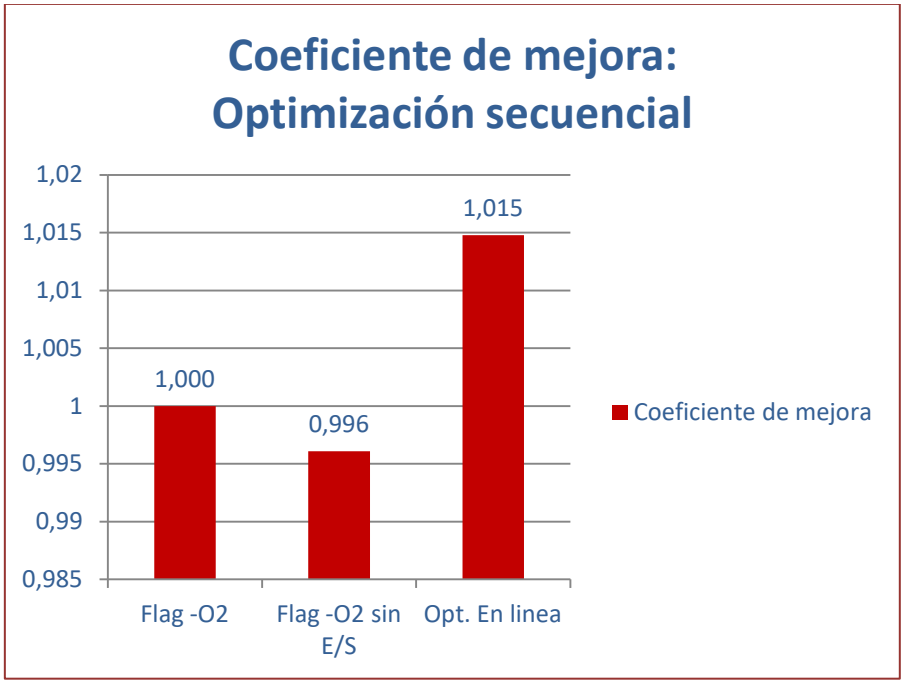

Figura 31. Tiempos medidos para optimizaciones secuenciales en el caso de cuerpos grandes, relativos al caso de compilación con –O2.

En la figura 31, se encuentran detalladas las mejoras porcentuales a través del coeficiente de mejora, puestos en referencia de la optimización al utilizar la opción de compilación –O2. En el caso de cuerpos grandes, como se puede ver, la optimización en línea provee una pequeña mejora porcentual.

Para las ejecuciones paralelas, se toma como base, el caso de optimización secuencial utilizando el reordenamiento de operaciones, ya que produce un resultado numérico diferente a los anteriores, y tenemos que considerarlo como una variación del caso de estudio. En la figura 32 se muestran los tiempos medidos para cada caso de ejecución.

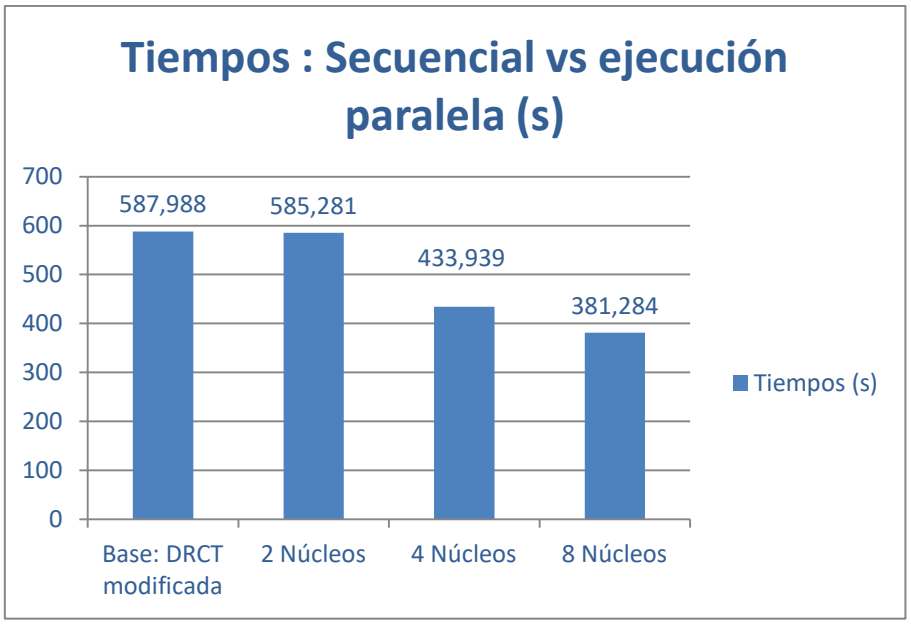

Figura 32. Tiempos para el caso de cuerpos grandes de ejecución secuencial versus ejecución paralela

A medida que se utilizan más núcleos, como se observa en la figura 32, los tiempos se reducen notablemente. Los coeficientes de mejora, para estos casos, se muestran en la figura 33, aumentando el rendimiento en más de un 50% en el caso de 8 núcleos en paralelo.

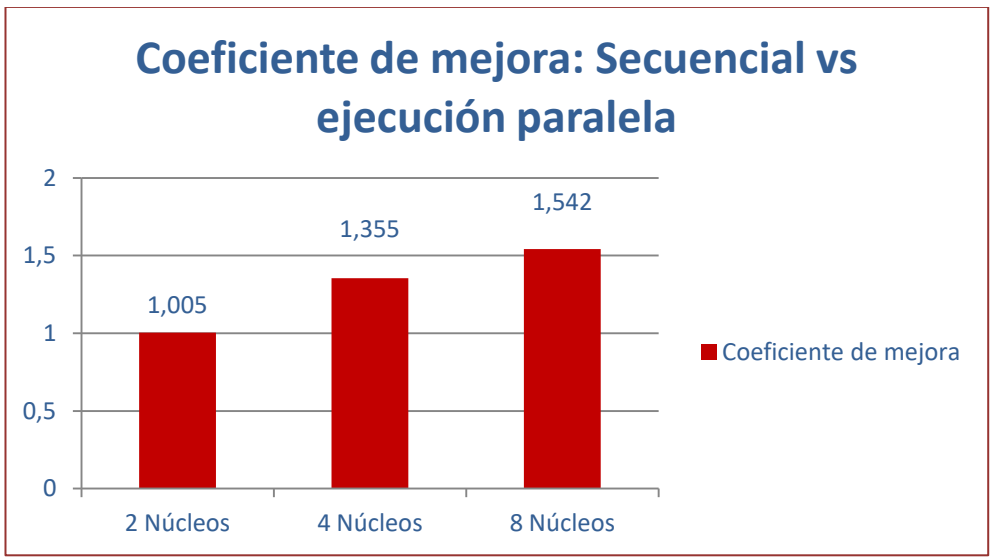

Figura 33. Coeficientes de mejora obtenidos para el caso de cuerpos grandes de ejecución secuencial versus ejecución paralela

7 Conclusiones

En el presente estudio, se evaluaron posibilidades de optimización junto con sus respectivas implementaciones para la obtención de un código con mejor rendimiento en el integrador Mercury. Aún hoy en día, Mercury es ampliamente utilizado por astrónomos en diversas partes del mundo, por lo tanto el aporte en optimización presentado, puede resultar beneficioso para la investigación de científicos especializados en ciencias planetarias.

El desarrollo de este trabajo, nos permitió la colaboración interdisciplinaria entre astrónomos y científicos computacionales, estableciendo un canal de comunicación entre estos ámbitos, donde se obtuvieron resultados beneficiosos para ambas partes, en el intercambio de conocimientos con un mismo objetivo.

Durante la implementación de las optimizaciones propuestas, nos encontramos con una optimización secuencial en la versión original del integrador, que generó una dependencia de datos. Esta optimización se tuvo que que eliminar, para poder aplicar la optimización paralela.

Esto es un indicio de que el paralelismo no siempre se beneficia de cualquier optimización. En ciertas ocasiones, se requiere una re-ingeniería de código para lograr mejores resultados

Se demostró la posibilidad de paralelizar un algoritmo escrito en Fortran y muy utilizado en la comunidad científica, junto con un procedimiento de análisis claro, y replicable, el cual se recomienda a la hora de buscar optimizar cualquier tipo de algoritmo, ya sea de astronomía, o de cualquier rama científica.

Se plantean entre otras perspectivas a futuro con este programa, las siguientes líneas de investigación

1. **Estudio de granularidad**

Es de utilidad conocer el límite de cuerpos en el cual conviene utilizar procesamiento paralelo, versus procesamiento secuencial. Esto permitiría conocer aun más las capacidades de las optimizaciones presentadas, en casos mucho más complejos, donde las integraciones en lugar de durar minutos, pueden durar meses.

2. **Estudios de casos mixtos entre cuerpos pequeños y grandes**

En el ámbito de uso del simulador, un caso frecuente es el de cuerpos pequeños y grandes coexistiendo en una misma simulación, donde se cumplen todas las características mencionadas de ambos casos: los cuerpos pequeños no se ven entre sí, pero ven a todos los cuerpos grandes, y los cuerpos grandes ven a

todos los demás cuerpos interactuantes. Los problemas numéricos resultantes requieren una participación más cercana del grupo de investigación de la FCAGLP para evaluar los resultados en estos casos, porque las complicaciones por errores numéricos crecen rápidamente

3. Análisis de Mercury utilizando contadores de hardware

Aunque se estableció que el caso de optimización de reducción de instrucciones produce un cómputo levemente diferente respecto del caso original, no se sabe exactamente aun cuanta diferencia existe entre ambos casos. La forma de establecer esta diferencia de forma empírica y determinística, es utilizando contadores de hardware, y medir exhaustivamente la ejecución de ambos casos.

Una vez determinada la diferencia de cómputo, se pueden establecer índices de ajuste para determinar si existen casos donde conviene utilizar soluciones secuenciales, o si ya definitivamente conviene utilizar la solución paralela para todo.

La interacción interdisciplinaria es una de las aéreas donde la aplicación de la ciencia computacional ayuda al avance de la construcción del camino científico futuro. Las modificaciones realizadas a Mercury en este trabajo, se pondrán a disposición del departamento de ciencias planetarias de la FCAGLP para continuar con el intercambio de conocimiento establecido durante este trabajo.

8 Referencias bibliográficas

[1] [2] Levison, H.; Duncan, M. ,(Ultima revisión: 11/29/2000 por Levison, H.), "*SWIFT*", http://www.boulder.swri.edu/~hal/swift.html
(Accedido por última vez: 29/03/2019)

[3] [19] Chambers, J.E; Migliorini, F., "*Mercury - A New Software Package for Orbital Integrations"*, Bull. American Astron. Soc. (1997), 29, 1024.

[4] Hellmich, S.; Mottola, S.; Hahn, G.,"*cuSwift - a suite of numerical integration methods for modelling planetary systems implemented in C/CUDA"*, Asteroids, Comets, Meteors Book of Abstracts (2014), Helsinki, Finland.

[5] [8] Aarseth, S. J. ,*"Gravitational N-body Simulations, Tools and Algorithms"*, Cambridge University Press (2003).

[6] Imagen extraída de InsideHPC, https://insidehpc.com/2015/05/direct-n-body-simulation/
(accedido el 29/03/2019)

[7] Hairer, E.; Lubich, C.; Wanner, G., "*Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*" (2006), (2nd Ed.). Springer.

[9] [10] Chambers, J.E.; M.A.Murison, *"Pseudo-High-Order Symplectic Integrators"* , Astronomical Journal (2000), 119, 425-433.

[11] Chambers, J.E., "*A Symplectic Integration Scheme that Allows Close Encounters between Massive Bodies*", IAU Colloquium 172, The Impact of Modern Dynamics in Astronomy (1999), 449-450.

[12] Grimm, S. L.;Stadel, J.G., "*The GENGA Code: Gravitational Encounters in N-body simulations with GPU Acceleration*", arXiv:1404.2324 [astro-ph.EP] (2014)

[13] Scott, K., *"On Proebsting's Law, Department of Computer Science"*, Technical Report CS-2001-12 (2001), University of Virginia

[14] Advanced Micro Devices, http://www.amd.com/
(accedido el 29/03/2019)

[15] INTEL corporation, http://www.intel.com/
(accedido el 29/03/2019)

[16] ARM Holdings, http://www.arm.com/
(accedido el 29/03/2019)

[17] NVIDIA Corporation, http://www.nvidia.com/
(accedido el 29/03/2019)

[18] GPU-BSM: a GPU-based tool to map bisulfite-treated reads, https://www.researchgate.net/figure/Multi-core-and-many-core-processors-Multi-core-processors-as-CPUs-are-devices-composed_fig22_262536613
(accedido el 29/03/2019)

[20] Jay Fenlason, *"GNU gprof manual"*, http://sourceware.org/binutils/docs/gprof/index.html
(accedido el 29/03/2019)

[21] Dijkstra, Edsger W., *"Letters to the editor: Go To Statement Considered Harmful"* En: Commun. ACM 11 (1968), Nr. 3, 147–148.

[22] OpenMP Architecture Review Board., *"OpenMP Application Programming Interface", Version 5.0, (Nov. 2018)*

[23] GNU GCC Compiler, https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html
(accedido el 29/03/2019)