Universidad de **Valladolid**
Facultad de Ciencias

TRABAJO FIN DE GRADO

Grado en Física

Implementación de un algoritmo de solución de las ecuaciones de movimiento en dinámica molecular clásica en Unidades de Procesado Gráfico (GPU)

Autor: Borja García Marín

Tutor/es: Marco Antonio Gigosos Pérez

Agradecimientos

Quiero aprovechar la ocasión para agradecer en especial el apoyo y constancia de mi tutor, Marco Antonio Gigosos, tanto de prácticas de empresa como del trabajo de fin de grado, en concreto la ayuda que me ha proporcionado para completar ambos proyectos, como la facilidad a acceder a ambos.

Además, no podría haber llegado a realizar el trabajo de fin de grado, sin todo lo que me han enseñado y aportado otros profesores de la misma carrera en Física.

Por otro lado, radicalmente opuesto, me gustaría destacar el apoyo en los campos tanto académicos como externos, pero sin los cuáles no podría haber realizado mi Grado en Física, a mis compañeros y a mi familia.

Resumen

Este trabajo de fin de grado trata de utilizar las ecuaciones electromagnéticas correspondientes para simular la situación física correspondiente a un plasma de partículas cargadas que interaccionan entre sí. Estos programas ya existen, sin embargo, los existentes hasta ahora cuentan con una limitación procedente del hardware en la cual los bloques de la GPU no pueden sincronizarse de forma eficiente. Sin embargo, esto se soluciona en 2017 mediante la introducción en el mercado de una nueva arquitectura de GPUs. Buscaremos crear un software que aproveche las nuevas metodologías de sincronismo para que se permita hacer simulaciones con un mayor número de partículas. Se comentarán los detalles y dificultades que añade el software necesario, CUDA 9.0 o superior, con respecto a los anteriores.

Palabras Clave: CUDA, Sincronismo entre Bloques, Simulación, Plasma

Abstract

This final year dissertation will consist on using the electromagnetic equations which take part on a plasma compound of charged particles, and their way of interact between each other. Although programs like we are studying already exist, they have a hardware limitation, the GPU's block synchronization wasn't possible on an efective way in older devices, but since 2017 it is already solved with the introduction of new GPU's architecture. This fact allows the simulations to be able to increase significantly the number of particles included. We will comment the details and difficulties which the new software adds concerning older ones.

Keywords: CUDA, Blocks Synchronization, Simulation, Plasm

Contents

1 Introducción

La física computacional es una técnica que ya se ha establecido como herramienta básica en muchos campos de investigación. Este tipo de experimentos que utiliza exclusivamente expresiones teóricas y una o varias maquinas que las procesen, se ha ido mejorando con los años. Poco a poco nacen herramientas informáticas que nos facilitan más el trabajo y nos permiten realizar códigos más complejos. Además, los avances en miniaturizar el hardware y hacer más eficientes los componentes, han resultado en tener mayor memoria que llenar y mayor capacidad de cálculo. Los nuevos sistemas de hardware nos permiten nuevas funciones, y nosotros las aprovechamos a la hora de realizar nuevos programas, perfilándolos para recrear mejor sistemas físicos.

La simulación de partículas de plasmas tiene una gran relevancia, nos los encontramos en lugares muy diversos, como en la descarga de gases, en plasmas de astrofísica, o en situaciones concretas como en el tokamak (investigación de la fusión nuclear). Las simulaciones nos sirven para analizar características sobre las cuales no podemos extraer datos, o confirmar que las leyes teóricas se cumplen por medio de la medida experimental. Por ejemplo, podemos calcular la estadística de la velocidad las partículas en el equilibrio, el campo eléctrico medio en un punto del plasma, la cantidad de iones que se formarían en el equilibrio...

Toda la utilidad que posee la simulación de partículas y el hecho de que sea necesario practicar competencias en C/C++ y aprender nuevas características de la programación en GPU (Unidad de Procesado Gráfico), que a continuación explicaremos, han sido mi motivo para realizar este TFG.

Cuando hablamos de simulación, entendemos de alguna manera el resultado de los cálculos que realiza el ordenador a partir de unas ecuaciones y condiciones que introducimos. Normalmente, el ordenador ejecuta el programa en su procesador o CPU (Unidad Central de Procesamiento), de manera que realiza los cálculos de uno en uno. Hasta que no acaba uno, no empieza el siguiente, a lo que le llamamos computación secuen-

cial. A pesar de que las CPUs modernas se denominan multinúcleo, de manera que pueden tener hasta 2, 4, 6 o 8 hilos de cálculo activos, dependiendo de los núcleos reales o virtuales que tenga, son una cantidad de procesos simultáneos muy pequeña.

En ciertas ocasiones, a la hora de realizar los cálculos, son necesarios ciertos valores anteriores propios de la simulación. Por ejemplo, a la hora de realizar los cálculos para la dinámica de una partícula necesitamos saber las posiciones de todas las partículas en el instante anterior de cálculo. Es lo que nos ocurrirá a nosotros, ya que para ejecutar la iteración n+1 necesitaremos haber finalizado la iteración n y que sus datos estén debidamente guardados. A este tipo de computación que necesita se ejecuta un paso en cuanto otro anterior acaba se denomina computación secuencial.

Sin embargo, en lo que nos atañe en la simulación de partículas, en cada instante tendremos que calcular la fuerza que cada partícula ejerce sobre todas las demás partículas. Si queremos realizar esta simulación en una CPU convencional tendríamos que calcular las fuerzas de todas las partículas sobre una y después pasar a la siguiente, hasta acabar con todas ellas. Es posible realizar estos cálculos, pero es muy ineficiente, debido a que hay maneras mejores de abordar el problema.

En contraposición a la computación secuencial, si tenemos una cantidad de datos conocidos y vamos a realizar distintos cálculos con ellos (¡sin cambiarlos en el proceso!) podemos ejecutar cada uno de los tratamientos de datos por separado. En el caso del cálculo de fuerzas en la simulación de partículas estamos en este caso, pues si tengo un plasma de 200 partículas, el cálculo de la fuerza que ejercen las 200 partículas sobre la número 34 es totalmente independiente a la que ejercen sobre la 122. Si les doy a dos ordenadores totalmente separados la información de la posición de todas las partículas y la situación física completa, y hacemos que cada uno calcule la fuerza sobre una partícula distinta, sería totalmente equivalente a calcular en el mismo ordenador la posición de las dos partículas, solo que invirtiendo la mitad de tiempo (despreciando el tiempo de juntar los datos al final de ambos ordenadores). En esto consiste la esencia de la

computación en paralelo, en realizar cálculos equivalentes de forma separada y análoga. A cada proceso de simulación que se ejecuta de forma paralela lo llamaremos hilo de procesado o simplemente, hilo.

Para realizar nuestro cálculo vamos a utilizar una combinación de ambas técnicas, en cada instante de tiempo utilizaremos una computación en paralelo para calcular las fuerzas y entre instantes de tiempo utilizaremos la computación secuencial, pues no podemos iniciar un nuevo instante o paso de tiempo sin acabar el anterior dado que todas las partículas se mueven simultáneamente.

Es relevante destacar que de las simulaciones que vamos a hacer ya existen muchos artículos y estudios en este área de trabajo. En nuestro caso, lo que vamos a hacer es aumentar el número de partículas y estudiar y comparar los resultados. Así, al haber un mayor número aumentamos la complejidad del sistema y disminuimos en la medida de lo posible el ruido estadístico correspondiente en los datos extraídos. Para aumentar el número de partículas vamos a aprovechar las nuevas prestaciones que el software y el hardware nos brindan.

2 Funcionamiento GPU

Una tarjeta gráfica o GPU (Unidad de Procesado Gráfico) es un dispositivo de hardware diseñado para la ejecución de procesos en paralelo. Ya hemos dicho que los procesadores modernos pueden tener varios núcleos, pero las tarjetas gráficas tienen tal cantidad que las hacen perfectas para ejecutar códigos de forma simultánea.

Los núcleos de una CPU son complejos, siendo capaces de ejecutar una gran multitud de tareas, como las que desarrollamos diariamente en cualquier ordenador. Mientras que los núcleos de la GPU, llamados núcleos CUDA, son infinitamente más sencillos, siendo útiles simplemente para operaciones básicas como cálculos, justo como necesitamos.

Las GPUs que vamos a utilizar serán las de la marca de NVIDIA, ya que la misma distribución nos ofrece el software capaz de desarrollar las actividades que nos interesan, denominado CUDA. Vamos a nombrar algunas de las gráficas de esta distribuidora, para ver la evolución de los núcleos que contienen:

- **Las gráficas antiguas, como la GTX 330, lanzada al mercado el 2001, ya contaban con una cantidad de 96-116 núcleos.**
- **Más moderna, la GTX 750ti, cuenta con 640 núcleos, es del 2013.**
- **Para realizar el TFG se han usado dos gráficas con el mismo modelo, la GTX 1060, la cual cuenta con 1280 núcleos, que fueron introducidas en 2016.**
- **La última que ha salido al mercado, el 20 de septiembre de 2018, es la GeForce RTX Titan, contiene 4608 núcleos CUDA. Sin embargo, su coste económico de 2720 euros ya nos va mostrando cual es una de las dificultades que tiene este tipo de programación.**

Figure 1: Tarjeta Gráfica GeForce RTX 1060

Llegamos a cifras del orden de miles de núcleos CUDA en un solo dispositivo, con ello será suficiente para realizar nuestros cálculos. Después de ver estas cantidades de núcleos, es innecesaria la explicación de porque seguimos llamando computación secuencial a aquella que se ejecuta en la CPU, aún siendo 8 procesos simultáneos.

***Nota:** Con el paso de los años se están desarrollando CPUs con mayor facilidad para emprender código en paralelo. Hay anunciados procesadores capaces de soportar 112 procesos por parte de intel. Sin embargo, dista mucho de las GPUs y tendrán otro tipo de propósitos, pero es algo que para este trabajo no nos incumbe.*

Dado que las tarjetas gráficas se dedican a procesar datos, además de los núcleos necesitan estar dotadas de memoria interna. Están optimizadas para que cada núcleo tenga acceso a distintos tipos de memorias, cada una con sus ventajas e inconvenientes:

- Memoria "Global" : es la más abundante en cualquier dispositivo GPU, se podría comparar con la memoria RAM de un ordenador. Su principal ventaja es su gran tamaño y que a ella pueden acceder todos los hilos ejecutados, sin embargo, peca de ser extremadamente lenta, por lo que es un lugar pésimo para guardar las variables a las que vamos a acceder un gran número de veces.
- Memoria "Constant" : es usada exclusivamente para guardar, como su nombre indica, valores constantes que la GPU utiliza con frecuencia, debido a que es de rápido acceso y todos los hilos pueden acceder a ella. Aprovecharemos su naturaleza para almacenar nuestras constantes, valores predefinidos antes de compilar el programa, en caso de que sea necesario.
- Memoria "Shared" : este tipo de memoria es propia de cada bloque, es decir, cada bloque tiene su propia unidad de memoria "Shared". Debido a la gran cantidad de bloques que tiene nuestra GPU, es una memoria limitada y a la que solo pueden acceder los hilos que se ejecutan en ese bloque. Como está en el propio bloque el acceso a ella es muy fácil, lo que se traduce en que es un tipo de memoria de manejo muy rápido. Desde la propia compañía de NVIDIA, se recomienda que de los 48kb con los que contamos en cada bloque, no se utilice más de 32kb, para no disminuir el rendimiento de la GPU.
- Memoria "Local" : se trata de una memoria que también es pequeña. La mayor desventaja que tiene es que solo pueden acceder a ella el mismo hilo que la declara, pero a cambio es igual de rápida que la "Shared" y con más capacidad de almacenamiento. Se puede ver como la memoria "Cache" en un ordenador.
- Memoria de Registro : es similar a la memoria "Local", de rápido acceso y solo es vista por el hilo donde se declara. Cada GPU tiene un máximo de registros disponibles definido por hardware.

- **Memoria de Textura : es más compleja y para nuestro trabajo tiene poca utilidad, por lo que no entraremos en detalles.**

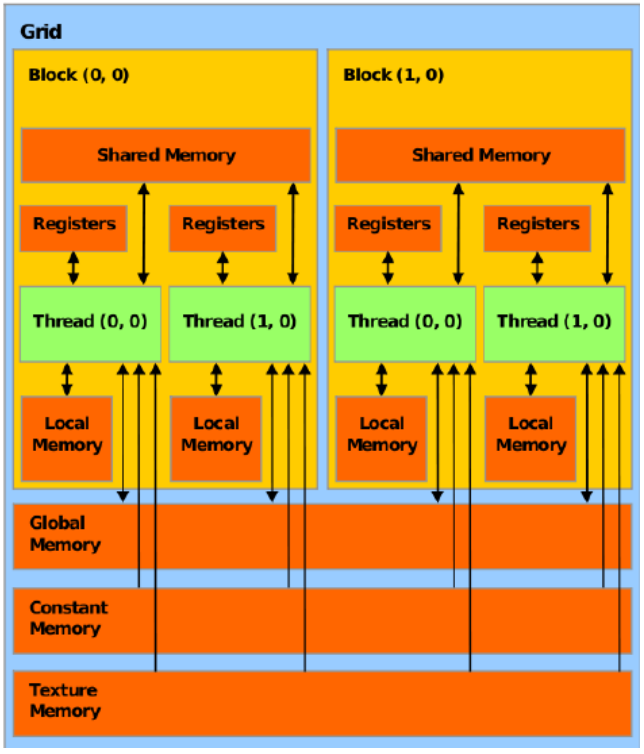

Figure 2: Gestión de la memoria en la GPU

Todos esos tipos de memoria existían tanto en las tarjetas gráficas antiguas de NVIDIA como en las GPUs actuales, por lo que no tendremos diferencias sustanciales en la capacidad de alguna de esas memorias. La memoria "Global" sí que es mucho mayor en las GPUs actuales, sin embargo, ya era suficientemente grande con la anterior, por lo que no nos afectará en absoluto.

La GPU está altamente segmentada en sus núcleos CUDA, lo que le permite poseer una gran cantidad de unidades funcionales. No todos los núcleos son iguales, se subdividen en vértices, pixeles, de salida de renderizado... y por otro lado, la gestión de la memoria. La manera de administrar el total de los elementos de la GPU se denomina arquitectura, la cual ha ido variando a lo largo de los años, adaptándose a las nuevas prestaciones y necesidades que el hardware permitía. La arquitectura define en gran parte las utilidades y operaciones que podemos realizar con la tarjeta gráfica.

La computación en paralelo en las GPU nos añade una cuestión: los calculos que realizamos son simultáneos, pero aun así está permitida la comunicación entre los procesos ya que ciertos datos se almacenan en memorias comunes. Así, trataremos de aprovechar esta característica. Si a un hilo le mandamos escribir en una memoria, y a otro hilo le mandamos leerla en un punto inmediatamente posterior en la secuencia del programa, no tenemos garantías de que el primer hilo haya grabado la memoria antes de que el segundo hilo la haya leído, ya que como no son procesos ideales pueden no computar todos a la misma velocidad.

Por eso vamos a añadir un proceso, la sincronización. Para entenderlo podemos utilizar un simil, el mecanismo de sincronismo consta de una barrera, deteniendo en ese punto el proceso a todos los hilos que se están ejecutando. Esta barrera se abrirá dejando que siga cada ejecución en cada hilo en el momento en el que todos los hilos lleguen a la barrera. Si introducimos esta barrera después de ordenar al primer hilo escribir en la memoria, obligaremos a que el segundo hilo lea la memoria ya actualizada.

Hay distintas maneras de ejecutar el sincronismo, por ejemplo, cuando se acaba de ejecutar código en la CUDA y se vuelve a la CPU se fuerza un sincronismo de todos los hilos ejecutados. Otra manera de hacerlo en código es mediante el comando "syncthread()", el cual hace que se produzca un sincronismo local. Se sincronizan todos los hilos que se encuentran en el mismo bloque, pero no los que están en distintos bloques entre ellos.

Para realizar el cálculo de las fuerzas vamos a necesitar realizar más de una vez sincronismos globales (de toda la malla de hilos que invocamos), ya describiremos en que puntos concretos los usaremos más adelante, en la descripción del programa. Sin considerar las nuevas versiones de CUDA tendríamos dos posibilidades:

- Limitar su uso a un bloque para la simulación. Existe un mecanismo por comando de sincronismo lo suficientemente potente y rápido para permitirnos sincronizar todos los hilos de un bloque. El punto negativo es que hay una cantidad máxima de hilos que se pueden declarar en un bloque, además que la memoria compartida de acceso rápido, la memoria "Shared", es bastante pequeña, por lo que limita la cantidad de partículas que podemos ejecutar en un solo bloque.
- Usar más de un bloque, pero los mecanismos de sincronización globales que existen son muy lentos, es necesario devolver el control del programa a la CPU para que lo realice y volver a ejecutar nuevo código en la CUDA. Sus ventajas son que admitiría más partículas, ya que cuentas con la memoria "Shared" de varios bloques y con los hilos que se puedan declarar en todos los bloques. Pero por otro lado se vuelve inviable de realizar por los largos tiempos de computación que produce el método de sincronismo.

Pero todo cambia con la llegada de CUDA 9.0 en 2017, es un software introducido después de un cambio en hardware muy relevante. En 2016 con la entrada de la gamma de tarjetas gráficas GeForce RTX 10 hubo un cambio de arquitectura en ellas, llegando a la arquitectura Pascal. Fue un avance que permitió al software crear una malla de hilos, es decir, podemos englobar todos los hilos que invoquemos dentro de una misma malla. En la malla podemos introducir hilos de distintos bloques, incluso definir distintas mallas que se entrecrucen con los bloques.

El avance que nosotros aprovecharemos es que mientras han conseguido definir una malla global de hilos, han conseguido crear una manera de sincronizar todos los hilos de la malla de manera eficiente. Utilizando un comando en código de forma análoga a como antes usabamos "syncthread()"

podremos realizar ejecuciones que comprendan más de un bloque y, en los puntos que nos interese, sincronizar todos nuestros procesos paralelos.

Con la última versión se introducen muchas más opciones de manejo de hilos, pudiendo definir varias mallas, dividiendolas en las que nos sea necesarias a lo largo del programa, uniendolas, parando la que nos interese... Añaden mucha libertad en cuanto al manejo del hilo que procesa en sí y no solo al proceso que ejecuta. Sin embargo, a nosotros no nos hace falta todo eso, simplemente necesitamos sincronizar los hilos de toda la malla, y eso lo podemos hacer gracias a la $CUDA 9.0^{TM}$.

3 Equipo de Simulación

En este apartado vamos a comentar cual será el equipo que vamos a usar, y que requerimientos o características se necesitarían para ejecutar determinados comandos obligatorios de $CUDA 9.0^{TM}$ y superiores. A continuación, mostraremos los datos de las dos GPUs que hemos usado, las dos son idénticas, siendo ambas tarjetas gráficas GTX NVIDIA 1060 de 6GB. Para ver estos datos, el propio software de CUDA contiene unos ejemplos, entre los cuales se encuentra un programa que lee la tarjeta de procesado gráfico conectada al dispositivo. Este programa se llama "deviceQuery", resultando la tabla de la figura 2, en la que se han resaltado los datos a comentar.

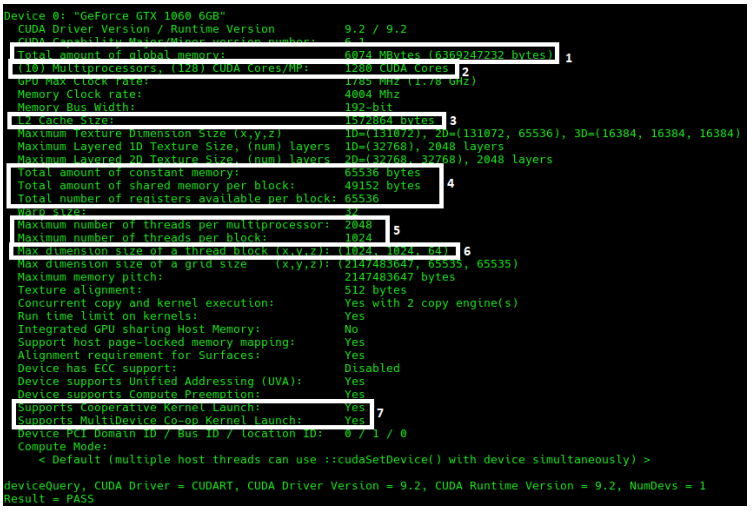

Figure 3: DeviceQuery - NVIDIA GeForce GTX 1060

- **Recuadro 1: como vemos, se trata de la memoria "Global", la más grande de la GPU, todas las tarjetas gráficas actuales tienen como capacidad varios gigabites, en nuestro caso serán 6 Gb. Como comentamos antes, no deberíamos tener problemas de capacidad en este tipo de memoria.**
- **Recuadro 2: aquí vemos los núcleos que tiene la gráfica.**

- **Recuadro 3: consta del tamaño de la memoria "local", la cual, aunque es pequeña, no nos dará muchos problemas de capacidad. Está relacionado con uno de los tipos de cache de la GPU.**
- **Recuadro 4: podemos ver la asignación máxima por hardware de las memorias "Constant", "Shared" y la máxima capacidad de la memoria de registro. Suele ser la misma en todos los dispositivos, pero no está de más comprobarlo.**
- **Recuadro 5: nos especifica el máximo número de hilos que podemos invocar, según el hardware por bloque y por multiprocesador.**
- **Recuadro 6: cuando invocamos hilos lo podemos hacer de forma que cada uno de ellos forma parte de una celda de una malla, de modo que su índice puede depender de un dato tridimensional, este recuadro muestra las dimensiones máximas de la malla en las tres dimensiones posibles. Nosotros, por simplicidad y por no necesitar más, usaremos una malla unidimensional, todos los hilos en el "eje x".**
- **Recuadro 7: aunque lo dejemos para el final, este es el primer sitio en el que nos tenemos que fijar para saber si podemos ejecutar el sincronismo entre bloques. Si en estas opciones nos resulta un "No", no podremos ejecutar los comandos adecuados para permitir el tipo de sincronismo que abarca más de un bloque de forma eficiente, y por lo tanto, no podremos realizar el tipo de programas descritos a continuación.**

No vamos a centrarnos en tanto detalle en las capacidades de la CPU porque cualquier comercial o cualquier CPU que sea compatible con una placa base que soporte una GPU moderna es capaz de ejecutar los procesos que vamos a realizar. Esto se debe a que éstos son muy sencillos, tales como imprimir datos en un archivo exterior, dar formato a variables o invocar a la GPU. Después de saber analizar nuestro dispositivo, ya podemos explicar en qué consiste la ejecución del programa en sí.

4 Física de la Simulación

4.1 Aproximaciones

Como ya hemos mencionado previamente, vamos a aprovechar las ventajas de la computación en paralelo para simular un plasma de partículas. Todas tendrán la misma carga tanto en cantidad como en tipo de carga. Las aproximaciones que haremos serán:

- **La situación física consistirá en una celda cúbica de lado L, en la que tendremos las partículas de la simulación. Se supondrán condiciones periódicas en el espacio, es decir, si una partícula sale con cierta velocidad por un lado de la celda, entrará con la misma velocidad y por la cara opuesta. Es decir, supongamos que sale de la celda en la coordenada (L, L/2, L/2), la partícula volvería a entrar en la celda original por (0, L/2, L/2), justo por el mismo punto de la cara opuesta por la que sale.**

- **Solo contaremos las interacciones de las partículas que estén como muy lejos a la mitad de lo que mide la celda. Si la partícula se encuentra en un lateral, en principio se acaba la celda, pero emularemos otra vez una celda situada en ese lugar que sea equivalente a la nuestra.**
  Es decir, supongamos una partícula situada en las coordenadas (L-0.1, 0, 0), una partícula situada en (0.1, 0, 0) no ejercería ninguna fuerza sobre ella, ya que está muy lejos. Sin embargo, una "partícula virtual" correspondiente a la segunda situada fuera de la celda original en las coordenadas (L + 0,1, 0, 0), sí que ejercerá fuerza sobre la primera partícula. Para contabilizar esa fuerza utilizaremos las coordenadas de la partícula que no ejerce la fuerza directamente.

- **La única interacción que se considerara será la de Coulomb, además, debido a la equidad de la carga de todas las partículas, esta interacción será siempre repulsiva y dependerá exclusivamente de la distancia entre ellas.**

$$\vec{F} = \frac{1}{4\epsilon_0 \pi} \frac{q^2}{r^3} \vec{r} \tag{1}$$

- **El tamaño de las partículas será despreciable frente al resto de distancias típicas de la simulación. Consideraremos para su dinámica el centro de masas, localizado en unas coordenadas concretas y únicas. El tamaño de las partículas, d, solo afectará a la acotación del campo eléctrico en sus proximidades inmediatas, ya que no tendremos en cuenta el choque entre partículas. A efectos prácticos, de darse el caso de choque directo entre ellas, se atravesarían como dos nubes cargadas positivamente. Sin embargo, es difícil que suceda, ya que las fuerza repulsiva tenderá a que no ocurra. La fuerza que notarán las partículas será:**
  Si $r < d$,

$$\vec{F} = -\frac{q^2}{4\pi\epsilon_0 d^2}\vec{r} \tag{2}$$

  Si $r > d$,

$$\vec{F} = -\frac{q^2}{4\pi\epsilon_0 r^2}\vec{r} \tag{3}$$

  La aproximación que estamos usando hace que la fuerza deje de ser conservativa. Sin embargo, la cantidad de veces que se usará la fórmula (2) será totalmente despreciable, ya que es muy poco probable que las partículas lleguen a acercarse tanto entre sí. La distancia d será mucho menor que las distancias típicas del problema, en principio usaremos $d = 10^{-4}$ **Unidades de Simulación de Distancia.**

Para asignar unas condiciones iniciales, consideraremos que todas las partículas tienen el módulo de la velocidad inicial igual, sin embargo, su dirección es fijada de forma aleatoria. Tras el paso del tiempo, las partículas irán intercambiando paulatinamente entre ellas energía cinética y potencial, obteniendo finalmente una curva de forma Maxwell Boltzmann en el histograma de su distribución de velocidades. Comprobaremos esta afirmación, además de ir viendo como llegamos a esta curva con el paso del tiempo.

Podríamos haber realizado simulaciones con partículas positivas y negativas, con lo que implicaría introducir potenciales regularizados. No hemos buscado tanto la complejidad en la naturaleza del potencial, por lo que con partículas de un mismo tipo será suficiente.

4.2 Unidades de Simulación

Para nuestra simulación, no vamos a usar unidades del sistema internacional ni similares, ya que no tienen sentido. Por un lado, no estamos resolviendo un problema totalmente real, y por el otro, la escala de unidades que nos propone el sistema internacional no nos resulta cómodo en absoluto. Así, trabajaremos con nuestro propio sistema y, de ser necesario, ya se realizará un cambio de escalas adecuado. Las magnitudes que vamos a fijar son:

- **Distancia: nuestra unidad de distancia típica será la distancia media entre partículas. Es decir, en cada simulación puede tener un valor distinto.**
- **Velocidad: fijaremos la unidad de velocidad de simulación en el módulo de la velocidad inicial de nuestras partículas, ya que todas partirán con la misma velocidad.**
- **Masa: la masa de nuestras partículas será la unidad.**
- **Parámetro de interacción (Γ): este parámetro nos ayudará a definir la relación entre la energía potencial y la energía cinética de nuestro sistema. La energía cinética de cada partícula depende de la velocidad y la masa, siendo ambas la unidad, la energía cinética será también del orden de la unidad. Por otro lado, la energía potencial depende del parámetro de interacción y la unidad de distancia. Sabiendo que esta segunda variable es también la unidad, se puede decir que depende solo de Γ. Al estar en un plasma de partículas libres, la dinámica tiene que ser dirigida fundamentalmente por la energía cinética, siendo ella de orden superior, por lo que Γ tiene que ser de orden inferior a la unidad. La expresión de Γ es:**

$$\Gamma = \frac{q^2}{4\pi\epsilon_0 r_0}\frac{1}{2kT} \tag{4}$$

siendo r_0 la distancia media entre partículas.

Para distintas situaciones este valor puede ser radicalmente distinto, por ejemplo, para situaciones de descarga de gases a unos cientos

de grados celsius, el valor de Γ será del orden de las centésimas de unidad.

Si la densidad de partículas es baja, este valor tenderá a tomar valores a su vez pequeños, por ejemplo, en el tokamak disminuye hasta valores del orden de 10^{-6}. Esto producirá un aumento en el tiempo que requerirá la simulación para alcanzar la estabilización que buscamos. En nuestro programa usaremos valores de Γ tales que:

$$0.1 > \Gamma > 0.001 \tag{5}$$

Dependiendo de la intensidad de la interacción que busquemos iremos variando su valor, sin embargo manteniendo el valor en un rango correspondiente al que tendría un gas en descarga. El tiempo de la simulación, aparte del valor de la intensidad de interacción, está relacionado de forma directa con el paso de tiempo que elijamos. Si aumentamos la interacción, habrá que hacer que el paso de tiempo sea más fino de forma inversamente proporcional para mantener la calidad. Más adelante se detallará el motivo en el Desarrollo Matemático.

4.3 Situación Inicial

Las condiciones iniciales, como hemos comentado, serán simples. Por un lado para la posición, elegiremos lugares aleatorios dentro de la celda, y por el otro, en la distribución de velocidades crearemos una delta en módulo de velocidad igual a uno, siendo la dirección de movimiento aleatoria.

En los primeros instantes, debido a que las posiciones son aleatorias y las velocidades son forzadas, no tenemos porque estar en una situación de equilibrio, de hecho, es obvio que no podemos estar en ella, pues el espectro en la distribución de velocidades será distinto ya que tiene que haber interacciones entre las partículas.

Si partimos de una situación inicial de no equilibrio van a suceder intercambios de energía tanto cinética como potencial. Si además la distribución de las posiciones iniciales no es la adecuada, habrá intercambio entre la energía cinética y potencial, lo que provocará un enfriamiento o un calentamiento global del sistema.

En una simulación con partículas positivas y negativas, es decir, un plasma real, lo intuitivo puede ser pensar que el gas siempre se calienta, pero nada más lejos de la verdad, dependerá totalmente de la disposición inicial de las partículas.

Al distribuir las partículas de distintos tipos de forma aleatoria, estas van a tender a estar repartidas de forma homogénea, es decir, tendrán energía potencial cerca de cero. En el equilibrio, la energía potencial debería ser negativa, por lo que se tendrá que producir un aumento en la energía cinética para que la potencial disminuya.

Sin embargo, si en la situación inicial se crea, tanto de forma casual como forzada, una distribución en la cuál la energía potencial es menor que la que tendría el sistema en equilibrio, el sistema necesitaría ganar energía potencial y el efecto sería una disminución de temperatura. De este hecho, ya se han realizado estudios previos, de los que se extraen resultados como:

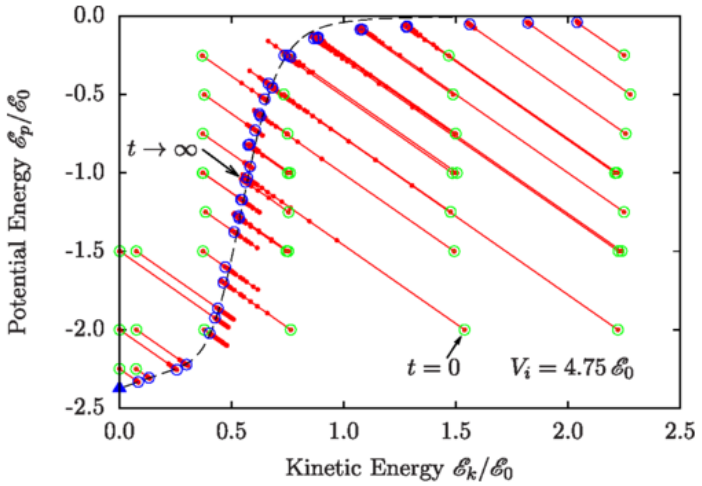

Figure 4: Estabilización energías desde situaciones iniciales - Classical molecular dynamics simulations of hydrogen plasmas and development of an analytical statical model for computational validity assessment.

En la figura (4), procedente de artículos de este campo [3], los puntos verdes representan los estados iniciales. Las rectas rojas son diferentes estados en los que se mantiene constante la energía del sistema, por lo que si empezamos en uno de los puntos verdes solo podemos avanzar por ellos.

Los puntos azules son posibles estados de equilibrio a los que llegamos en la estabilización de las partículas. Dependiendo si el sistema comienza con mayor o menor energía potencial de la que tendría en el equilibrio, el sistema se calentará o se enfriará respectivamente.

En el caso en el que dejaramos libres a un conjunto de partículas positivas, no tendríamos curva de equilibrio, por lo que las partículas tenderán a calentarse de forma casi indefinida. Tenemos un plasma en el que todas las partículas son positivas, la energía potencial contenida él es desmesurada. En el intento de que se equilibren la energía potencial y la cinética se provocará un aumento de temperatura del plasma, sin llegar nunca a el equilibrio.

Sin embargo, nuestras condiciones de periodicidad provocarán que esto no suceda. Será posible llegar a un equilibrio, al menos en las velocidades, al cabo de un número de unidades de tiempo.

5 Desarrollo Matemático

5.1 Resolución de las Ecuaciones

Vamos a comentar brevemente las ecuaciones dinámicas que rigen nuestro sistema. Simplemente será la ecuación diferencial de Newton igualando la masa a la unidad:

$$\frac{d(m\vec{v})}{dt} = \frac{d\vec{v}}{dt} = \vec{F} = \sum_{i=1} \vec{F_i} \tag{6}$$

Como se ve en la ecuación escrita anteriormente, sumaremos todas las fuerzas que afectan a las partículas, y eso será el factor que haga a la velocidad de cada partícula cambiar. Resolveremos la ecuación de forma numérica, resultando que la velocidad de nuestras partículas será:

$$\vec{v} = \vec{v_0} + \Delta t * \vec{F} \tag{7}$$

Donde $\vec{v_0}$ será la velocidad de las partículas en el paso anterior al que estoy calculando. De forma análoga, llegaremos a la expresión de las posiciones de nuestras partículas:

$$\frac{d\vec{x}}{dt} = \vec{v} \rightarrow \vec{x} = \vec{x_0} + \Delta t * \vec{v} \tag{8}$$

En esencia, lo que estamos haciendo es pasar de una ecuación diferencial a una ecuación de diferencias. Lo único que nos interesa para cada instante de tiempo es el paso de tiempo que definamos, Δt, y los datos instantáneos de la velocidad y posición en el mismo instante.

El efecto que provocamos es considerar que en la duración del paso de tiempo las partículas no se mueven, es decir, ejercen la misma fuerza sobre el resto de partículas durante un intervalo de tiempo, cuando eso es radicalmente falso. En la realidad las partículas se mueven de forma continua por lo que hay un cambio progresivo en la fuerza total.

Sin embargo, aunque no sea lo que sucede en la realidad, debido a los límites de la computación, tendremos que realizar una simulación donde esta será una aproximación en los cálculos. El paso de tiempo que usemos tendremos que definirlo de manera que sea ínfimo para que se parezca a una situación real. Con pequeño, nos referimos a que sea un tiempo despreciable, es decir, que sea un tiempo tan pequeño, que en ese tiempo las partículas hayan variado su posición una cantidad extremadamente pequeña en comparación a las distancias típicas de la simulación (la distancia media entre partículas).

Así sucederá, si la fuerza media que se ejercen las partículas entre sí es la unidad multiplicada por Γ, es decir, aproximadamente 10^{-2}. La velocidad media de cada una será del orden de la unidad y usaremos pasos de tiempo del orden de 10^{-4}. De las ecuaciones (7) y (8) extraemos que el cambio en la posición en cada paso será de aproximadamente 10^{-4}, lo que consideramos suficiente. El cambio en la velocidad en cada instante será del orden de 10^{-6}.

El efecto de tener la ecuación de diferencias será que la fuerza promedio sobre cada partícula será mayor que si fuera continua. Las partículas, al tener un movimiento continuo, regulan la fuerza que ejercen unas sobre otras en todo momento. Nosotros no podemos simular el anterior efecto, ya que tenemos que elegir un intervalo de tiempo en los que medir la fuerza. En el paso de tiempo elegido caben infinitos momentos del continuo, por lo que nuestras partículas se moverán menos veces y no irán regulando sus fuerzas paulativamente. El resultado será que si elegimos un paso de tiempo mayor, como las fuerzas promedio son mayores, el sistema se calentará de forma automática. Este incremento de temperatura es un incremento parásito que comentaremos más adelante.

5.2 Situación Equilibrio

La distribución de velocidades del conjunto de partículas en el equilibrio debe ser una distribución estadística de Maxwell Boltzman, es decir, la curva tendrá siguiente forma:

$$f(v) = 4\pi(\frac{m}{2\pi kT})^{\frac{3}{2}}v^2\exp(\frac{-mv^2}{2kT}) \tag{9}$$

Como hemos forzado a que la velocidad inicial de las partículas es la unidad, la velocidad cuadrática media también lo será, resultando en la siguiente simplificación:

$$f(v) = \sqrt{\frac{54}{\pi}}v^2\exp(-\frac{3}{2}v^2) \tag{10}$$

Compararemos la expresión (11), introduciendo si fuera necesario algún factor de cambio de temperatura relativo a la que corresponde a la velocidad cuadrática media, con los histogramas de las velocidades de nuestras partículas. Introduciendo el factor mencionado, resultará en,

$$f(v) = \sqrt{\frac{54}{\pi}}(\frac{1}{T_R})^{3/2}v^2\exp(-\frac{3}{2}(\frac{1}{T_R})^2v^2) \tag{11}$$

siendo T_R el factor relativo de temperatura respecto a la inicial.

6 Funcionamiento del Programa

A continuación, expondremos paso a paso las acciones que vamos a utilizar para el cálculo final de nuestra simulación. Vamos a analizar cómo utilizamos las expresiones y aproximaciones mencionadas previamente, como los mecanismos de sincronismo. Será útil para aquellos que tengan interés en ver de forma más específica cómo se programa en una GPU. Además, que se enunciaran los cambios que CUDA 9.0 añade.

6.1 Definición de Elementos

La definición de variables de distintas naturalezas y otros elementos de forma correcta es elemental para la ejecución del código. En la figura (5) se incluirá la definición de las variables "Global", "Shared" y "de registro".

En el programa tendremos los datos de todas las partículas y necesitaremos almacenarlos en la GPU para utilizarlos en ella. Debido a la complejidad que tiene la memoria de una tarjeta gráfica es importante el cómo los situemos. El mayor problema que vamos a tener es que la memoria "Shared" es muy pequeña como para almacenar los datos de todas las partículas que podamos tener, en 32KB caben los datos de aproximadamente 900 partículas. Por este motivo tendremos que hacer "malabares" con las memorias, ya explicaremos estas técnicas en su propio apartado. Por ahora vamos a realizar una breve explicación de para qué usaremos los diferentes tipos de memoria:

- Memoria en "Global" : incluiremos una copia de las posiciones de todos los datos de partículas, es decir, posición, velocidad y fuerza sobre ella misma. No realizaremos apenas cálculos en ella pues es de acceso lenta. Servirá como nexo de unión entre las memorias propias de los bloques.
- Memoria en "Shared" : en cada bloque definiremos dos variables auxiliares de memoria, que denominaremos auxiliar de posición y auxiliar de fuerza. Explicaremos su cometido más adelante, demomento el programa tiene dos variables de acceso rápido y cuyo tamaño es limitado disponibles.

```
//Defino unas clases utiles

typedef struct {
    double x;
    double y;
    double z;
} vector;

typedef struct {
    vector r;
    vector f;
    vector v;
} particle;

//Variables en la memoria compartida de la GPU
__device__ particle Part[NTH];
__device__ vector InterFuerzaBloques[NB][NB][NT];
__device__ double d_mod_velocidad[NTH];
__device__ double d_energia_media;
//__device__ vector CampoElectricoSuma[NT];
//__device__ vector CampoElectrico;

__constant__ double L;
__constant__ double L_2;

//Variable en la memoria compartida del bloque
__shared__ vector FuerAuxInt[NT];
__shared__ vector PosAuxInt[NT];

__global__ void kernel( )
{
    cg::grid_group grid = cg::this_grid();

    //Ctes para el paso de Unidades de Simulacion al Sistema Internacional
    register double dt = PASO_tiempo;
    register double inversadt_auxiliar = 1.0/PASO_tiempo;
    register int    inversadt = (int) inversadt_auxiliar;


    register int iabs = blockIdx.x * NT + threadIdx.x; //Indice del hilo donde estás
    register int i = threadIdx.x;

    //Defino variables de registro
    register vector PosAuxBase;
    register vector FuerAuxBase;

    register double sqrt_energia;
```

Figure 5: Definición de algunas variables en CUDA

- **Memoria de "registro" y "Local" : definiremos la posición y la fuerza de las partículas "principales". También incluiremos las variables constantes o dinámicas que sean necesarias para agilizar el cálculo, como por ejemplo, las distancias entre partículas. Cada hilo definirá su propia variable de posición y de fuerza. Vamos a tener tantas variables de registro de posición como de partículas tenemos en la simulación. Al principio del programa cada hilo guardara una posición de partícula en su memoria de registro, desde ese momento diremos que el hilo se "encarga" de esa partícula o que el hilo está vinculado a la partícula. En ese hilo se calcularan las fuerzas sobre la partícula vinculada a él.**

El primer y más importante objetivo de la GPU será calcular la fuerza que cada partícula ejerce sobre todas las demás. Para ello, cada hilo calculará la fuerza que todas las demás ejercen sobre la partícula a la que está vinculada. Por lo tanto es necesario que en algún momento del código todos los hilos sean capaces de leer las posiciones de todas las demás partículas.

Podríamos hacer que lean directamente de la memoria "Global", sin embargo es tremendamente lenta. Lo que haremos será hacer que los hilos de un bloque, cada uno realizando solo un intercambio de memoria, guarden una parte de las posiciones de las partículas de la memoria "Global" en la variable auxiliar de posición situado en "Shared". De esta manera hemos conseguido que los hilos de ese bloque puedan acceder de manera rápida a las posiciones de todas las partículas que caben en la memoria auxiliar.

Cuando estos hilos acaben de usar las posiciones haremos que se almacenen la información de nuevas partículas, y así hasta que recorramos todas. El proceso resultará en que todos los hilos han podido calcular la fuerza de todas otras partículas sobre la que tienen vinculada. Esta fuerza la irán almacenando en la variable de registro de fuerza para acabar volcándola en la memoria "Global". Para acceder a cada tipo de memoria declararemos distintos índices:

- i absoluto ("iabs" en el código) : será el índice que nos indique el hilo que estamos usando en la malla global, de manera que recorra desde 0 hasta el número de hilos invocados menos uno. Será necesario para usarlo en la memoria "Global" ya que definiremos arrays de tantos elementos como partículas hay en la simulación.
- i relativo ("i" en el código) : este índice nos muestra la posición de nuestro hilo en el bloque. Cada bloque tiene el mismo número de hilos invocados en él, de manera que el índice irá de 0 a el número de hilos por bloque menos uno. Es necesario para utilizarlo en la memoria "Shared", pues todos los hilos del bloque van a acceder a ella y cada uno tiene que hacerlo a un lugar específico.

- **j absoluto ("jabs" en el código) : índice que va avanzando en el programa útil para identificar las partículas que copiamos a la memoria "Shared". Para cada bloque e hilo el índice j absoluto tendrá un valor distinto.**
- **j relativo ("j" en el código) : después de copiar los datos de las partículas para interaccionar en la memoria auxiliar tendremos que calcular la fuerza que ejercen sobre las que están en la memoria de registro de cada hilo del bloque. Para el cálculo recorreremos todas las partículas almacenadas en "Shared" con el índice j.**

Los índices están organizados de tal manera que vamos a calcular la interacción de j sobre i. Haremos un barrido en las j en el programa y cada hilo se encarga de una i, de manera que obtenemos todas las interacciones.

Si en programa invocamos la memoria el elemento 3 del array guardado en "Shared", todos los hilos en los que se invoca el comando accederan a la memoria 3 de su bloque, en otras palabras, si son de distinto bloque accederán a distintos puntos de memoria, existe una degeneración en los índices i relativo y j relativo. Esta degeneración es necesaria para acceder a la memoria "Shared", sin embargo, se puede romper utilizando los índices i absoluto y j absoluto. Es necesario romper la degeneración para acceder a la memoria "Global", ya que es común a todas.

Sin embargo, al utilizar la memoria de registro o "Local" no necesitamos utilizar índices, cada hilo se autogestiona su propia memoria. Si nombro una variable de registro cada hilo accederá a la suya.

La ventaja mencionada que nos proporciona 9.0 de crear mallas la vamos a necesitar, pues incluiremos todos los hilos de todos los bloques que se ejecuten en el programa en la misma malla. Ésta contendrá una cantidad de hilos como partículas tiene nuestra simulación.

```
cg::grid_group grid = cg::this_grid();
```

Figure 6: Definición de la malla en la que trabajaremos

6.2 Movimientos de memoria

Los datos de todas las partículas se encontrarán en la memoria "Global" son muy lentas, por lo que tendremos que transferirlos a la memoria "Shared" para su uso. En la memoria de posiciones auxiliares de cada bloque vamos a guardar la información de tantas partículas como hilos se ejecutan en cada bloque. Queremos recorrer todas las partículas, por lo que, en principio, nos hará falta tantos movimientos de memoria como bloques usemos en el programa.

Al principio del programa, encargaremos a cada hilo guardar los datos de la posición de su partícula vinculada a la memoria local del propio hilo. Esta variable no cambiará en ningún momento del cálculo de fuerzas. Inmediatamente después, inicializaremos a cero todas las fuerzas y copiaremos la primera ronda de partículas a la memoria auxiliar de cada bloque.

La primera ronda consistirá en copiar a la memoria "Shared" de cada bloque las posiciones que están en las memorias de registro de cada bloque. En la segunda ronda copiaremos los datos de partículas que pertenecen al siguiente bloque adyacente, cuyo índice de bloque es una unidad superior al que se van a copiar las memorias. En el tercero, serán las partículas de dos unidades superiores, y así hasta que recorramos todos los bloques.

En la imagen (7) se muestra el desarrollo de las interacciones entre bloques de forma completa, donde, los números simbolizan en que paso del cambio de memorias de "Global" a "Shared", es decir, primero se calcularán las interacciones de la diagonal, e iremos moviéndonos en las columnas.

Sin embargo, no es necesario todos los movimientos de memoria que aparecen en la imagen, ya que aprovecharemos a calcular la fuerza del Bloque 1 sobre el Bloque 2 a la vez que calculamos la interacción del Bloque 2 sobre el Bloque 1. Reduciremos el número de iteraciones en el bucle correspondiente a interaccionar a la mitad más uno.

	Bloque 1	Bloque 2	Bloque 3	Bloque 4	Bloque 5
Bloque 1	1	2	3	4	5
Bloque 2	5	1	2	3	4
Bloque 3	4	5	1	2	3
Bloque 4	3	4	5	1	2
Bloque 5	2	3	4	5	1

Figure 7: Máxima cantidad interacciones de los bloques

	Bloque 1	Bloque 2	Bloque 3	Bloque 4	Bloque 5
Bloque 1	1	2	3	3	2
Bloque 2	2	1	2	3	3
Bloque 3	3	2	1	2	3
Bloque 4	3	3	2	1	2
Bloque 5	2	3	3	2	1

Figure 8: Bloques que interaccionan de igual manera

Para realizar el cálculo de ambas fuerzas de forma simultánea entrará en juego la variable auxiliar de fuerzas almacenada en "Shared" que aún no hemos mencionado. En esa variable calcularemos la fuerza que se ejerce sobre la partícula j relativa por parte de todas las partículas pertenecientes a la variable de registro de cada hilo en el bloque en el que se encuentran.

Cada hilo vinculado a su partícula tendrá como función calcular la fuerza que es ejercida sobre ella misma. Además, le introduciremos la nueva tarea de calcular la fuerza que su partícula ejerce sobre cada partícula j relativa. Como los datos de la memoria j relativa están guardados en "Shared" todos los hilos podrán leer su posición y escribir sobre su fuerza.

En cada paso de movimientos de memoria estaremos pasando a guardar distintas partículas en la memoria "Shared", por lo que volcaremos la variable de fuerza auxiliar, distinguida en cada bloque por un j relativo, a la memoria "Global", pasando al índice j absoluto.

Este proceso de cálculo de fuerzas complementario y secundario tiene su utilidad, ya que disminuye el tiempo de cálculo a la mitad. Tendremos que realizar la mitad de iteraciones en los movimientos de memoria. No añadiremos cálculos a las iteraciones que realicemos, ya que la fuerza de i relativo sobre j relativo será igual que la fuerza de j sobre i. Guardaremos en la variable de fuerzas "Local" el sumatorio de fuerzas en i, mientras que en la variable auxiliar de fuerzas en "Shared" el índice en el sumatorio será el j.

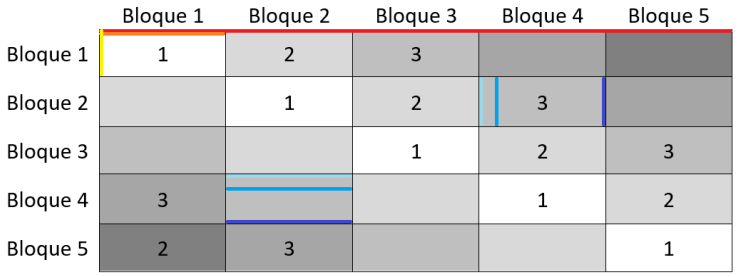

Figure 9: Nuestros cálculos de interacción de blqoues

Para acabar de dejarlo claro, la Figura (9) es el resumen de cómo vamos a sumar las fuerzas para calcular las totales. La línea roja representa lo que queremos conseguir, es decir, la suma de todas esas fuerzas individuales resultará en la fuerza que ejercen todas las demás partículas sobre la ligada al hilo.

En cada iteración de movimientos de memoria vamos a cálcular la suma de los datos de la línea naranja de la imagen y amarilla por separado, de manera que la suma horizontal se añada a la memoria de registro, mientras que la vertical se introduzca en la memoria auxiliar.

El sumatorio de las líneas verticales, debido a la simetría de la tabla, como se ve gracias a las lineas azules, son equivalentes a los sumatorios de las líneas azules horizontales. Por lo que no haremos los movimientos de memoria correspondientes para calcular lo que serían las líneas horizontales azules. Simplemente, sumaremos las verticales en el lugar correspondiente.

```
for(int PASO_sin_escrituras = 0; PASO_sin_escrituras < PASO_sin_escrituras_max; PASO_sin_escrituras++){

    PosAuxBase = Part[iabs].r;

    Part[iabs].f.x = 0;
    Part[iabs].f.y = 0;
    Part[iabs].f.z = 0;

    FuerAuxBase.x = 0;
    FuerAuxBase.y = 0;
    FuerAuxBase.z = 0;

    FuerAuxInt[i].x = 0;
    FuerAuxInt[i].y = 0;
    FuerAuxInt[i].z = 0;

__syncthreads();
```

Figure 10: Movimientos iniciales descritos anteriormente y algunas inicializaciones a cero

```
for(int ifbloque = 0; ifbloque < NB_2; ifbloque++){

FuerAuxInt[i].x = 0;
FuerAuxInt[i].y = 0;
FuerAuxInt[i].z = 0;
__syncthreads();

register int jabs = ( (blockIdx.x + ifbloque)%(NB) ) * NT + i;

PosAuxInt[i] = Part[jabs].r;
__syncthreads();
```

Figure 11: Movimientos iniciales descritos anteriormente

Un dato importante que hay que considerar antes de mover la memoria es, si nos compensa moverla. Si la vamos a mover de una variable a otra para simplemente hacer un número pequeño de cálculos, nos cuesta el mismo tiempo que acceder a ella para copiarla a otra memoria más rápida. En los procesos anteriores sí que nos interesaba, pues el número de cálculos hechos para las fuerzas son 640^2 en los peores casos que hemos ejecutado, es decir, del orden de $4 * 10^6$ por cada movimiento de memoria, mientras que en los casos más simples son del orden de 350 operaciones.

Por ejemplo, para realizar los cálculos de la velocidad a partir de la fuerza y de la posición a partir de la velocidad, que consistirá en realizar dos productos en cada hilo, lo haremos directamente en la memoria Global.

En las imágenes (10) y (11) vemos como pasamos los datos de las memorias "Global" a las memorias "Shared" y "Local", siendo las de "Shared" ya dentro del bloque correspondiente.

Por otro lado, en las imágenes (12) y (13) mostramos como devolvemos a memoria "Global" las variables de fuerzas, sumándolas a la número que existía previamente a esa variable.

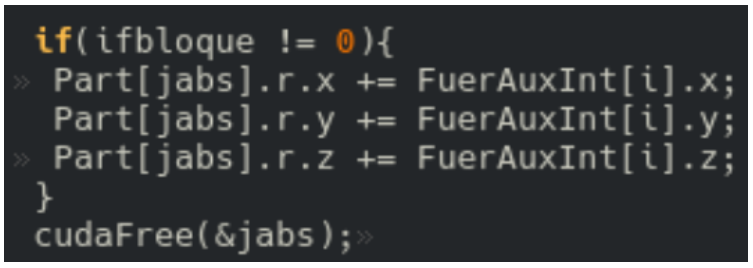

Figure 12: Devolvemos fuerzas de shared a la memoria Global en cada iteración en los bloques

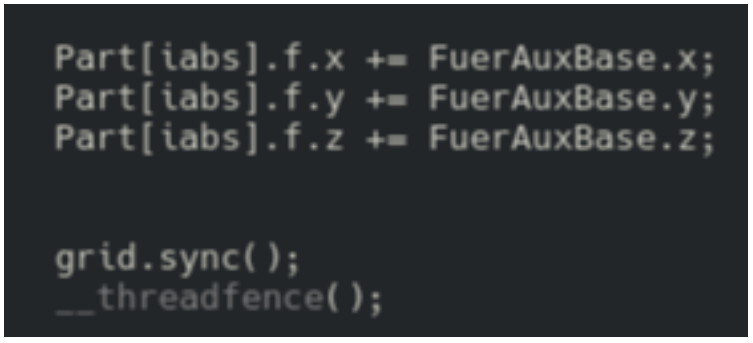

Figure 13: Devolvemos fuerzas de registro a la memoria Global en cada paso de tiempo

6.3 Cálculo de Fuerzas

A pesar de todo lo que hemos comentado, es en éste apartado en el que se creará la física de la simulación. La evolución de nuestro sistema dependerá del tipo de interacción que establezcamos entre nuestros elementos, en este caso partículas. Todos los otros apartados con detalles técnicos de la simulación son necesarios y nos interesa analizarlos a fondo, sin embargo son más como un andamio si quieres pintar una fachada.

La interacción entre las cargas, como ya hemos comentado, se deberá de forma exclusiva a la interacción coulombiana entre las partículas.

Esta parte del código es bastante fácil de implementar una vez que tenemos los datos que las fórmulas matemáticas usan en memorias de fácil acceso. Tendremos en cuenta dos cosas:

- Primero, tenemos que asegurarnos que la partícula está en la caja en la que consideramos que interacciona con cada partícula, es decir, que las partículas estén entre sí a una distancia menor de L/2. Para ello, calcularemos el vector entre la partícula del hilo y la que va a ejercer fuerza sobre ella. Si alguna de las componentes es mayor a L/2 le restaremos o sumaremos L a esa componente.

  Hemos conseguido arreglar las interacciones. Si una partícula i está en el borde superior de la celda y la partícula j está en la parte inferior, la distancia entre ellas es mayor que L/2, por lo que no interactuarían. Sin embargo, la partícula j situada en la parte inferior es la correspondiente a una partícula virtual en la celda inmediatamente superior a la nuestra. Esta partícula virtual sí que la tendremos que tener en cuenta para la fuerza sobre la partícula i. Lo que conseguimos al sumar L a la componente correspondiente del vector que une i con j es obtener el vector entre i y la partícula virtual correspondiente a j, por lo que podemos proseguir con el cálculo de fuerzas.

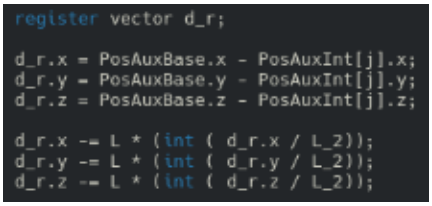

Figure 14: Redistribuir Partículas en el Programa

- **Segundo, utilizaremos una variable intermedia para almacenar la distancia entre las dos partículas y su valor al cubo. La guardaremos como variable de registro, para que sea de acceso rápido. Esta variable sirve simplemente para agilizar brevemente el cálculo. La usaremos para el cálculo de un vector secundario en el que almacenaremos de forma momentánea la fuerza que j relativo ejerce sobre i relativo. Este valor será distinto en cada hilo, pues las variables de registro como hemos dicho, aunque las llamemos igual, se ocupan en distintos sitios de memoria para cada proceso paralelo.**

  Sumaremos a la variable de fuerza de registro, la fuerza de j sobre i. Por otro lado, añadiremos en un paso de cada hilo a la variable de fuerza almacenada en "Shared" la fuerza de i sobre j, que será el valor opuesto de j sobre i.

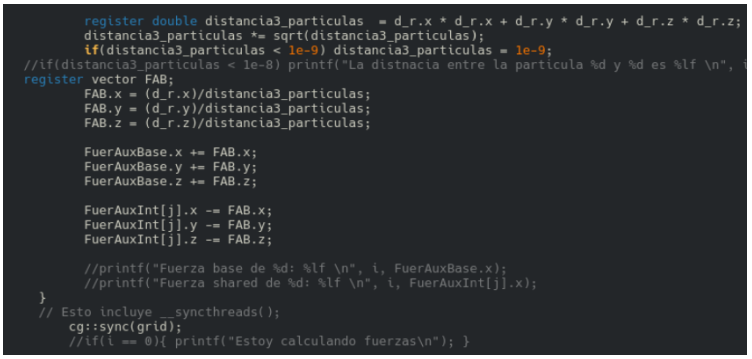

Figure 15: Cálculo de Fuerza en el Programa

6.4 Calculo de Posiciones

El dato de la fuerza total sobre la partícula lo almacenamos finalmente en la memoria "Global", en esta memoria calcularemos las posiciones nuevas de las partículas.

Debido a que cada hilo solo tendrá que realizar un número muy limitado de operaciones, no más de 5 - 6, las realizaremos en la memoria lenta, "Global", ya que ya estaban ahí y moverlas es contraproducente. Simplemente realizaremos las siguientes operaciones:

```
Part[iabs].f.x *= 0.01;
Part[iabs].f.y *= 0.01;
Part[iabs].f.z *= 0.01;

//Calculo de velocidades y posiciones
Part[iabs].v.x += Part[iabs].f.x * dt;
Part[iabs].v.y += Part[iabs].f.y * dt;
Part[iabs].v.z += Part[iabs].f.z * dt;

Part[iabs].r.x += Part[iabs].v.x * dt;
Part[iabs].r.y += Part[iabs].v.y * dt;
Part[iabs].r.z += Part[iabs].v.z * dt;

//Llevo las particulas a su caja
Part[iabs].r.x -= L * floor(Part[iabs].r.x / L);
Part[iabs].r.y -= L * floor(Part[iabs].r.y / L);
Part[iabs].r.z -= L * floor(Part[iabs].r.z / L);

//if(i == 0){ printf("Estoy llevando a todas a su caja \n"); }
cg::sync(grid);
__threadfence();
```

Figure 16: Cálculo de Posiciones en el Programa

La fuerza que hemos calculado previamente es incompleta, ya que falta multiplicarla por el factor Γ. Lo haremos en la memoria "Global" con una simple multiplicación. Después, multiplicamos por los pasos de tiempo correspondientes y obtenemos las nuevas velocidades y posiciones de nuestras partículas.

Por último, falta ver si alguna caja se ha salido de la celda. Para ello mediante la función módulo restaremos o sumaremos a cada coordenada del vector posición el tamaño de la celda para que vuelva a ella. Es análogo a olvidarnos de la partícula que ha salido y considerar una nueva que entra con la misma velocidad y por un sitio opuesto del que ha salido la primera.

6.5 Temperatura del Sistema

Como es lógico, nos interesa conocer la temperatura a la que se encuentra nuestro sistema, por sencillez, calcularemos la velocidad cuadrática media del sistema. Ésta está relacionado con la temperatura de forma unequivoca:

$$< v^2 >= \frac{3KT}{m} \tag{12}$$

De tal manera que:

$$\frac{T_i}{T_0} = \frac{< v^2 >_i}{< v^2 >_0} =< v^2 >_i \tag{13}$$

Resultando que la temperatura relativa en el instante i, no es más que la velocidad cuadrática media de todas las partículas.

En la práctica, al final de cada unidad de tiempo (UT) calcularemos la velocidad cuadrática media. Vamos a crear dos tipos de programas cuya diferencia radica en este punto.

- **Programa de evolución natural : simplemente calcularemos la velocidad cuadrática media y la exportaremos a un archivo externo.**
- **Programa que fuerza el cambio de temperatura : calcularemos la velocidad cuadrática media y después, dividiremos a todas las velocidades entre la raíz cuadrada del valor calculado. Mediante esta operación lo que provocamos es que si la velocidad cuadrática media no fuera uno, le forzamos que lo sea. El dato que guardaremos en esta ocasión será el mismo, la velocidad cuadrática media, sin embargo en este caso no simbolizará lo mismo. En su lugar, este factor representará lo que se ha calentado la simulación en la última unidad de tiempo. Aunque parezca un cambio insignificante, más adelante vemos las implicaciones que conlleva sin perder información de ningún tipo.**

La operación consistirá en un primer momento, en calcular los módulos de las velocidades de las partículas y después simplemente calcular su media. Unos programas se mantendrán ahí, y guardaran el dato, siguiendo así con la siguiente unidad de tiempo.

Los programas que fuerzan el cambio de temperatura dividirán todas las velocidades en la memoria "Global" entre la raiz cuadrada de la velocidad cuadrática media.

6.6 Tiempos de Ejecución

En este apartado vamos a comentar brevemente lo que tarda en ejecutarse nuestro programa en función de varios parámetros, como pueden ser: la cantidad total de partículas o el número de bloques usados junto al número de partículas por bloque. Haremos un pequeño estudio de en qué apartados invertimos más tiempo. Además, será interesante comentar como afectará la elección de un intervalo de tiempo u otro en el tiempo de ejecución.

En la siguiente tabla, mostraremos cuanto se tarda en ejecutar cada parte del programa. El tiempo representará lo que tarda en realizar la maquina, 100 iteraciones de nuestro código sin salir a escribir archivos. Pues despreciamos este tiempo, ya que en los tiempos reales, serán apenas unos milisegundos de escrituras cada varios minutos.

	3025p (19b x 170h/b)	*900p (3b x 300h/b)*	*899p (31b x 29 h/b)*
Programa Vacío	0.07 ms	0.04 ms	0.22 ms
Movimientos memoria	1.67 ms	0.39 ms	0.33 ms
Cálculos Fuerzas	650.5 ms	151.3 ms	118.8 ms
Fuerzas a la memoria Global	35.47 ms	8.4 ms	13 ms
Cálculo de Posiciones y Velocidades	0.7 ms	2 ms	1.8 ms
Mecanismo de Enfriado	3.0 ms	1.7 ms	2.7 ms

Figure 17: Cálculo de Posiciones en el Programa // (b x h/b) equivale a (bloques x hilos por bloque)

Después de ver la tabla, se hace evidente que en lo que invertimos más tiempo es en el cálculo de fuerzas. Justamente es la parte por la que usamos la computación en paralelo.

Si los cálculos que hemos realizado en la memoria "Global" no tardan apenas tiempo en realizarse, es porque el número de operaciones ha sido mínima.

El paso de tiempo elegido es de gran importancia para calcular el tiempo final de computación. Para una misma configuración de partículas cada iteración tardará lo mismo en completarse, independientemente del paso de tiempo introducido.

Sin embargo, si disminuimos el factor de tiempo en un factor de 10, vamos a tener que realizar 10 veces más de iteraciones para realizar un cálculo que llegue hasta la misma unidad de tiempo. Es importante elegir un paso de tiempo cómodo, es decir, que haga que el programa tarde lo menos posible, pero que no haga que pierda demasiada calidad.

7 Resultados

Nota: todas las gráficas que se verán a continuación han sido realizadas gracias al programa xmgrace, y el tratamiento de datos necesario se ha completado mediante programas escritos en C/C++ por mí mismo.

A continuación, vamos a describir los datos que hemos obtenido de la ejecución del programa en sus diferentes variantes. La simulación se ha inicializado con distintas condiciones iniciales y distintos parámetros, como el tamaño de las partículas, el paso de tiempo...

Como hemos mencionado previamente, vamos a hacer las ejecuciones con el programa que realiza la simulación enfriando al plasma en cada unidad de tiempo, puesto que nos da resultados más fáciles de comparar. Nos vamos a centrar en representar histogramas de la velocidad de las partículas, dado que este tipo de gráficas va a depender totalmente de la temperatura del sistema, y veremos que en cada simulación varía de forma distinta. Por lo que para compararlas o representarlas en las mismas imágenes es mucho más cómodo que tengan la misma temperatura. Dedicaremos un apartado a analizar el factor de cambio de energía en el que comentaremos como variaría la temperatura del sistema en caso de "dejarlo libre".

7.1 Primeros Pasos de Tiempo

En el primer instante, todas las partículas se encontrarán con módulo uno, esto no tardará en cambiar, pues en cuanto empiecen a interaccionar variará su energía cinética. Vamos a observar, como sucede este cambio, y si depende de los parámetros controlables que tenemos, como el número de partículas o el intervalo de paso temporal. Según la velocidad con la que se intercambia energía en los primeros pasos, podemos hacer una comparativa de que sistemas tardarán más en llegar al equilibrio.

En primer lugar vamos a realizar una serie de simulaciones con 1860 partículas en las cuales variaremos el paso de tiempo, resultando en las imágenes (18), (19) y (20). En realizar las 50 Unidades de Tiempo (UT) con un paso de tiempo u otro tardaremos distintas cantidades de tiempo, de manera inversamente proporcional con el incremento del paso de tiempo.

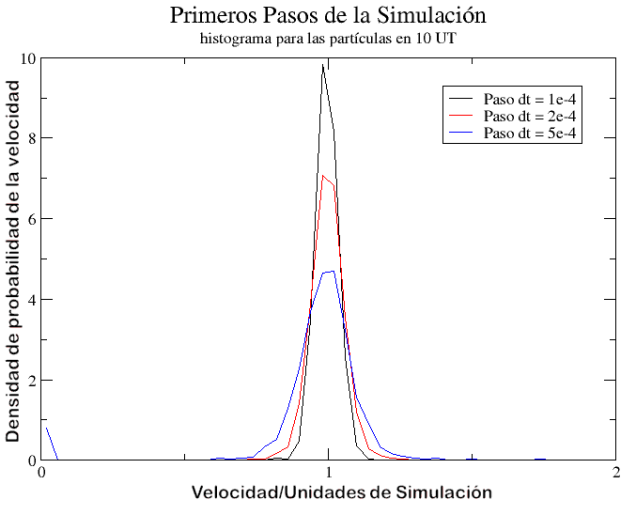

Figure 18: Primeros pasos de la simulación

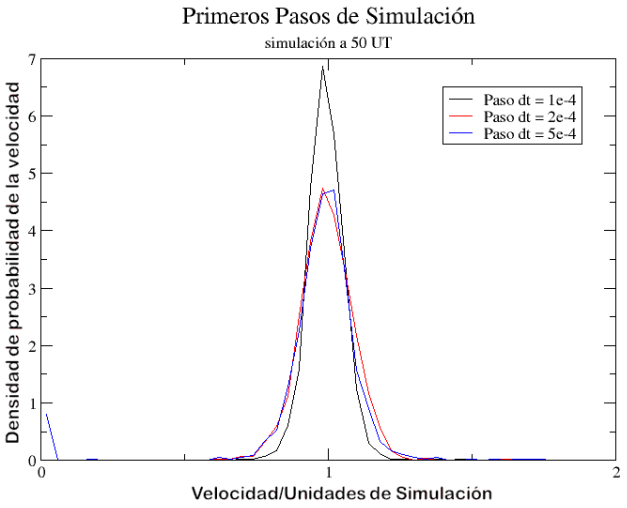

Figure 19: Primeros pasos de la simulación

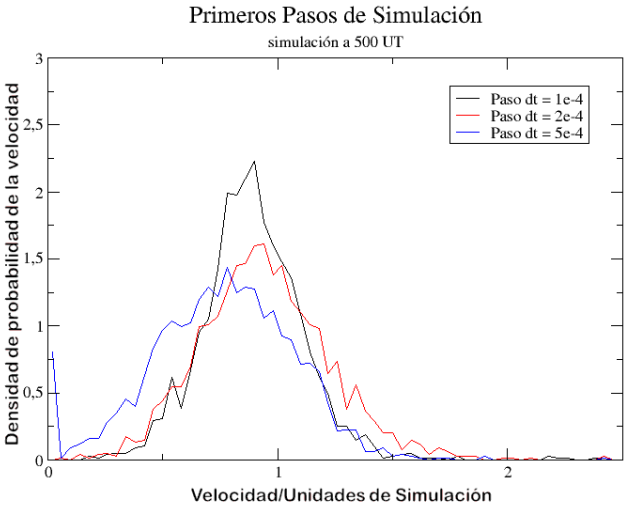

Figure 20: Primeros pasos de la simulación

Las conclusiones que sacamos son que, a pesar de representar las mismas unidades de tiempo, no todas abandonan la delta inicial de la misma manera. El histograma correspondiente a paso de tiempo $5 * 10^{-4}$ se aprecia como está situado por encima del resto continuamente, es más lento a la hora de abandonar la situación inicial, y por tanto tardará más en estabilizarse.

Esto se debe a que en cada paso de interacción, si el paso de tiempo es menor, las partículas recorren una distancia menor. Las fuerzas que se ejercen unas sobre otras tienden a compensar sus errores, siendo a veces más grandes y otras más pequeñas. Provocará un cambio aleatorio en la trayectoria, sin embargo, no nos preocupa ya que de primeras iba a ser aleatorio. Por otro lado, la energía del sistema aumentará de forma indefinida, pues al hacer las cuentas resulta un termino cuadrático que depende del paso de tiempo. Si el paso de tiempo es menor, este termino es menor, y disminuirá el incremento de temperatura en la simulación. Ya analizaremos los incrementos de temperatura en el siguiente apartado.

Cabe destacar, aunque será explicado en un apartado posterior, que en la última imagen se puede apreciar que la velocidad cuadrática media no es la misma en todas las ejecuciones. Es decir, no todos los sistemas tienen la misma energía cinética efectiva. Esto se debe a que en nuestros programas se crean lo que llamaremos balas, no son más que partículas que adquieren una velocidad tres veces superior a la velocidad cuadrática media. Son anomalías que en algunos casos, a largo plazo, llegan a obtener cantidades nada despreciables de la energía del sistema. Provocará que el resto de partículas, no tan rápidas, tengan una velocidad cuadrática media menor. Podríamos dejar de tener en cuenta a las balas como partículas, pero es más interesante analizar por qué aparecen.

El segundo análisis que vamos a realizar es si el hecho de tener una mayor cantidad de partículas provocaría que el sistema evolucione más rápido u no.

La motivación para realizar esta propuesta se basa en el siguiente argumento: al aumentar el número de partículas la aproximación que hace-

mos será menor, pues si hay más partículas la celda será más grande, y tendremos en cuenta a más partículas para la interacción. Puesto que aumenta ese número, es intuitivo pensar que cuántas más interacciones hay, menos tardará a llegar al equilibrio.

En las siguientes figuras, (21), (22) y (23), vemos la evolución descrita, y a diferencia de en el primer caso, las tres simulaciones se ejecutan de forma paralela, sin aparentemente ser ninguna más rápida que otra. Podemos deducir que el tiempo de llegar al equilibrio no depende del número de partículas, al menos de una manera considerable.

Como vemos en la última imagen, la (23), el histograma correspondiente a 500 partículas es radicalmente distinta, su temperatura media no se acerca a la unidad. El origen del problema vuelven a ser las balas.

A efectos prácticos es como la simulación correspondiente a 500 partículas y la de 3025 partículas estuvieran a distintas temperaturas. Vemos que son curvas situadas en lugares distintos y es más difícil comparar su forma en lugar de si las dos estuvieran dibujadas en el mismo lugar, como las correspondientes a 1860 y 3025 partículas.

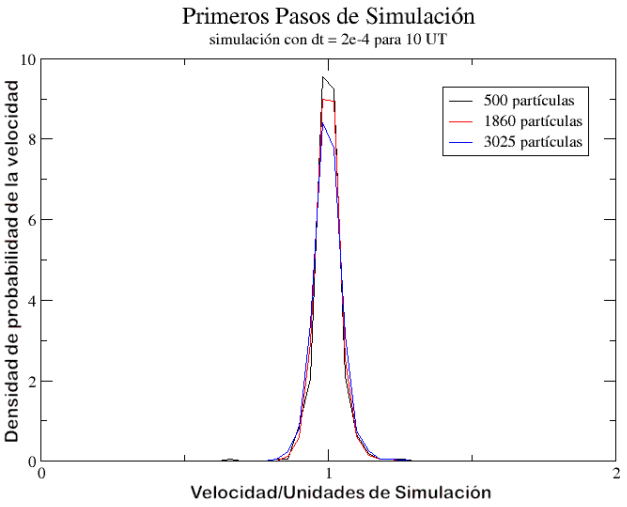

Figure 21: Primeros Pasos de Simulación

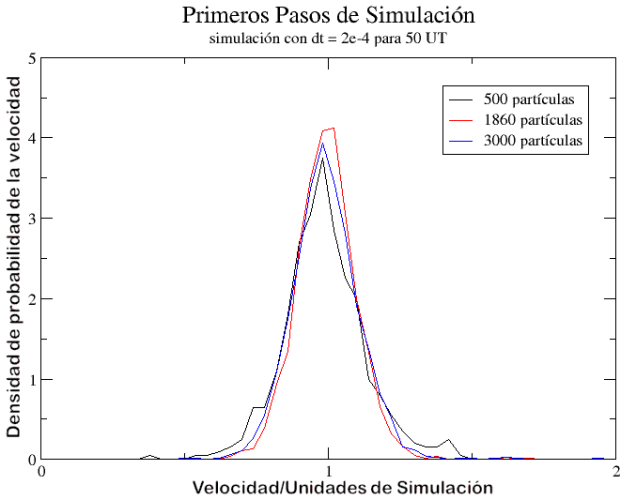

Figure 22: Primeros Pasos de Simulación

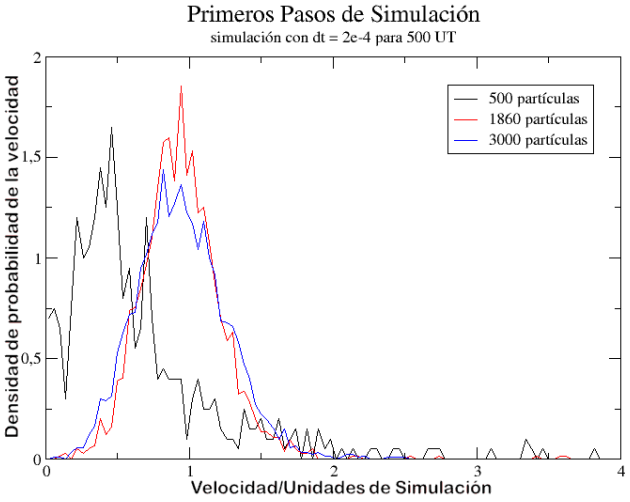

Figure 23: Primeros Pasos de Simulación

7.2 Temperatura del Sistema

El factor de cambio de temperatura de la simulación irá cambiando. Si su valor es muy alto indica que la temperatura en la unidad de tiempo que se ha calculado se habría calentado más que si su valor es cercano a la unidad. A su vez, puede ser menor que la unidad, de manera que representaría un enfriamiento del sistema.

Como hemos visto en la ecuación (14), la temperatura relativa a la temperatura correspondiente a la velocidad cuadrática media unidad, será simplemente la velocidad cuadrática media en el instante que queremos medirla.

La energía cinética del sistema también tiene una relación biyectiva con la velocidad cuadrática media, por lo que se dará la misma situación que con la temperatura. Así pues, podremos ver el factor de cambio de temperatura como el factor de cambio de velocidad cuadrática media o el factor de cambio de energía cinética del sistema. Siendo todos los anteriores factores equivalentes tanto en sentido físico como matemático, por lo que utilizaremos los tres indistintamente.

Como medimos este factor de cambio cada unidad de tiempo no dejamos al sistema suficiente libertad como para que sea muy elevado. Nos encontraremos con valores cercanos a la unidad, como podemos ver en la imagen (24).

Tras la realización de todas las simulaciones, vamos a analizar de que factores depende la variación de temperatura en el sistema. La primera fuente de incremento de este factor será la magnitud de los pasos de tiempo.

A continuación, compararemos los factores de enfriamiento para simulaciones con el mismo número de partículas y la misma constante de interacción, pero con pasos de tiempos distintos. Es de esperar que cuanto más pequeño sea el paso de tiempo, más cercano a la unidad sea el factor de enfriamiento, como vemos en la imagen (24).

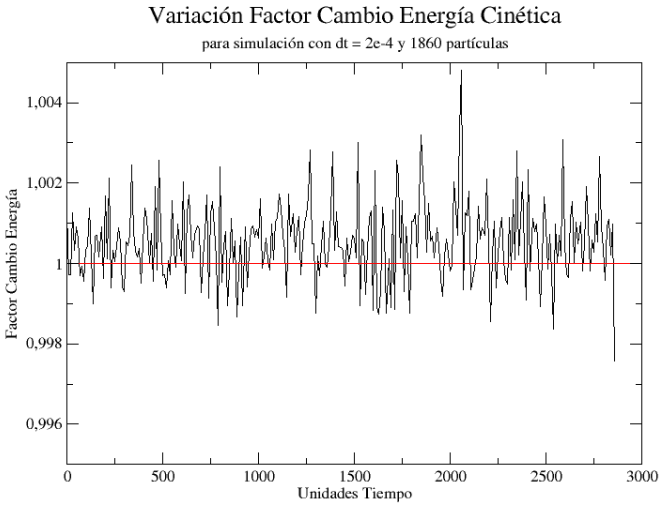

Figure 24: Valores de renormalización de energía

Claramente, en la figura (25) vemos cómo sucede lo que hemos descrito. Es relevante recordar que el definir pasos de tiempo muy pequeños, eleva el tiempo de computo de maneras considerables, antes de ejecutar el código hay que hacer un balance de que compensa más. Una simulación que tarde poco tiempo y una "calidad" peor, en la que se está obligado a introducir un mecanismo de enfriamiento o una simulación más fiel a la realidad en la que no haya un calentamiento artificial con la desventaja de ser mucho más lenta.

Si sabemos que el plasma que estamos simulando se va a calentar hasta un equilibrio y queremos saber cómo y cuánto, se tendrá que realizar un estudio de cuanto se calienta por culpa de los pasos temporales elegidos en esa configuración para el plasma. Posteriormente elegir un paso de tiempo que nos ha proporcionado una variación de temperatura nula o despreciable en lo que el sistema se equilibra.

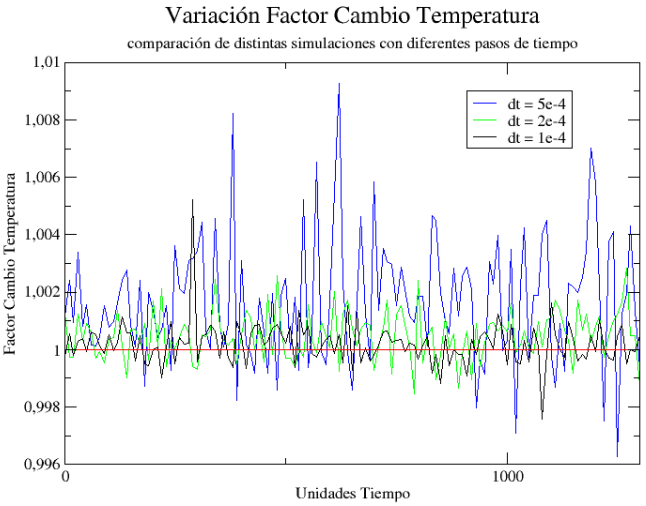

Figure 25: Variación renormalización energía para distintos pasos de tiempo

El siguiente resultado corresponderá a realizar dos simulaciones, cada una con un factor de interacción entre las partículas distinto. Todos los demás parámetros de la simulación son iguales en ambas, resultando la figura (26).

Cuando aumentamos la constante de interacción, aumenta de forma drástica la variable que utilizamos para enfriar el programa, es decir, la temperatura aumenta más rápidamente. Si necesitamos que nuestro parámetro de interacción sea grande, tendremos que ser consecuentes y utilizar intervalos de tiempo más finos en compensación. Por otro lado, si usamos factores Γ muy pequeños, nos da una ligera libertad de ampliar el paso de tiempo.

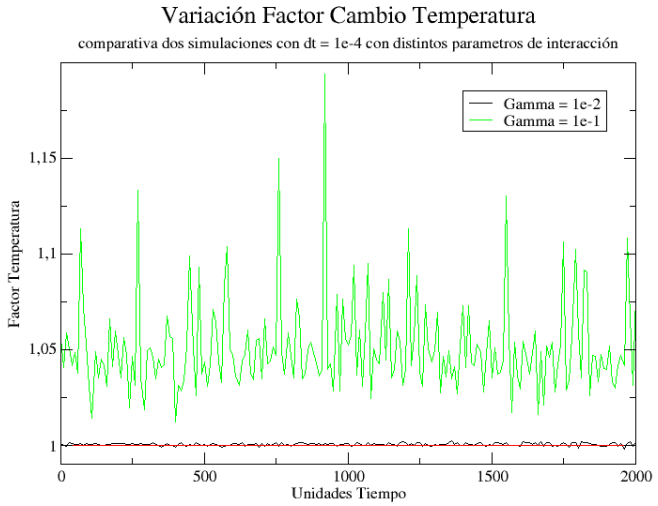

Figure 26: Valores de renormalización de energía para distintas Gammas

Hay que tener cuidado, el par paso de tiempo y factor de interacción solo afecta a la velocidad de la partícula, sin embargo, para el cálculo de posiciones en cada instante solo utilizaremos el paso de tiempo. No podemos elegir pasos de tiempo que nos estropeen el cálculo de las posiciones aunque no nos estropeen las velocidades gracias al factor de interacción.

Por último, en este apartado, será en el único en el que hablaremos de las ejecuciones con el programa que no enfría, si no que permite una evolución natural, pues lo que veremos en él será lo que aparece en la figura (27).

Lo único que vamos a ver será que la temperatura aumenta de forma progresiva. Si que es verdad, que cuánto más grande sea el paso de tiempo la temperatura crece más rápidamente. Es posible encontrar pasos de tiempo donde el incremento de temperatura artificial, es decir culpa del paso de tiempo, sea despreciable a lo largo del programa, sin embargo,

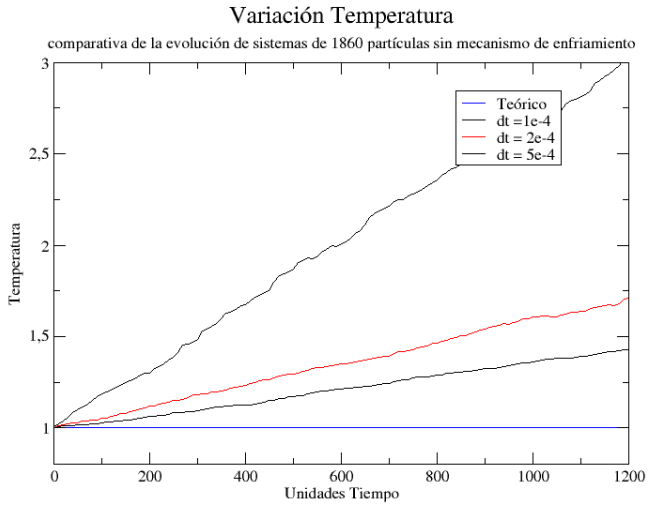

Figure 27: Valores de la Temperatura sin mecanismo de enfriamiento

alarga el proceso de simulación bastante. Si nuestro objetivo es realizar simulaciones con parámetros concretos y un estudio más concienzudo de los resultados, si que interesaría buscar y usar esos pasos de tiempo. Sin embargo, nosotros estamos observando que sucede cuando variamos los parámetros, nos interesa poder realizar simulaciones rápidas, por lo que introducimos el mecanismo de enfriado y elegimos pasos más gruesos, como los de la figura (27).

De todos los resultados anteriores extraemos que no hay incremento de temperatura que no se deba al efecto de la discretización (que se va reduciendo a medida que el paso de tiempo se hace más pequeño), es decir, un error numérico por el hecho de haber introducido ecuaciones de diferencias.

La conclusión final de este apartado será que no tenemos que perder de vista cuál es nuestro objetivo, si no nos importa forzar un mecanismo de enfriado, no hay mayor problema en que el paso de tiempo haga que el factor de enfriamiento sea algo mayor que la unidad, mientras no provoque otros problemas.

Si no queremos forzar mecanismos sino que evolucione de forma natural, necesitaremos un paso de tiempo cuyo factor de enfriamiento se mantenga constante en el equilibrio del sistema.

7.3 Balas

A continuación, vamos a describir lo que es una de las dificultades de nuestro programa: las balas. Éstas parecen ser incontrolables salvo que se aumente de forma excesiva el tiempo de computación, u obligaría a introducir aproximaciones que son aparentes nuevas fuentes de error en la física de la simulación.

A lo que nos referimos cuando hablamos de "Balas" es a partículas que tienen un valor de velocidad cuadrática mayor a tres veces la velocidad cuadrática media del sistema total, es decir, partículas que son extremadamente rápidas.

En otro tipo de simulaciones de este campo, aparentemente surgen, pero dejan de interactuar con el resto de las partículas, adquieren una velocidad y dejan de intercambiar energía con su entorno. Sin embargo, en nuestra simulación, estas balas siguen adquiriendo velocidad, surgen nuevas balas, y se pierden otras. Cuando el sistema está en equilibrio, 10 balas dentro de 2000 partículas totales, pueden llegar a adquirir un 30% de la energía total.

El número de balas, y sobretodo, la energía "perdida" en ellas depende del paso de tiempo utilizado. Cuánto menor sea, menos balas se producirán, mientras que cuanto más grueso sea, el efecto será el opuesto.

La tabla de la figura (28) es comparativa entre las balas creadas en distintas simulaciones.

Balas producidas	*500 UT*		*1000 UT*		*1500 UT*	
	Nº balas	% energía	Nº balas	% energía	Nº balas	% energía
paso de tiempo = 1e-4	2	11.9 %	5	13.7 %	3	14.0 %
paso de tiempo = 2e-4	1	1.25 %	1	45.2 %	6	61.16 %
paso de tiempo = 5e-4	6	19.9 %	7	37.7 %	6	34.85 %

Figure 28: Tabla comparativa para distintos dt con simulaciones de 1860 partículas

Ocurre lo que habíamos predicho, por lo que podremos buscar una explicación al origen de las balas. Lo que sucede es que las partículas por azar o de forma anomalística se juntan demasiado, lo que provoca que la fuerza de una sobre la otra sea enorme, esta es la semilla de una bala. Si se repite en los siguientes pasos, se crearan partículas con una velocidad excesivamente grande. Una vez que existe una sola bala, como el paso de tiempo no está diseñado para que las partículas tengan tanta velocidad, será muy fácil que a raíz de ella nazcan nuevas balas.

Si el número del paso de tiempo es muy pequeño es más difícil que las partículas lleguen a estar muy cerca, es más improbable la creación de balas. Mientras que los pasos más gruesos favorecerán su producción.

Estamos haciendo una simulación en la que hacemos una aproximación que consiste en acotar la fuerza máxima que puede recibir una partícula. Si las partículas están a una distancia menor a 10^{-4}, no sufrirán fuerzas mayores a la que describe la ecuación (2).

Vamos a acotar aún más la fuerza, de manera análoga a lo que ya hacíamos, pero para distancias de 10^{-3}. En principio, esta limitación solo afectará a las partículas que fueran a producir balas, es decir, seguirá siendo una cantidad de interacciones totalmente despreciables, por lo que no tendremos en cuenta que estamos haciendo esta aproximación para otros resultados.

Balas producidas	*500 UT*		*1000 UT*		*1500 UT*	
	Nº balas	% energía	Nº balas	% energía	Nº balas	% energía
Tamaño de la partícula igual a 1e-4	6	19.9 %	7	37.7 %	6	34.85 %
Tamaño de la partícula igual a 1e-3	2	2.03 %	2	2.08	1	1.5 %

Figure 29: Tabla comparativa para distintos tamaños de partículas con simulaciones de 1860 partículas con dt = 5e-4

Balas producidas	500 UT		1000 UT		1500 UT	
	Nº balas	% energía	Nº balas	% energía	Nº balas	% energía
Tamaño de la partícula igual a 1e-4	2	11.9 %	5	13.7 %	3	14.0 %
Tamaño de la partícula igual a 1e-3	0	0	0	0	0	0

Figure 30: Tabla comparativa para distintos tamaños de partículas con simulaciones de 1860 partículas para dt = 10^4

Además, vamos a comparar si la cantidad de balas puede depender del parámetro de interacción, en la tabla (31) vemos el resultado.

Balas producidas	500 UT		1000 UT		1500 UT	
	Nº balas	%energía	Nº balas	%energia	Nº balas	%energía
Gamma = 0.01	0	0	0	0	0	0
Gamma = 0.1	8	12%	5	8.7%	6	8.6%

Figure 31: Tabla comparativa para distintos tamaños de partículas con simulaciones de 1860 partículas, con dt = 5e-4 y con tamaños de partículas de 1e-3

Como era de esperar, vemos que al aumentar Γ aumentaremos el número de balas si no variamos ningún otro parámetro. La fuerza promedio será mayor en cada partícula, será equivalente a aumentar el paso de tiempo.

Como vemos en todas las tablas de este apartado, tanto disminuyendo el paso de tiempo como aumentando el paso de la partícula, podemos evitar que aparezcan balas, sin embargo son dos hechos separados que producen distintos efectos en la simulación.

Por un lado, disminuir el paso de tiempo, recrea una situación más fiel a la realidad, sin embargo eleva el tiempo de cálculo muchísimo.

Por otro lado, aumentar la distancia de las partículas para limitar la fuerza que se ejercen las unas a las otras provoca que podamos tener problemas, pues estamos haciendo que la fuerza deje de ser conservativa. Sin embargo, siempre y cuando el tamaño que supongamos a la partícula, sea despreciable frente a las distancias típicas del sistema no habrá mayor inconveniente. Si el tamaño es tan pequeño, que las partículas solo llegan a "chocar" cuando se producen anomalías, no estaríamos haciendo aproximaciones que estropeen la física de la simulación.

7.4 Situación de Equilibrio

Este apartado podría considerarse el principal de nuestro trabajo, pues vamos a comprobar si el estado de las velocidades que tienen las partículas de nuestra simulación es equivalente a un estado de equilibrio teórico.

El equilibrio teórico al que nos referimos viene dado por la fórmula (11), procedente de una curva Maxwell-Boltzmann, teniendo en cuenta que la velocidad cuadrática media es igual a la unidad y que esta expresión ya está normalizada.

Vamos a exponer distintas gráficas de nuestras simulaciones, mostrando con distinto número de partículas como resultarían las curvas. Debido a que es el único parámetro controlable que cambiará la forma de las curvas. El paso de tiempo afectará a los cambios de temperatura, sin embargo forzamos el enfriamiento, por lo que no afectará a los resultados.

Por culpa de la existencia de las balas, la temperatura de nuestras partículas, sin tener en cuenta a las balas, será una fracción de la energía total. Tendremos que adaptar la ecuación (11) ajustando mediante la temperatura relativa. De esta forma, que las curvas que mostramos, no tienen que ser correspondientes con velocidad cuadrática media unidad, sino que dependerá de las balas que se hayan creado en esa ejecución en concreto.

En primer lugar, la curva que conlleva la fórmula (11) sería replicable únicamente teniendo un número infinito de partículas con las que realizar la simulación. Sin embargo, y estando muy lejos de esa ficticia realidad, tendremos que soportar cierto ruido estadístico, en el cual, sin entrar en mucho detalle, vamos a mostrar como cambia la cantidad de él ante las diferentes cantidades de partículas.

La figura (32) que vemos a continuación trata de la representación en la situación de equilibrio del cálculo correspondiente a 500 partículas. Tiene una cantidad de ruido estadístico enorme, pero si nos fijamos un poco tratando de ir a la figura global cuadra con la curva teórica.

La gráfica de la figura (33), correspondiente a 1860 partículas, es algo intermedia, mientras que la imagen (34), es causada por 3025 partículas, se ajusta bastante bien a la curva teórica. Es esta una de las razones, por las que necesitamos o buscamos aumentar el número de partículas, pues aumenta bastante la precisión en este tipo de datos.

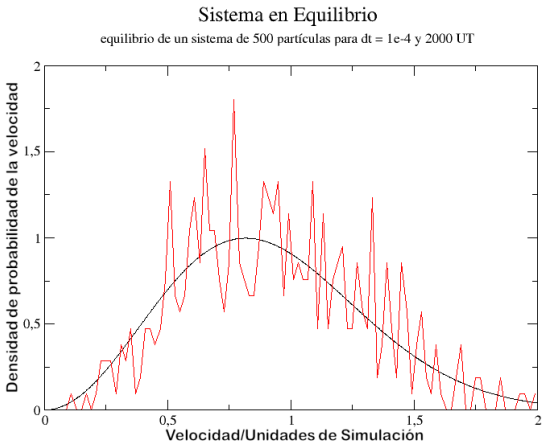

Figure 32: Tabla de equilibrio para 500 partículas

En la gráfica (34) vemos cual es la dificultad que nos han añadido las balas, la temperatura efectiva del sistema cambia, los distintos histogramas que resultan en las distintas unidades de tiempo no se ajustan a las mismas curvas de equilibrio. Lo que nos impide comparar de forma clara lo que sucede en este tipo de simulaciones.

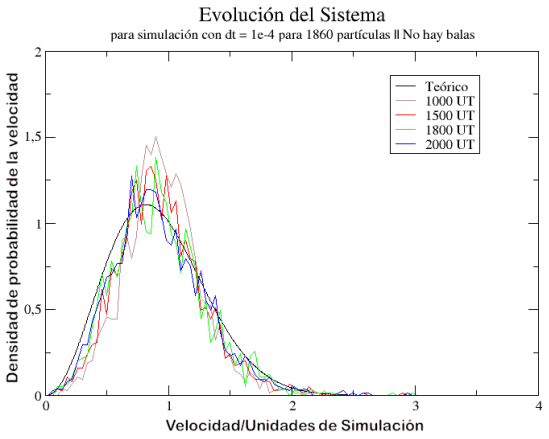

Figure 33: Tabla de equilibrio para 1860 partículas

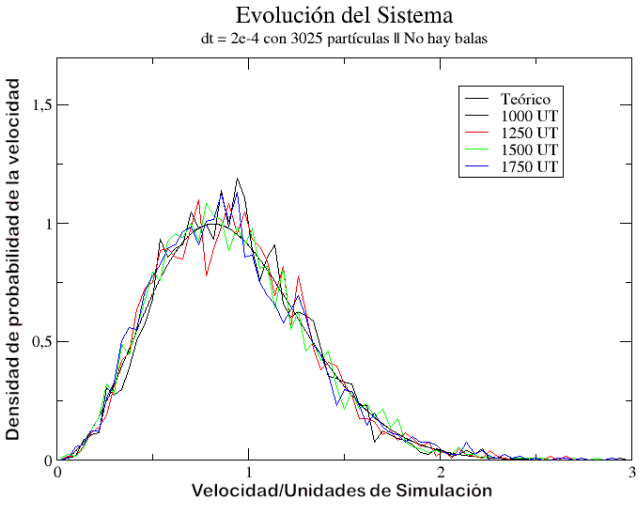

Figure 34: Tabla de equilibrio para 3025 partículas

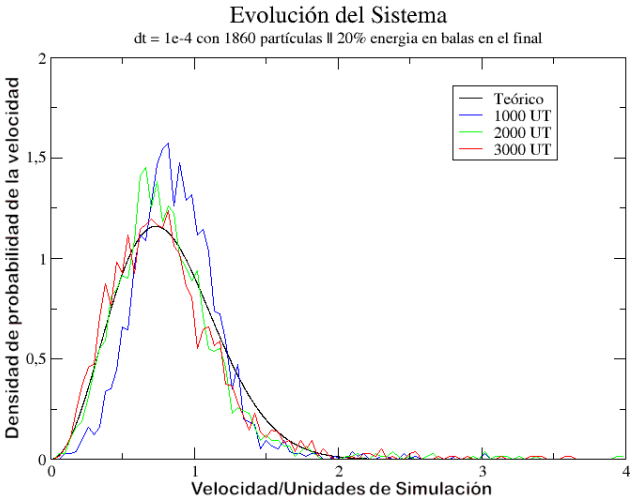

Figure 35: Tabla comparativa para distintos tamaños de partículas con simulaciones de 1860 partículas con dt = 5e-4

8 Conclusiones

Se ha conseguido realizar una simulación que, a pesar de las dificultades tratadas en este trabajo, es capaz de emular las interacciones en las que intervienen un conjunto de partículas cargadas todas de forma uniforme. Además, hemos aprovechado las ventajas que nos ofrecen las nuevas versiones de CUDA 9.0^{TM}. Hemos implementado alguna de las mejoras ofrecidas por el software de manera que tenemos la capacidad de realizar la simulación en más de un bloque.

Tener la capacidad de utilizar varios bloques para la simulación de un mismo conjunto de partículas tiene sus ventajas, como tener más potencia de cálculo y ser capaz de tener una mayor cantidad de partículas. Sin embargo, no todo son mejoras, puesto que son ejecuciones que tardan mucho más tiempo en realizar los cálculos, tanto por aumentar el número de partículas, como por los movimientos de memoria que son necesarios. Además, si utilizamos solo un bloque, los demás están libres, por lo que podemos ejecutar distintos programas en ellos, y así tener varias simulaciones activas simultáneamente. En nuestro caso, al usar hasta 30 bloques para una única tarea, no tenemos la misma capacidad.

Por otro lado el TFG, motivado por el mismo alumno, ha sido de gran ayuda para el entendimiento de cómo funciona una simulación física real, de igual manera que de saciar la curiosidad en este campo. Ha conseguido fomentar una capacidad de manejo, aunque básico, de código de programación en paralelo y de su implementación en programas. La física usada, a pesar de ser sencilla, se le ha dado alguna vuelta que puede ser interesante, como el imaginar un proceso real marcado por una ecuación diferencial pasado a una ecuación en diferencias, viendo sus repercusiones.

9 Tareas Propuestas

Para proseguir con la línea de trabajo que encamina el TFG en uno u otro aspecto, se proponen los siguientes puntos:

- **En primer lugar, sería destacable el hecho de seguir experimentos que hemos realizado en nuestro TFG, sin limitaciones de tiempo, de forma que se pueda simular con muchas más partículas, con pasos de tiempo minúsculos e ir probando con lo que pueda surgir para arreglar algunos de los problemas que hemos tenido, como la producción de balas.**

- **Muchos de los procesos nombrados en las anteriores páginas pueden utilizarse para otros tipos de programas completamente distintos, de hecho, aunque no sean de simulaciones físicas, mientras sean programas que utilicen la tecnología que hemos ido mencionando, y busquen la eficiencia en la computación en paralelo.**

- **Experimentar con el máximo de partículas que se pueden ejecutar en los bloques para sacar el mayor rendimiento a la simulación, tanto en la gráfica que hemos usado, como en otras, ya que no todas tienen las mismas prestaciones.**

- **El uso de varios dispositivos interconectados para cualquier tipo de simulación. Hablamos del siguiente paso de mejora en la cantidad de hilos ejecutables mediante la ejecución del mismo código en distintas GPUs. De esta manera, que se introducen en la misma placa base, de manera que las gestiona la misma CPU. Se podrán comunicar de igual manera que ahora lo hacemos, con distintos comandos y teniendo algo más de cuidado. Para fijarnos si nuestra GPU sería capaz de ello, tendremos que ir al deviceQuery que ya hemos introducido y comprobar si admite el "MultiDevice Co-op Kernel Launch".**

10 Bibliografía

- **CUDA Toolkit Documentation v10.1.168 (y versiones anteriores) https://docs.nvidia.com/cuda/index.html**

- **© 2006-2010 NVIDIA Corporation. NVIDIA CUDA Programming Guide Version 3.0.**

- **M. A. Gigosos, D. González-Herrero, N. Lara, R. Florido, A. Calisti, S. Ferri, B. Talin. Classical molecular dynamics simulations of hydrogen plasmas and development of an analytical statical model for computational validity assessment. Physical Review E 98, 033307 (2018).**

- **Khoirudin y Jiang Shun-Liang. 2015. GPU APPLICATION IN CUDA MEMORY. Advanced Computing: An International Journal (ACIJ), Vol.6, No.2, March 2015**

- **Jiwei Liu. B.S. 2010. EFFICIENT SYNCHRONIZATION FOR GPGPU. Zhejiang University, Ph.D. in Electrical Engineering.**

- **Isaac Gelado y Michael Garland. 2019. Throughput-Oriented GPU Memory Allocation. PPoPP '19, February 16–20, Washington, DC, USA.**

- **R.A.Fonseca, L.O.Silva, F.S.Tsung, V.K.Decyk, W.Lu, C.Ren, W.B.Mori, S.Deng, S.Lee, T.Katsouleas, and J.C.Adam. OSIRIS: A Three-Dimensional, Fully Relativistic Particle in Cell Code for Modeling Plasma Based. Springer-Verlag Berlin Heidelberg 2002.**

- **John M. Dawson. Particle Simulation of Plasmas. Department of PHysics, University of California, Los Ángeles, California. 2 Abril 1983.**

- **Mary Thomas. COMP 605: Introduction to Parallel Computing Lecture: CUDA Shared Memory. Department of Computer Science, San Diego University. 25 de Abril de 2017.**

Nota: las siguientes páginas web, consistentes en foros con una comunidad que se dedica a solucionar problemas de otros usuarios, y a enseñar los conocimientos necesarios, si bien no se pueden llamar "referencias bibliográficas" en el más estricto sentido, han sido de ayuda para la realización de este trabajo de fin de grado, por lo que se merecen una cita.

- **https://stackoverflow.com/**
- **https://github.com/**
- **https://www.nvidia.com/es-es/about-nvidia/communities/**
- **https://physics.stackexchange.com/questions/55665/plasma-stability#**