Implementación de un código de Dinámica Molecular en serie y en paralelo usando el módulo MPI

Oscar Fajardo, Xavier Marugán, Xabier Méndez, Alberto Pla, Sergi Roca, Sotiris Samatas, Elena Sesé i Lorena Vega

Jueves 30 de Marzo

1. Inicialización del sistema

El sistema se inicializa colocando las partículas en los nodos de una red cúbica simple. Para conseguir esto, primero definimos el máximo número de partículas que habrá en una dimensión (a lo largo de una aresta), lin_dim, con la siguiente línea de código en fortran:

$$lin\_dim = ceiling(particles * *(1{,}0/3{,}0)) \tag{1}$$

donde $particles$ se refiere al nuúmero de partículas que tenemos. Con esto ya podemos calcular la distancia enre partículas, dr:

$$dr = L/(lin\_dim + 1) \tag{2}$$

donde L es la longitud de la aresta del cubo en el cual se encontrarán las partículas.

A continuación, para poder trabajar con varios procesadores en paralelo, distribuimos las partículas que cada uno tendrá según:

$$partxproc = nint(real(particles)/real(numproc)) \tag{3}$$

donde $numproc$ es el número de procesadores que tenemos y $partxproc$ será el número de partículas que cada uno tendrá menos el último al cual se le asignarán todas las partículas que sobren ($particles - partxproc(numproc - 1)$). Hace falta mencionar que la ecuación (3) está pensada para trabajar con muchas más partículas que procesadores ya que para pocas partículas (por ejemplo 30 partículas y 12 procesadores) el penúltimo procesador podría quedarse incompleto y esto produciría un error (esto se podría arreglar utilizando la función $floor$ en lugar de $nint$ en la ecuación pero en algunos casos podría llevar a tener una configuración espacial de las partículas "menos cúbica").

Para que cada procesador pueda trabajar con sus partículas asignadas, colocando cada una de ellas en el nodo que le toca independientemente de los otros procesadores, hace falta encontrar un índice "global" de partícula que dependa del número de procesador (*rank*) y de la variable *lin_dim* definda antes. Tal índice se define como:

$$N = partxproc * rank + i \tag{4}$$

Con i tomando valores desde 1 hasta *partxproc* para todos los procesadores menos el último, donde i va desde 1 hasta $particles - partxproc(numproc - 1)$ (el número de partículas asignadas al último procesador). De esta manera, pasando por todos los posibles valores de i (con un bucle *do*), N recorre todas las partículas y cada procesador tendrá sus propios valores de N (partículas).

Ahora ya podemos asignarle a cada partícula su posición x, y y z en función de su identidad (N):

$$x = \begin{cases} mod(N, lin\_dim) & si \quad mod(N, lin\_dim) \neq 0 \\ \\ lin\_dim & si \quad mod(N, lin\_dim) = 0 \end{cases} \tag{5}$$

$$y = \begin{cases} mod(\frac{N-1}{lin\_dim+1}, lin\_dim) & si \quad mod(\frac{N-1}{lin\_dim+1}, lin\_dim) \neq 0 \\ \\ lin\_dim & si \quad mod(\frac{N-1}{lin\_dim+1}, lin\_dim) = 0 \end{cases} \tag{6}$$

$$z = (N-1)/(lin\_dim * *2) + 1 \tag{7}$$

Como resultado de este procedimiento tendremos un vector posición con componentes enteras para cada partícula (por ejemplo la primera partícula estará en (1, 1, 1), la segunda en (2, 1, 1), etc...). Multiplicando cada vector por dr y restando $L/2$ acabaremos con el sistema centrado en (0, 0, 0) y con una estructura cúbica simple. Se puede introducir aleatoriedad en las posiciones iniciales sumandoles una perturbación pequeña aleatoria. Esto se consigue creando un vector aleatorio (*displacement*) cuyas componentes se generan según una distribución uniforme en el rango: $(-\frac{3dr}{4}, \frac{3dr}{4})$. Análogamente, se pueden generar velocidades iniciales aleatorias con componentes dentro del rango: $(-\frac{dr}{6}, \frac{dr}{6})$ (se escogen velocidades pequeñas para asegurar que las particulas no se acerquen demasiado). En el caso de que el usuario escoja inicializar el sistema con velocidades iniciales aleatorias, se les resta la velocidad del centro de masas del sistema para asegurar que el momento total se conserva.

Por último, cuando ya se acaba todo el cálculo de posiciones y velocidades iniciales, cada procesador envía esta información al procesador $MASTER$, el cual se encarga de escribir las posiciones y velocidades iniciales en sus ficheros correspondientes y reenviar las matrices de posicion y velocidad actualizadas a todos los procesadores.

Finalmente, comparamos el programa hecho en serie y hecho en paralelo del mismo algoritmo para estudiar cuándo conviene utilizar uno o el otro. La figura

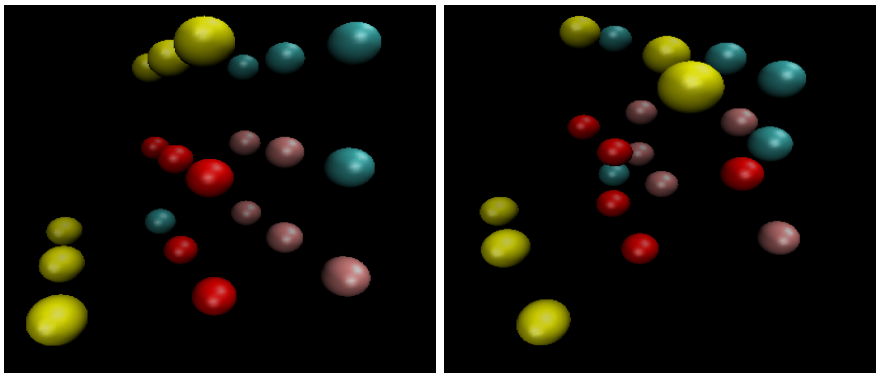

Figura 1: Ejemplo de la configuración inicial ordenada (figura de la izquierda) y aleatoria (figura de la derecha) de las partículas un sistema de 21 particulas. El programa se ha ejecutado con 4 procesadores representados por los 4 colores.

2 muestra los tiempos que ha tardado el programa en ejecutarse en cada caso para diferentes valores del numero de partículas. Podemos ver que a partir de 10000 particulas ya va más rapido el programa en paralelo.

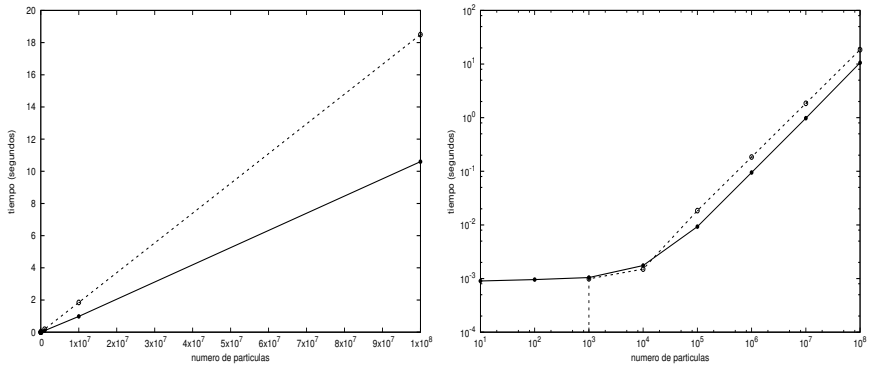

Figura 2: Representación gráfica del tiempo que tarda en ejecutarse el programa en función del número de partículas en escala lineal (figura de la izquierda) y logarítmica (figura de la derecha) para el programa en serie (línea discontínua) y el programa en paralelo (línea contínua).

La figura 3 muestra los tiempos de cálculo y de actualización para el programa en paralelo. El tiempo de actualización se refiere al tiempo que tarda el programa en enviar y recibir la información necesaria entre los procesadores.

Haciendo un estudio de cómo varía el tiempo de cálculo y actualización con el número de procesadores utilizados vemos que el tiempo ganado no es proporcional al número de procesadores utlilizados, de acuerdo con la ley de Amdahl.

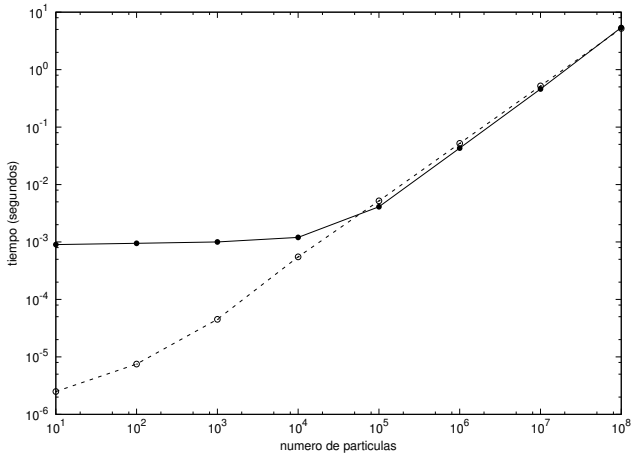

Figura 3: Representación gráfica del tiempo de cálculo (línea discontínua) y del tiempo de actualización (línea continua) como función del número de particulas en escala logarítmica.

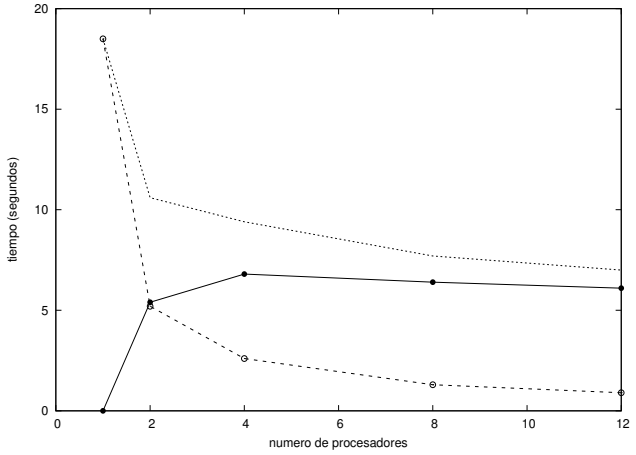

Figura 4: Tiempo de cálculo (línea discontínua), tiempo de actualización (línea contínua) y tiempo total (línea punteada) para el programa de inicialización en función del número de procesadores utilizado.

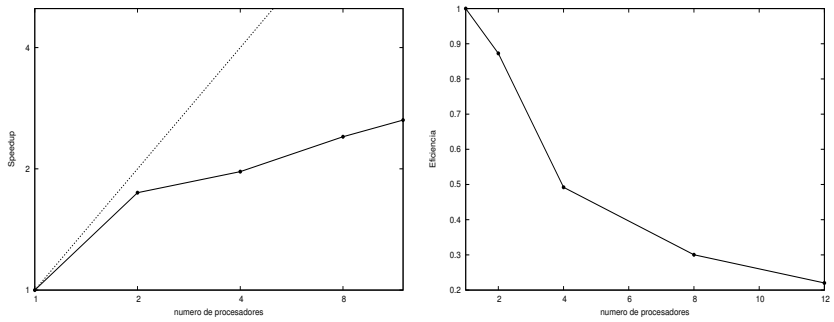

Figura 5: Figura de la izquierda: speedup respecto al caso ideal (línea discontínua). Figura de la derecha: eficiencia del programa.

2. Cálculo de Fuerzas: Lennard-Jones

Hemos implementado la interacción de Lennard-Jones para calcular las fuerzas entre pares de partículas.

$$\mathbf{F}_{LJ}(r) = 24\epsilon \left(2\frac{\sigma^{12}}{r^{13}} - \frac{\sigma^6}{r^7} \right) \hat{r} \tag{8}$$

donde $\epsilon = 1$ y $\sigma = 1$.

El cómputo de las fuerzas en serie se reduce a calcular la distancia entre los pares de partículas, aplicando condiciones periódicas de contorno, y usar la Eq. (8) para obtener el valor de la fuerza que recibe cada partícula.

Para realizar el algoritmo en paralelo hemos divido las partículas por procesadores, lo que se conoce como *Atom decomposition algorithm*. Es decir, hemos repartido un número igual de partículas para cada procesador, Fig 6. Cuando el número de partículas no sea divisible entre el número de procesadores, el último procesador se quedará con las partículas de más, hasta la última.

$$G_{i,j} = \begin{pmatrix} \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \triangle & \triangle & \triangle & \triangle & \triangle & \triangle & \triangle & \triangle \\ \triangle & \triangle & \triangle & \triangle & \triangle & \triangle & \triangle & \triangle \\ \diamond & \diamond & \diamond & \diamond & \diamond & \diamond & \diamond & \diamond \\ \diamond & \diamond & \diamond & \diamond & \diamond & \diamond & \diamond & \diamond \\ \star & \star & \star & \star & \star & \star & \star & \star \\ \star & \star & \star & \star & \star & \star & \star & \star \end{pmatrix}$$

Figura 6: Separación de partículas por procesador. En este caso tenemos 8 partículas y 4 procesadores. Cada símbolo representa las interacciones que calculará cada procesador.

2.1. Método 1: *Atom decomposition* algorithm

Trabajando con N partículas y 3 dimensiones, cada *worker* calculará la interacción entre el número de partículas que se le han asignado y las N partículas del sistema. Ésta será la información que cada *worker* le pase al procesador master, que finalmente obtendrá una matriz de fuerzas NxNx3, $\mathbf{G}_{ij}$. Esta matriz contiene la fuerza de cada partícula del sistema con todas las restantes. Una vez toda la información esté en el master, hay que sumar, en nuestro caso el segundo índice, para obtener la fuerza que cada partícula recibe por acción del resto, obteniendo así una matriz de fuerzas Nx3, $\mathbf{F}_k$. A continuación, se procederá a enviar esta matriz a todos los *workers*. El proceso finaliza cuando todos los *workers* reciben la matriz entera actualizada.

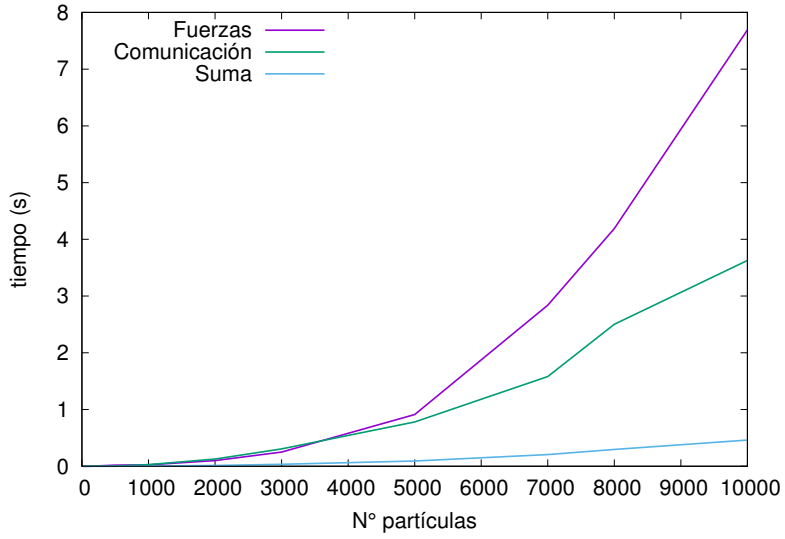

Figura 7: Tiempo de cálculo de las fuerzas, de la comunicación entre procesadores y de la suma de interacciones en función de las partículas del sistema, fijando 8 procesadores, según el método 1.

Esquemáticamente, el algoritmo utilizado es:

1. Separación de partículas por procesadores.
2. Envío de cada trozo de la matriz $\mathbf{G}_{ij}$ de los *workers* al master.
3. El master recibe toda la matriz $\mathbf{G}_{ij}$.
4. El master calcula la fuerza total por partícula $\mathbf{F}_k$ (Eq. 8).
5. Envío de la matriz $\mathbf{F}_k$ desde el master a todos los demás procesadores.
6. Todos los procesadores reciben la matriz $\mathbf{F}_k$ del master.

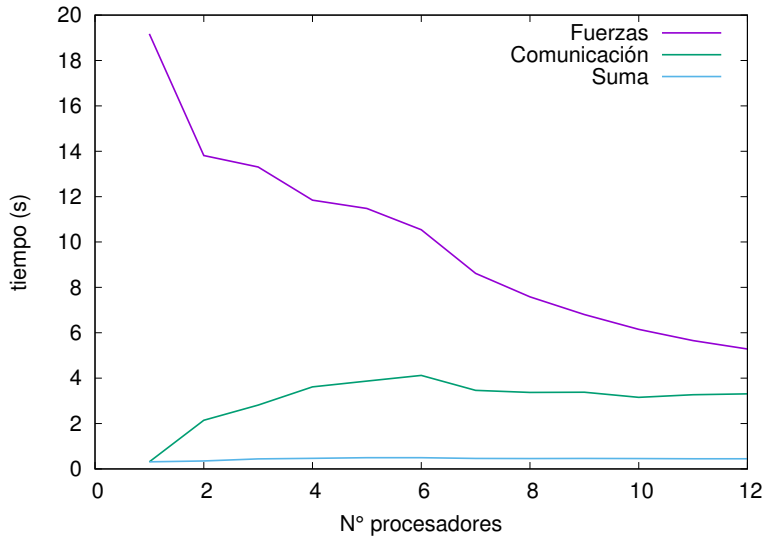

Figura 8: Tiempo de cálculo de las fuerzas, de la comunicación entre procesadores y de la suma de interacciones en función del número de procesadores, fijando el número de partículas a 10000, según el método 1.

Observando la Fig. 7 y como se esperaba, el tiempo de todos los procesos que hace el programa en paralelo aumenta con el número de partículas.

El aumento de partículas se traduce en un aumento de pares entre los cuales calcular la fuerza de interacción. Por eso el cálculo de las fuerzas se ralentiza. Del mismo modo, la comunicación entre procesadores es más lenta porque las matrices $\mathbf{G}_{ij}$ que tienen que enviar y recibir son de mayor tamaño.

El proceso que menos afectado se ve por el aumento de partículas es la suma de la fuerza total por partícula.

En la Fig. 8 se ve la variación del tiempo de computación de los distintos procesos que realiza el programa en función del número de procesadores usados para computar en paralelo.

El tiempo de computación de las fuerzas es el proceso que lleva más tiempo. Se observa también que repartir el cálculo entre más procesadores disminuye el tiempo de computación. Es más óptimo por tanto dividir el trabajo entre distintos procesadores, como cabía esperar.

El tiempo que lleva calcular la fuerza total por partícula se mantiene constante porque es un proceso que se hace en serie y, por tanto, únicamente depende del número de partículas, aquí fijo.

La comunicación es más rápida cuando hay menos procesadores entre los que pasar información. No obstante, a partir de 6 procesadores aproximadamente, el tiempo de computación se mantiene constante. Podemos concluir que la comunicación entre procesadores se ve mucho más afectada por la cantidad de datos

que se tengan que enviar y recibir que no por el número de procesadores que se usen.

2.2. Método 2: *Atom decomposition* algorithm with *Newton's second law*

Con el *Atom decomposition algorithm* se puede aplicar la tercera ley de Newton, por la cual $\mathbf{G}_{ij} = -\mathbf{G}_{ji}$. y por lo tanto no hace falta calcular todos los pares de partículas sino solamente la mitad.

En un programa en serie esto es fácilmente implementable ya que solo se calcula la diagonal superior de la matriz de interacciones $\mathbf{F}_{ij}$. En el caso del programa en paralelo, al separar la matriz de interacciones por partículas, es decir, cada procesador solo calcula la interacción entre unas pocas partículas con todas las demás como se muestra en la Fig. 9, la forma de calcular la mitad de las interacciones tiene que hacerse de una forma un tanto diferente. Para que todos los procesadores tengan un numero igual o parecido de interacciones que calcular solo se calcularan los coeficientes de matriz en los que $i + j$ es par cuando $j > i$ y los coeficientes donde $i + j$ es impar cuando $j < i$.

$$G_{ij} = \begin{pmatrix} \circ & \circ & \bullet & \circ & \bullet \\ \bullet & \circ & \circ & \bullet & \circ \\ \circ & \bullet & \circ & \circ & \bullet \\ \bullet & \circ & \bullet & \circ & \circ \\ \circ & \bullet & \circ & \bullet & \circ \end{pmatrix}$$

Figura 9: Matriz de interacciones entre todas las partículas. Solo se calculan los coeficientes en los que $i + j$ es par cuando $j > i$ y los coeficientes donde $i + j$ es impar cuando $j < i$.

Por consiguiente, cada procesador calculará un número predeterminado de filas de la matriz de interacción $\mathbf{F}_{ij}$ de la Fig. 9. Una vez calculado, cada procesador mandará su trozo de la matriz al master y éste calculará la fuerza total que siente cada partícula de la siguiente manera:

$$F_k = \sum_{j=1}^{N} G_{k,j} - \sum_{i=1}^{N} G_{i,k} \tag{9}$$

dónde F_k es la fuerza total que siente cada partícula.

Por último, ahora que el master tiene la fuerza que siente cada partícula tiene que mandar esta matriz a todos los demás procesadores. Resumiendo, la paralelización se ha hecho de la siguiente manera:

1. Separación de partículas por procesadores.

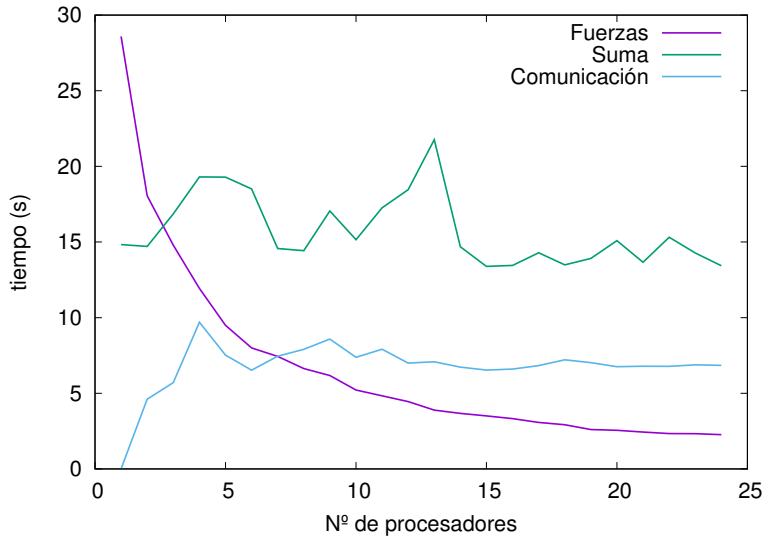

Figura 10: Cálculo de las fuerzas con el método 2. Número de partículas = 10000.

2. Cálculo de la mitad de términos de la matriz de interacciones $\mathbf{G}_{ij}$ como en la Fig. 9. Cada procesador calculara un número de filas.
3. Envío de cada trozo de la matriz $\mathbf{G}_{ij}$ al master.
4. El master recibe toda la matriz $\mathbf{G}_{ij}$.
5. El master calcula la fuerza total por partícula $\mathbf{F}_k$ (Eq. 2.2).
6. Envio de la matriz $\mathbf{F}_k$ desde el master a todos los demás procesadores
7. Todos los procesadores reciben la matriz $\mathbf{F}_k$ del master.

Como se puede observar en la Fig. 10, vemos que el proceso de calcular la fuerza total por partícula mediante la suma de la Eq. 2.2 (5º paso) es el proceso que más tiempo de computación lleva. Esto se debe a que este proceso solo lo hace el master y por lo tanto, se hace en serie. Por otro lado, vemos que el tiempo de comunicación se mantiene constante casi para cualquier número de procesadores y que el tiempo de cálculo de fuerzas decrece lentamente con muchos procesadores por lo que por mucho que añadamos muchos más procesadores que 24 no conseguiremos disminuir mucho el tiempo.

En la linea verde de la Fig. 14 podemos ver que el tiempo total utilizando este método decrece con el número de procesadores pero a partir de 15 se mantiene casi constante, es decir, 15 procesadores sería el número óptimo de procesadores para éste tipo de paralelización.

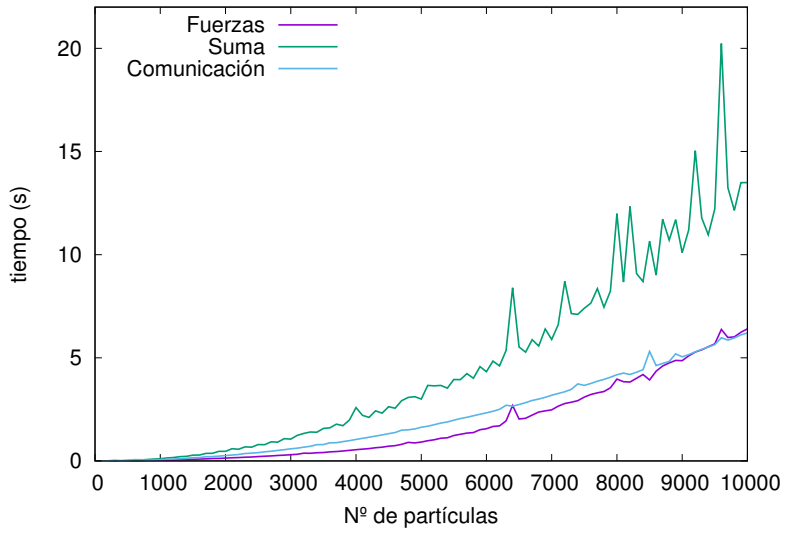

Figura 11: Cálculo de fuerzas con diferente número de partículas con el método 2. Número de procesadores = 8.

Si se analiza el tiempo de computación respecto al número de partículas (Fig. 11) se puede observar que el tiempo de cálculo de fuerzas será más grande que el de comunicación entre procesadores para un número de partículas muy grande. Aún así, el tiempo de cálculo de la suma de fuerzas sigue siendo mucho más grande que los otros dos y és lo que hace que este método sea muy lento. Por esa razón, a continuación se propone otro método para reducir el tiempo de cálculo del proceso de la suma de la fuerza.

2.3. Método 3: Full parallelization of *Atom decomposition* algorithm with *Newton's second law*

Para este método de paralelización se han seguido los mismos pasos que en el método 2 hasta el 4º paso. Después, en este caso, una vez el master tiene la matriz de interacciones $\mathbf{G}_{ij}$ entera, la envía a todos los demás procesadores. Ahora, cada procesador calcula la fuerza total de las partículas que se le han asignado por lo que cada procesador tendrá un trozo de la matriz $\mathbf{F}_k$. Para juntar esta matriz, cada procesador tiene que enviarla el master y una vez que el master tenga la matriz entera volverla a enviar a todos los procesadores. En resumidas cuentas, el método quedaría de la siguiente manera:

1. Separación de partículas por procesadores.
2. Cálculo de la mitad de términos de la matriz de interacciones $\mathbf{G}_{ij}$ como en la Fig. 9. Cada procesador calculara un número de filas.

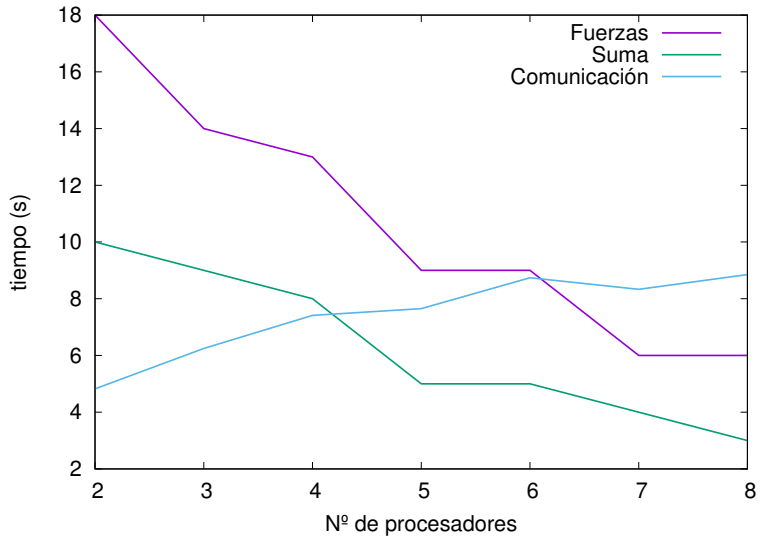

Figura 12: Cálculo de fuerzas con diferente número de procesadores con el método 3. Número de partículas = 10000.

3. Envio de cada trozo de la matriz $\mathbf{G}_{ij}$ al master.
4. El master recive toda la matriz $\mathbf{F}_{ij}$.
5. El master envía la matriz $\mathbf{G}_{ij}$ entera a todos los demás procesadores.
6. Todos los procesadores reciben la matriz $\mathbf{G}_{ij}$ del master.
7. Cada procesador calcula la fuerza total sobre las partículas que le hayan sido asignadas en el paso 1 calculando así un trozo de la matriz $\mathbf{F}_k$.
8. Envío de cada trozo de la matriz $\mathbf{F}_k$ al master.
9. El master recibe toda la matriz $\mathbf{F}_k$.
10. El master envía la matriz $\mathbf{F}_k$ entera a todos los demás procesadores.
11. Todos los procesadores reciben la matriz $\mathbf{F}_k$ del master.

Como se puede observar en la Fig. 12, vemos que comparado con los 15 segundos que tardaba en el método 2 (Fig. 10, el proceso de sumar las fuerzas ha disminuido en tiempo considerablemente. Además, como la suma está paralelizada vemos que disminuye con el número de procesadores utilizados. Hay que destacar que este método es muy complejo a la hora de implementar la comunicación entre procesadores y es por eso que no se ha conseguido un código

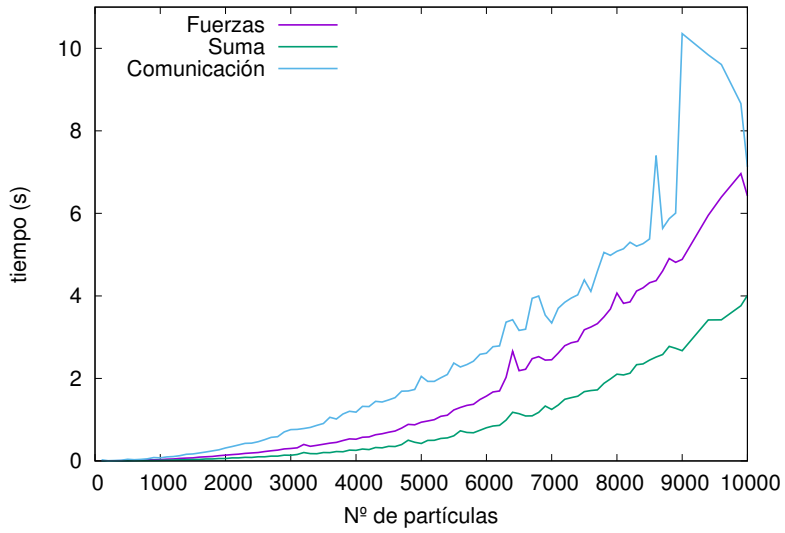

Figura 13: Cálculo de fuerzas con diferente número de partículas con el método 3. Número de procesadores = 8.

que funcione siempre. Por esa razón, solo se han podido utilizar un máximo de 8 procesadores.

También podemos ver que el tiempo de comunicación es ligeramente mayor en el método 3 (Fig. 12) que en método 2 (Fig. 10) pero como el tiempo de suma es muchísimo más pequeño este método es mucho más rápido, como se puede observar en la linea azul de la Fig. 14.

Analizando el tiempo de computación según el número de partículas se puede ver en la Fig. 13 que el tiempo de suma ha decrecido considerablemente comparándolo con el de la Fig. 11. Cabe destacar que los cálculos a partir del número de partículas igual a 9000 no fueron realizados correctamente y por lo tanto esos puntos no se deben tomar en cuenta.

Para terminar, en la Fig. 15 se puede observar que el método 3 es muchísimo mejor que el 2 y que para pocas partículas es tan eficiente como el código en serie aunque hay que resaltar que nunca mejora el tiempo del código en serio.

2.4. Comparación de los métodos y conclusiones

En todos los métodos en paralelo, el tiempo de computación disminuye al aumentar los procesadores. Los métodos 2 y 3 siempre son más lentos que aplicar el algoritmo en serie. Esto se debe a que al realizar la suma de las fuerzas el tiempo computacional es mucho más alto que en el método 1, aunque podría ser un problema de diferentes implementaciones de la suma. La única diferencia está en el método 1. Si bien para pocos procesadores el algoritmo en serie sigue

siendo más rápido, a partir de 7 procesadores esta tendencia se invierte, tal y como se ve en la Fig. 14.

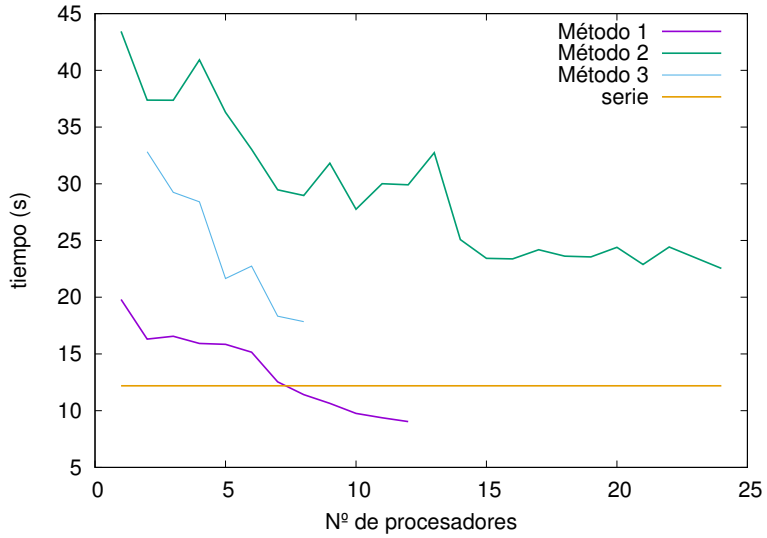

Figura 14: Cálculo de fuerzas con diferentes procesadores y diferentes métodos de paralelización. Número de partículas = 10000.

También se puede apreciar en la Fig. 14 que el hecho de calcular las fuerzas en paralelo (método 3) disminuye mucho el tiempo de computación total comparando con el método 2. Aún así, nunca se llega a tiempos tan bajos como los del método 1. En un futuro se podría intentar paralelizar las fuerzas en el método 1 para así conseguir un código todavía más eficiente.

En el caso de la Fig. 15, el tiempo de computación aumenta con el número de partículas. Como el número de procesadores es 8, el método 1 es más rápido que el algoritmo en serie, independientemente del número de partículas. Igual que en la Fig. 14, los métodos 2 y 3 son mucho más lentos.

En conclusión, a la hora de paralelizar el código para optimizar el tiempo podemos ver que el único método capaz de superar en velocidad al código en serie es el método 1 aunque puede que para sistemas con muchísimas más partículas el método 3 implementado de manera correcta consiga superar esta velocidad.

Por último, destacar que hemos tenido muchos problemas con las funciones de MPI a la hora de tratar con matrices de tamaño muy grande. Sobretodo, estos problemas se acentúan cuando hay muchos envíos en un mismo código (como es el caso del método 3). Nos hemos dado cuenta de que no teníamos las herramientas de MPI necesarias para solucionar este tipo de problemas.

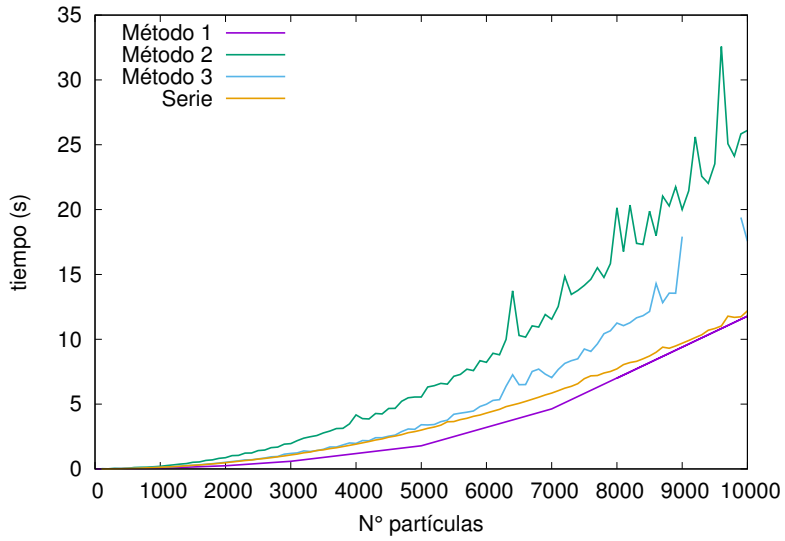

Figura 15: Cálculo de fuerzas con distinto número de partículas y diferentes métodos de paralelización. Número de procesadores = 8.

3. Integrador de Euler

Una de las principales partes de una código de dinámica molecular son los integradores de las ecuaciones de movimiento, aquellos que permiten la propia dinámica.

Para hacer un buen código de dinámica molecular, es menester elegir correctamente qué integrador se va a usar en la simulación. En esta sección se va a explicar el algoritmo de Euler explícito, uno de los más sencillos que se pueden implementar.

El algoritmo de Euler discretiza e integra las ecuaciones de movimiento Newton, (10), con tal de obtener la trayectoria de las partículas del sistema simulado.

$$\frac{\mathrm{d}\mathbf{r_k}}{\mathrm{d}t} = \mathbf{v_k} \qquad\qquad \frac{\mathrm{d}\mathbf{p_k}}{\mathrm{d}t} = \mathbf{F_k}. \tag{10}$$

Dado que se trata de una dinámica molecular, el integrador implementará la actualización de las posiciones y la de las velocidades a cada paso de tiempo, siguiendo las ecuaciones (11) y (12),

$$\mathbf{r_k}(t+\Delta t) = \mathbf{r_k}(t) + \mathbf{v_k}(t)\Delta t + \frac{1}{2m_k}\mathbf{F_k}(t)\Delta t^2, \tag{11}$$

$$\mathbf{v_k}(t+\Delta t) = \mathbf{v_k}(t) + \frac{1}{m_k}\mathbf{F_k}(t)\Delta t. \tag{12}$$

No obstante, este integrador presenta un problema severo. Dicho problema es la falta de simetría bajo una transformación temporal inversa para Δt suficientemente grandes. Esto implica que si se quiere implementar un buen algoritmo integrador, habría que tomar pasos de tiempo infinitesimalmente pequeños al usar el integrador de Euler.

La implementación numérica de este método en un código de DM es sencilla. En cada paso de tiempo, una vez las fuerzas que actúan sobre cada partícula hayan sido calculadas, se añade este algoritmo para calcular las nuevas posiciones y velocidades.

3.1. Código en paralelo

El método usado para paralelizar las dos subrutinas que he implementado, los integradores para las posiciones y para las velocidades, ha sido la herramienta MPI para Fortran90.

Con tal de paralelizar el cálculo del update de posiciones y velocidades, he elegido dividir los vectores según el índice correspondiente al número de partículas, i.e., que cada procesador ejecute el cálculo de un segmento de partículas, $[k, k + \delta k]$. El número de partículas que entre en cada procesador vendrá dado por el número de procesadores y el número de partículas, e.g, en el caso de 12 partículas y 4 procesadores, cada uno ejecutará los cálculos para 3 partículas.

Una vez cada procesador ha calculado su correspondiente segmento del vector, lo envían al Master (procesador 0), el cual hace un *merge* de todos los segmentos con tal de obtener el vector entero de posiciones y velocidades ya actualizado a ese instante de tiempo. Cuando todo el vector está ya actualizado, el Master vuelve a mandar el vector a todos los procesadores, consiguiendo así que todos ellos tengan toda la información actualizada.

Para enviar datos entre procesadores, he usado la subrutina del módulo 'mpif.h' MPI_ISEND. Una vez los datos han sido enviados, antes de ser recibidos, he implementado una barrier, MPI_BARRIER, con tal de esperar a que todos los procesadores hallan enviado su sección del vector y evitar fallos en la comunicación. Cuando la barrera ya ha sido superada, los destinatarios reciben los datos con un MPI_RECV, seguido también por una barrera que sólo sea superada una vez todos los destinatarios hayan recibido la información.

Para testear la paralelización, he estudiado la dependencia del tiempo de ejecución de mis subrutinas (posiciones y velocidades por Euler) en función del número de procesadores y del número de partículas tratadas. Como se puede ver en los resultados mostrados en la Figura 16, la paralelización no ha sido efectuada de modo eficiente, pues cuantos más procesadores, más lenta es la ejecución. El mismo comportamiento ha sido observado para el caso del integrador de las velocidades.

También se puede observar en la misma figura que el número de partículas tratadas tampoco influye significativamente en el tiempo de ejecución. Todo ello lleva a pensar que el método usado no ha sido el óptimo, y que la paralelización efectuada no ha servido para acelerar el código. Un posible motivo es la simplicidad de las subrutinas, que sólo consistían en un único bucle que recorría

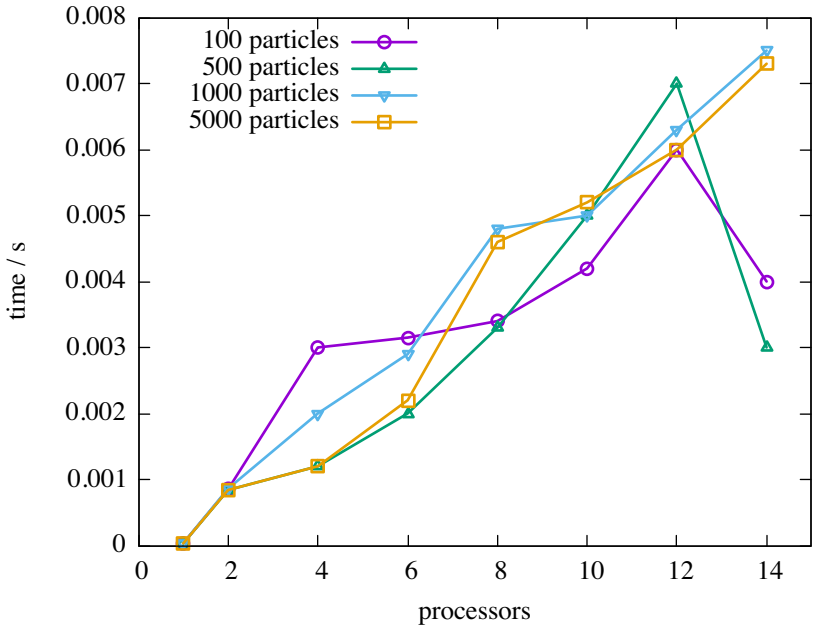

Figura 16: Dependencia del tiempo de ejecución del integrador Euler para las posiciones con el número de procesadores y partículas.

todo el vector de posiciones. Bucles de esta índole no son lentos, y escalan correctamente, mientras que el hecho de implementar la comunicación entre los procesadores relentiza el código. Cuantos más procesadores, más comunicaciones habrá, y más aumenta el tiempo de ejecución, tal y como se muestra en la Figura 16.

4. Integrador Velocity Verlet

El segundo integrador usado en el código es velocity Verlet y se aplica usando las siguientes expresiones dependiendo de la variable a integrar:

$$\mathbf{r}(t+\Delta t) = \mathbf{r}(t) + \mathbf{v}(t)\Delta t + \frac{\mathbf{F}(t)}{2m}\Delta t^2, \tag{13}$$

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) + (\mathbf{F}(t) + \mathbf{F}(t+\Delta t))\,\frac{\Delta t}{2m}. \tag{14}$$

Para calcular las posiciones se usa exactamente la expresión de Velocity Verlet pero para las velocidades está estructurado en dos partes. Primero se calcula la velocidad en el paso de tiempo actual, luego se calculan las fuerzas para actualizar la aceleración de las partículas y finalmente, con la nueva aceleración, se calcula el término a tiempo posterior. El programa tiene esta estructura para

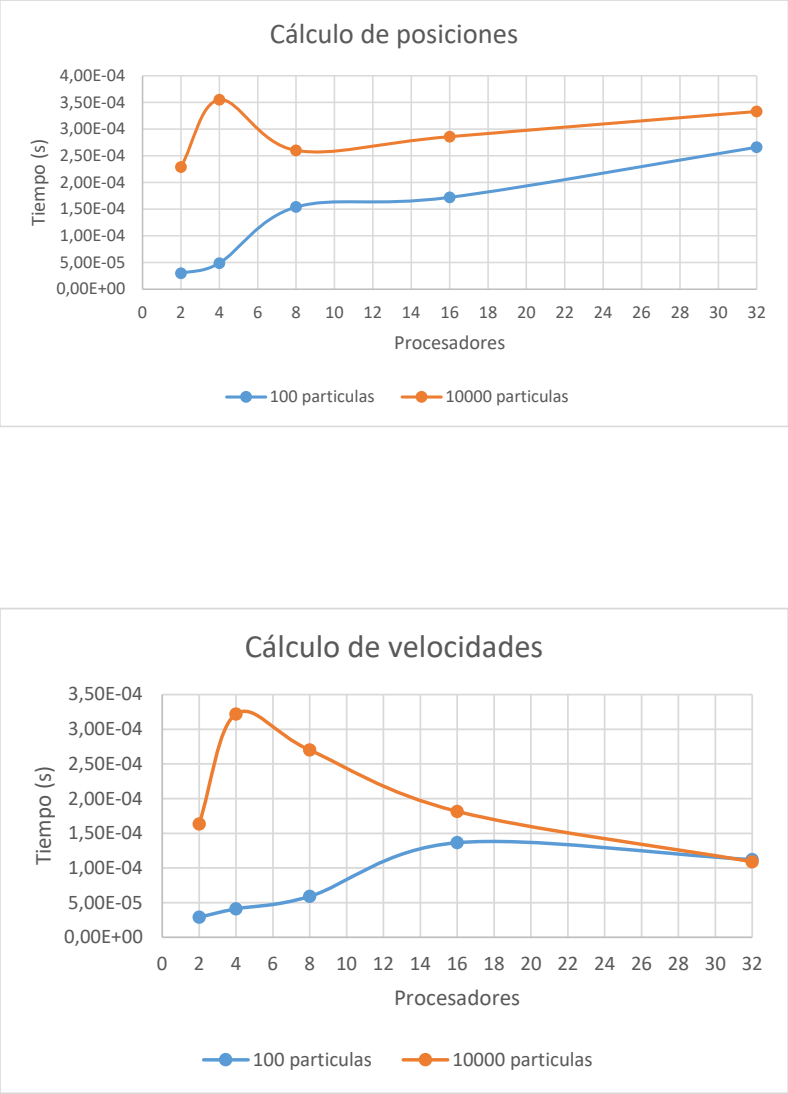

Figura 17: Dependencia del tiempo de ejecución del integrador Verlet para las posiciones y velocidades con el número de procesadores y partículas.

no tener que guardar la aceleración en el tiempo anterior para cada partícula, lo que reduce la memoria que necesita el programa.

Al realizar el cálculo en paralelo primero se realiza una parte del cálculo en cada uno de los procesadores a los que se ha enviado el cálculo. El reparto de los datos en cada procesador se realiza en el módulo paralelizar e importamos las variables a nuestro módulo. Luego enviamos la parte calculada de cada worker al master usando mpi_isend e introducimos una barrier para asegurar-nos de que todos los workers han enviado su parte antes de seguir con el programa. Entonces usamos un mpi_recv para que el master reciba todos los datos enviados.

Finalmente, el master envía el vector de datos entero actualizado a todos los workers usando también un mpi_isend, mpi_barrier y mpi_recv.

Una vez comprobado que el programa funciona correctamente realizamos un estudio para comprobar como escala el tiempo de ejecución de las subrutinas respecto al número de procesadores para dos cantidades de partículas diferentes (100 y 10.000).

Teóricamente esperaríamos el mismo comportamiento para las dos subrutinas, ya que solo difieren en el cálculo a realizar, pero no es exactamente lo que se observa. Para 100 partículas vemos en los dos casos que el tiempo de ejecución aumenta al aumentar el número de procesadores de una forma más o menos lineal, exceptuando uno de los puntos en cada caso. Para 10.000 partículas se observa un máximo de tiempo al usar 4 procesadores y luego en el caso de las velocidades el tiempo disminuye al aumentar los procesadores y para las posiciones primero disminuye y luego aumenta ligeramente.

Esta diferencia entre las dos subrutinas puede ser debida a que algunos cálculos se han realizado en nodos diferentes, por lo que no se puede garantizar que el tiempo de cálculo sea el mismo para cada ejecución del programa. Para estudiar rigurosamente la escalabilidad del programa se requeriría de más tiempo y más control sobre los procesadores donde se ejecuta el programa.

5. Condiciones periódicas de contorno

Las condiciones periódicas de contorno nos aseguran que si la partícula se encuentra fuera de la caja nuestra subrutina encauza a ésta de manera que se halle en la posición de convención de imagen mínima. Esto implica que, si la partícula sale por un lado de la celda, entrará por el lado opuesto al de salida (Figura 18). Estas son necesarias para simular sistemas grandes en un espacio más reducido de éste.

Para aplicar esto, el código sitúa el origen de coordenadas en el centro de nuestra celda de simulación. Entonces, debemos restar una longitud proporcional al tamaño de la caja (L) a aquellas partículas que se encuentren más allá de L/2 en cualquiera de las tres direcciones, en caso de 3D (Figura 19). De manera que, la constante de proporcionalidad venga dada por el número entero más cercano resultante de la división entre la posición de dicha partícula y L. En caso de que esta se encuentre en el interior de la celda esta constante valdrá 0 y la posición de la partícula no será variada. Para realizar este redondeo en

Fortran90 se emplea la función nint, quedando el código como:

$$\mathbf{r}(i, \text{dimensión}) = \mathbf{r}(i, \text{dimensión}) - L \cdot \text{nint}\left(\frac{\mathbf{r}(i, \text{dimensión})}{L}\right), \qquad (15)$$

dónde i recorrería de uno a N (número total de partículas), y dimensión las diferentes coordenadas para el eje x, y, z.

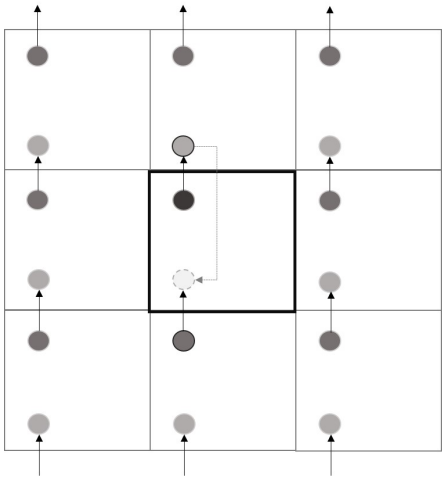

Figura 18: Representación de la convención de imagen mínima para una lattice 2D.

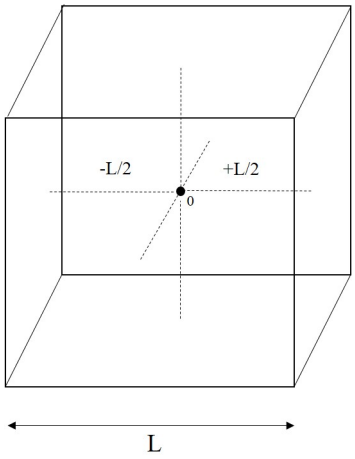

Figura 19: Celda de simulación 3D centrada en el origen de coordenadas.

5.1. Paralelización MPI

Para paralelizar la subrutina haremos la inicialización y finalización de MPI en el código principal incluyendo el módulo 'mpif.h', y repartiremos el número de partículas según el número de procesadores que empleemos, con los vectores ini y fin. Haciendo que, si no hay un número entero de partículas, el último procesador haga una partícula de más. Entonces en la subrutina, los diferentes workers con un MPI_ISEND enviarán sus posiciones ya modificadas de las partículas que les pertenecen al master (procesador 0). El master con un MPI_RECV recibirá toda la información actualizada y enviará toda esta con un MPI_ISEND a todos los workers. Estos también la recibirán con MPI_RECV, consiguiendo así que todos dispongan de las nuevas posiciones ya modificadas.

Entre estos procesos de comunicación entre procesadores encontraremos llamadas a las subrutinas del módulo mpi que crean las barreras, MPI_BARRIER. Estas obligan a esperar a tener toda la información de todos los workers o master antes del siguiente proceso de comunicación.

Para ver el efecto de la paralelización de dicha subrutina se ha graficado el tiempo que tarda esta con un número concreto de partículas (N=10000 y N=100) variando el número de procesadores empleados, obteniendo,

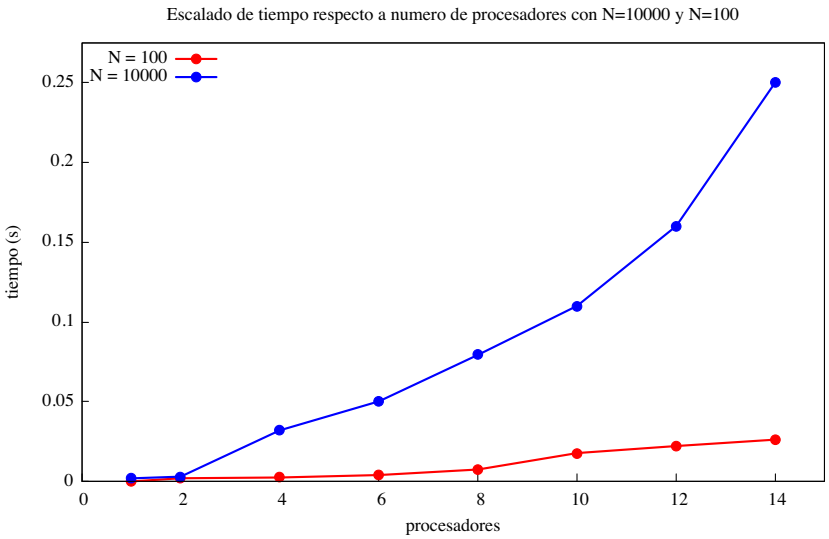

Figura 20: Dependencia del tiempo de ejecución de las condiciones periódicas de contorno con el número de procesadores a un número de partículas constante.

Se puede observar que en este caso particular tarda más cuanto más procesadores haya. Esto es porque lo que ralentiza el proceso no es la línea de código de condiciones periódicas, si no el proceso de comunicación entre procesadores. Observaríamos, la ley de Amdahl's pero a la inversa con N=10000. Este efecto se observa con mayor número de partículas. En este caso, no aumenta el tiempo

linealmente con el número de procesadores porque siempre tenemos una parte serial de código.

También se ha estudiado como varia el tiempo de cálculo en función del número de partículas manteniendo un número de procesadores constante (procesadores=4).

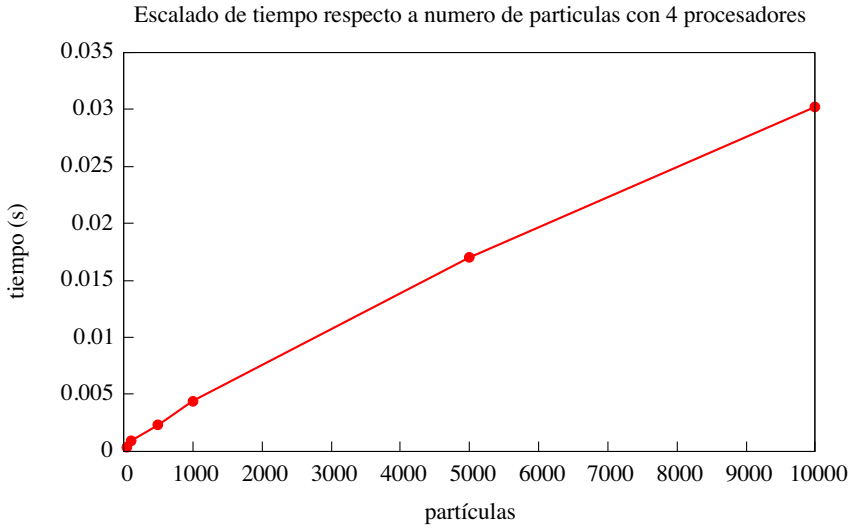

Figura 21: Dependencia del tiempo de ejecución de las condiciones periódicas de contorno con el número de partículas a un número de procesadores constante.

Como es lógico a mayor número de partículas se observa un mayor tiempo de cálculo. Y se observa una tendencia aceptablemente lineal.

La determinación de puntos en ambas gráficas son resultado del promedio de diferentes medidas del tiempo ya que el clúster utilizado presenta distintos nodos, no pudiéndose controlar a cuál de ellos envía. De este modo, el resultado no es reproducible de una ejecución a otra.

6. Cálculo de magnitudes en equilibrio: Presión y temperatura

6.1. Presión y temperatura:

Para el cálculo de la presión hemos hecho uso del teorema del virial que nos relaciona el promedio del trabajo de todas las partículas (la energía potencial) con la energía cinética promedia.

$$\langle E_c \rangle = -\frac{1}{2} \left\langle \mathbf{F}^{\mathrm{TOT}} \circ \mathbf{r} \right\rangle \tag{16}$$

Descomponemos la fuerza total como la suma de la fuerza interna debido a la interacción entre partículas y la externa al sistema:

$$\mathbf{F}^{\mathrm{TOT}} = \mathbf{F}^{\mathrm{int}} + \mathbf{F}^{\mathrm{ext}} \tag{17}$$

Es promedio del trabajo de las fuerzas externas será debido a la presión que ejerce la caja. Descomponiendo por componentes, dicha fuerza externa obtenemos la relación del promedio del trabajo externo con la presión:

$$\left\langle W^{ext} \right\rangle = L_x(-PL_yL_z) + L_y(-PL_xL_z) + L_z(-PL_xL_y) = -3PV \tag{18}$$

Donde L_x es el valor del lado x de la caja (ídem para L_y y L_z).
Utilizando el teorema de equipartición de la energía:

$$\langle E_c \rangle = \frac{3}{2} N k_B T \tag{19}$$

en la ecuación 16, obtenemos que la presión vale:

$$P = \frac{N k_B T}{V} + \frac{1}{3} \left\langle \mathbf{F}^{\mathrm{int}} \circ \mathbf{r} \right\rangle \tag{20}$$

Para la temperatura hemos usado el teorema de equipartición de la energía (eq. 19):

$$T = \frac{1}{3m} \left\langle \mathbf{v}^2 \right\rangle \tag{21}$$

6.2. Implementación:

Viendo las expresiones 20 y 21, podemos ver que por cada magnitud hay que hacer la suma de N productos escalares y luego sumarles i/o multiplicar unas constantes. Así que la implementación del cálculo de ambas magnitudes son muy parecidas.

En términos de paralelización, lo que he hecho es hacer que cada procesador me coja el mismo número de partículas a excepción del último que me cogerá el resto que queden. Luego lo que hará cada procesador será la suma parcial de dichos productos escalares y luego con la subrutina MPI_REDUCE las sumaré a un procesador común de manera jerárquica 22.

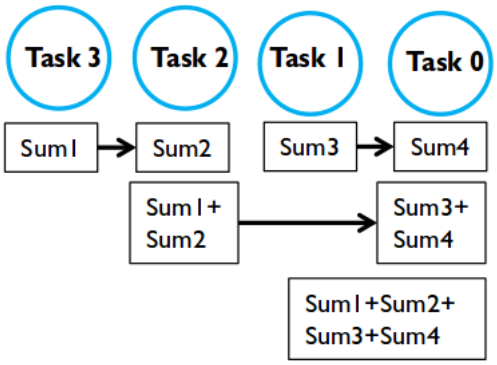

Figura 22: Esquema de cómo MPI_REDUCE funciona.

El escalado del tiempo de cómputo con el número de partículas ha sido diferente en función de la máquina. En mi caso he usado mi portáti 23 y el clúster cerqt2 24.

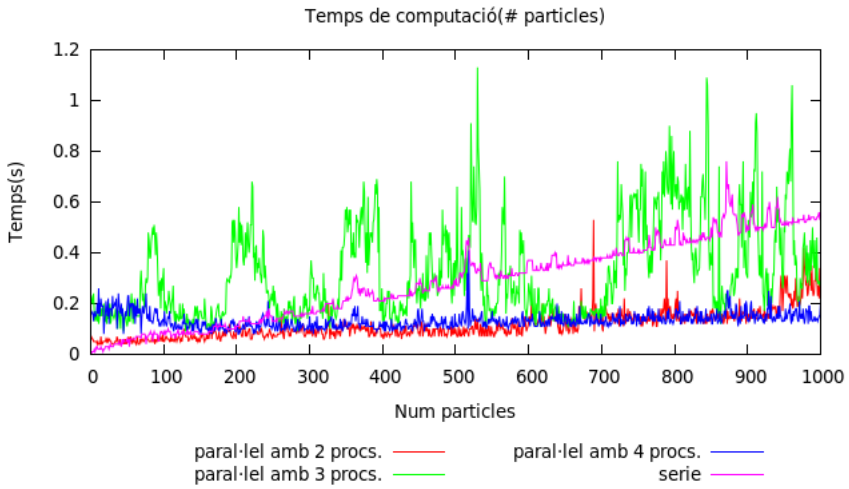

Figura 23: Tiempo de cálculo (s) para diferente número de procesadores de los cálculos de presión y temperatura hechos en un portátil.

Lo curioso aquí, es que es mejor trabajar en un ordenador portátil con un número par de procesadores que en un clúster. Además, en el clúster oscila mucho el tiempo en función del número de partículas.

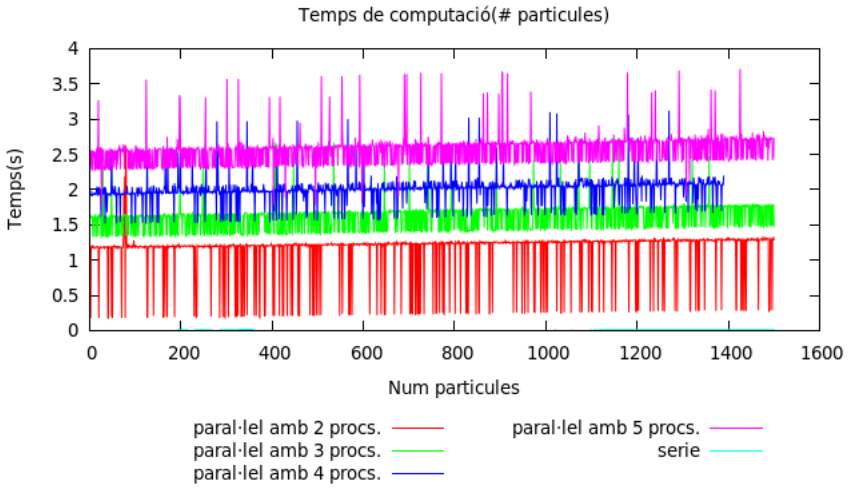

Figura 24: Tiempo de cálculo (s) para diferente número de procesadores de los cálculos de presión y temperatura hechos en cerqt2.

6.3. Resultados

Vemos en las figuras 25 y 26, que la temperatura y la presión llegan a un estado de equilibrio.

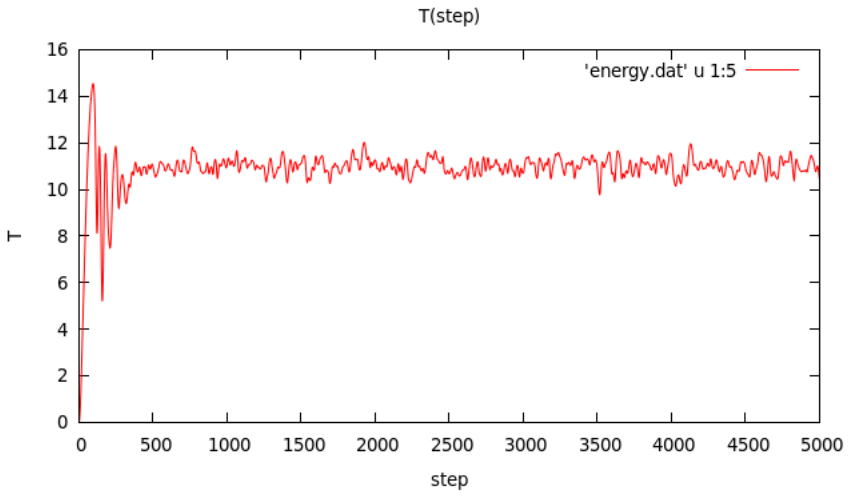

Figura 25: Temperatura en función del tiempo

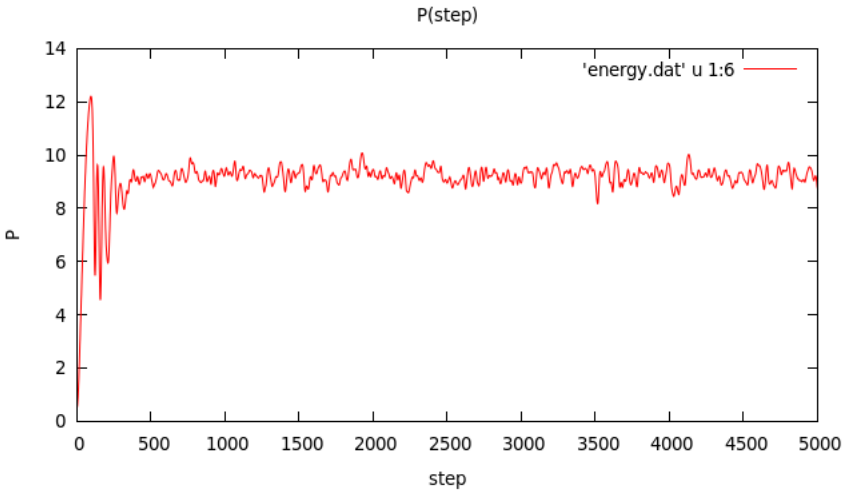

Figura 26: Presión en función del tiempo

7. Desplazamiento cuadrático medio

A continuación se detallará el procedimiento seguido en la implementación de la paralelización del código de análisis visual, última parte del código de dinámica molecular desarrollado.

Para esta parte se decide calcular el desplazamiento cuadrático medio (MSD) de las partículas a lo largo de la simulación, con la expresión:

$$MSD \equiv \langle (x(t) - x_o)^2 \rangle \tag{22}$$

Para calcular dicha expresión, se introduce en el archivo de entrada un valor de la frecuencia de escritura de las posiciones de las partículas (*restart*) y en el *main* del archivo que las contiene (con el mismo nombre y extensión *rst*). Del mismo modo, al iniciar la simulación se determina el número de frames que contendrá el archivo de restart.

Una vez finalizada la simulación, previamente a finalizar el entorno MPI, se llama a la subrutina *postvisual* con la entrada de número de frames, número de partículas y dimensión del sistema.

La subrutina (en serie y en paralelo) lee el archivo generado a lo largo de la dinámica molecular en una matriz. Posteriormente, mide para cada partícula el MSD a cada frame impreso. Es decir, para una misma partícula conociendo el número de frames y partículas del sistema recorre la matriz calculando la expresión (22). Dichos valores se van imprimiendo en una nueva matriz con F filas (correspondientes una a cada tiempo) y C columnas (una por cada partícula).

Finalmente, el resultado se imprime en un archivo con nombre *desplazamientos.dat*.

7.1. Implementación en paralelo

Se decide que la paralelización de la rutina se realice dividiendo las N partículas del sistema en los diferentes procesadores a usar. De este modo, se consigue que cada procesador pueda calcular el valor de MSD con la rutina ya generada para las partículas asignadas de manera autónoma, y sólo deba comunicarse con los demás al finalizar el cálculo.

Por otra parte, la lectura del archivo *restart.rst* se realiza también en paralelo. Éste hecho se debe a que en un primer momento se realizaba de manera única a través del MASTER y éste enviaba a cada procesador el rango correspondiente –según la división de partículas realizada– de la matriz de entrada para que realizara el cálculo. Se observaron problemas de comunicación por parte del MASTER a los procesadores y se decidió implementar también la lectura en paralelo.

Con la lectura en paralelo se han observado los siguientes resultados:

- Desaparición de los problemas de comunicación existentes, al no tener que comunicarse el MASTER con cada procesador.
- Tiempo de cálculo similar en ambos métodos, ya que la lectura del archivo se debía realizar de todos modos.

Se debe considerar que pese a que el tiempo de cálculo no aumenta, el gasto de memoria sí que incrementa debido a que la lectura del archivo *restart.rst* es realizada por todos los procesadores. En cualquier caso, se hace un balance positivo ya que en el método anterior intentado se habían producido problemas en algunos tests realizados a nivel de comunicación que no se habían podido solucionar.

Posteriormente, cada procesador calcula el MSD para las partículas asignadas en la división de trabajo realizada y mandan los rangos de la matriz de desplazamientos a MASTER. Éste, después de recibir los datos, los imprime en el archivo de resultados final.

Así, el proceso se puede resumir como:

1. Lectura del archivo de entrada completo en paralelo.
2. Cálculo del MSD en paralelo, con división por partículas en los diferentes procesadores.
3. Envío de valores de MSD de los diferentes procesadores a MASTER.
4. Impresión de los resultados en *desplazamientos.dat*.

7.2. Análisis de la implementación en paralelo

La implementación en paralelo se ha realizado de manera satisfactoria obteniendo unos resultados de disiminución de tiempo considerables en aumentar el número de procesadores.

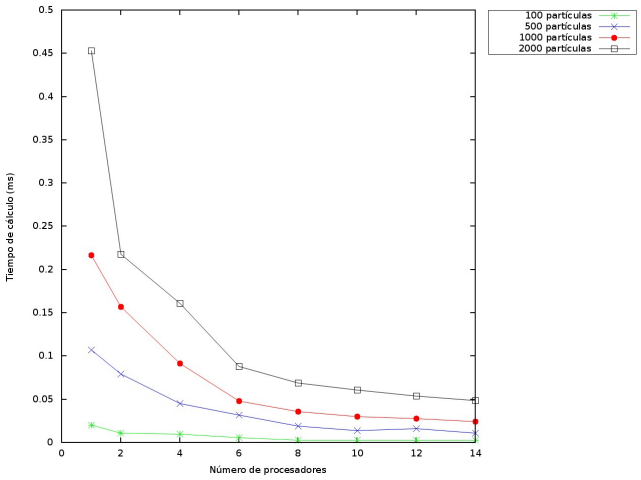

Figura 27: Tiempo de cálculo (ms) para diferentes sistemas (N=100, 500, 1000, 2000) en función de los procesadores utilizados.

Para diferente número de partículas (figura 27 se ha analizado el tiempo de cálculo de la rutina para un número creciente de procesadores (de 1 –en serie– a 14) observando como la disminución del tiempo de cálculo es considerable entre 2 y 6 procesadores, pero a partir de éstos el tiempo es muy parecido.

Se puede observar como para sistemas de mayor tamaño (N=1000,2000) la disminución de velocidad entre el cálculo en serie y con dos procesadores es remarcable. Prácticamente, con 2000 partículas en aumentar el número de procesadores en dos se consiguen tiempos de cálculo comparables al sistema de 1000 partículas con dos procesadores menos. A partir de 8 procesadores dicha tendencia se pierde obteniendo un comportamiento asintótico.

Analizando el tiempo de cálculo para el mismo número de procesadores en aumentar el de partículas obtenemos un incremento lineal del tiempo, tal y como se puede observar en la figura 28. En este caso, se puede observar una dependencia cuasi-lineal respecto el tiempo de cálculo y el tamaño del sistema.

7.3. Conclusiones

Del proceso de paralelización de la rutina de cálculo del desplazamiento cuadrático medio de las partículas, podemos concluir:

- El proceso de paralelización a través de la división por partículas permite disminuir el tiempo de cálculo de manera relevante en aumentar los procesadores utilizados, equiparando la velocidad a sistemas más pequeños para un número de procesadores menor o igual a 6.
- Dicha disminución del tiempo de cálculo no se observa para un número de procesadores mayor a 6, ya que se observa un comportamiento asintótico.

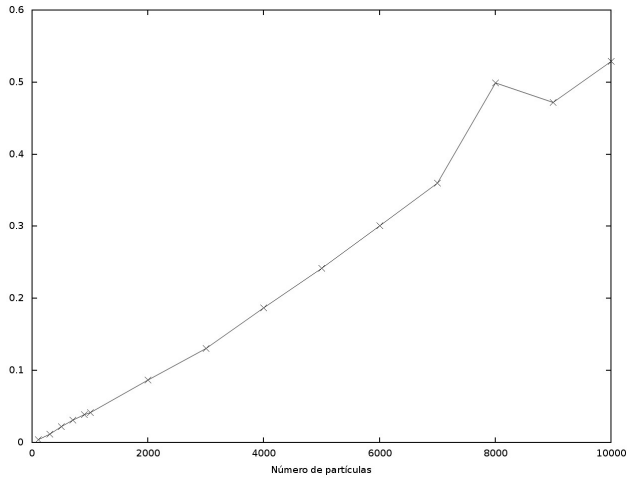

Figura 28: Tiempo de cálculo (ms) para seis procesadores en función del tamaño del sistema.

- Para un mismo número de procesadores, hay una relación cuasi-lineal entre el tiempo de cálculo y el tamaño del sistema.
- La paralelización, pues, es efectiva para pocos procesadores y permite calcular sistemas 4 veces mayores en el mismo tiempo (veáse figura 27, la velocidad para el sistema con 2000 partículas con 6 procesadores es similar al cálculo en serie con 500 partículas).