Applicazione del Codice LAMMPS per l'Analisi e la Simulazione della Produzione di Nanoparticelle di Silicio

Giuliano Carchini

18 giugno 2018

Indice

Ringraziamenti

Molte persone hanno contribuito alla realizzazione di questo documento e del tirocinio ad esso associato; particolari ringraziamenti vanno alle seguenti persone e istituzioni.

Ringrazio il mio relatore Dr. Cristiano Padrin per avermi seguito durante l'intero processo. Ringrazio il Dr. Massimo Celino per la gentile ospitalità presso la sede della Casaccia dell'ENEA e il Dr. Simone Giusepponi per le indicazioni essenziali a muovere i primi passi in questo studio; voglio inoltre ringraziare entrambi per gli innumerevoli utilissimi consigli. La parte operativa non sarebbe stata possibile senza l'utilizzo delle risorse computazionali dell'ENEA e del CINECA.

Infine ringrazio la mia famiglia ed in particolare i miei genitori per il loro continuo supporto morale.

Introduzione

Negli ultimi anni, l'impiego dei materiali sviluppati a partire da composti dell'ordine dei nanometri (*i.e.* nanotubi, nanoparticelle e nanocristalli) si è sempre più diffuso, trovando applicazione in diversi settori. Caratteristiche quali la forma, la dimensione ed i gruppi funzionali superficiali, esercitano enorme influenza sulle proprietà dei macro-materiali formati a partire dalle nano-componenti;[1, 2] grande attenzione è stata quindi rivolta alle tecniche di produzione che permettano una gestione raffinata dei parametri microscopici. Discipline come la Dinamica Molecolare permettono (tra le altre cose) di approfondire i meccanismi che governano le variazioni di tali fattori. Inoltre, la sempre maggior disponibilità di grandi sistemi di calcolo parallelo ha permesso lo sviluppo di numerosi software (librerie, strumenti, applicativi) in grado di simulare i fenomeni che alterano le suddette caratteristiche dei materiali, ed i loro effetti sul materiale stesso. In questa tesi viene descritta la formazione di nanoparticelle su di un sottile strato di silicio cristallino, una struttura comunemente chiamata wafer, e la simulazione del fenomeno ottenuta adottando il codice *LAMMPS*[1].[3] Questo codice offre diverse caratteristiche che lo rendono una scelta preferenziale rispetto a programmi analoghi, in quanto è open-source e pertanto può essere facilmente modificato (secondo le proprie esigenze puntuali) ed integrato con nuove caratteristiche,

[1]Large-scale Atomic/Molecular Massively Parallel Simulator

come campi di forza, tipi di atomi, condizioni al bordo o strumenti di analisi. Tra i codici simili a LAMMPS evidenziamo ad esempio CHARMM, AMBER, NAMD, NWCHEM, e Tinker. A differenza di LAMMPS però, questi sono stati sviluppati principalmente per simulare modelli di molecole biologiche, mentre questo ha un ambito di applicazione più generale. LAMMPS utilizza una strategia di parallelizzazione basata su un approccio di decomposizione spaziale, così come NAMD e NWCHEM, mentre CHARMM e AMBER basano la parallelizzazione su una decomposizione per atomi. Tinker invece è un codice seriale, e come tale è estremamente limitato nell'applicabilità a sistemi grandi e simulazioni di lunga durata. La scelta di simulare la formazione di nanoparticelle su un wafer di silicio è stata presa perché, a dispetto delle numerose applicazioni in campo elettronico, la produzione di nanomateriali da semiconduttori quali il silicio non ha prodotto storicamente risultati apprezzabili. I risulati di questo studio sono stati considerati soddisfacenti e sono oggetto di pubblicazione.[4]

Capitolo 1

Il problema fisico

L'interesse dal punto di vista scientifico e tecnologico per le nanoparticelle è cresciuto nel corso degli anni, per via delle numerose proprietà che le caratterizzano. In particolare, la presenza di questi composti determina un eccellente effetto antiriflesso, permettendo l'intrappolamento di un'elevata quantità di luce. Il conseguente aumento dell'assorbimento ottico (nello spettro del visibile) risulta estremamente utile in diversi campi: dall'optoelettronica, alla fotocatalisi, fino alla produzione di energia pulita.[5, 6, 7, 8]

Di conseguenza, è stata rivolta sempre più attenzione allo sviluppo ed al miglioramento delle tecniche di produzione. Gli studi effettuati in questo settore hanno permesso di individuare due tipologie di processo produttivo dei nanomateriali: sintesi per reazione chimica o processo fisico. La produzione di nanocristalli tramite processi di sintesi per reazione chimica risulta essere estremamente sensibile ai parametri sperimentali (*e.g.* pH, precursori, temperatura e tempo di crescita), con conseguente alterazione significativa delle proprietà finali del materiale.[9, 10, 11] Inoltre, tempi di reazione lunghi ($>$ ms) portano alla formazione di grandi aggregati amorfi in soluzione, allargando la distribuzione delle dimensioni e portando alla perdita dell'ef-

fetto di contenimento quantistico. Tale effetto consiste nella variazione delle proprietà elettroniche ed ottiche che si verifca quando un campione è sufficientemente piccolo (dimensione < 10 nm).[1] Questa variazione è causata dalla compressione di elettroni e buchi presenti nel materiale a dimensioni vicine alla *misura quantistica*, nota come *raggio dell'eccitone di Bohr*.

Un processo fisico permette invece di ottenere un maggiore controllo sulle dimensioni e forme, caratteristiche fondamentali per l'analisi delle funzionalità delle particelle. Appartengono a questa categoria la tecnica PLD[1][12, 13] e la Litografia Ionica ed Elettronica' (*i.e.* FIB[2][14] o LASiS[3][15]), spesso impiegate nella produzione di nanoparticelle metalliche. Per quanto riguarda semiconduttori e isolanti invece, impiegando tali tecniche, sono state sperimentate diverse difficoltà nell'ottenere nanoparticelle uniformi da un bersaglio bi- o multiatomico; conseguentemente lo studio e la produzione di nanoparticelle presentano ancora diverse complicazioni. Le cause di queste difficoltà sono state individuate nel comportamento dei laser a luce ultravioletta (UV) e dei raggi ionici in atmosfera controllata e sotto vuoto, quando provocano l'ablazione del materiale della superficie bersaglio, generando la *plume* di plasma ad alta energia. Sotto queste condizioni infatti l'irradiazione causa l'evaporazione di atomi, molecole ed aggregati ad alta energia dal bersaglio, che si combinano in modo non ordinato quando vanno in collisione con il substrato.

[1]Pulsed Laser Deposition
[2]Focused Ion Beam
[3]Laser Ablation Synthesis in Solution

Il processo fisico che permette di superare queste difficoltà consiste nell'irradiare la superficie del materiale semiconduttore o isolante con un fascio di protoni generato da un laser ad alta energia in meno di 1 ps applicato su di un materiale opportuno. L'impiego di questa tecnica ha trovato grande applicazione in diversi settori, ma solo recentemente è stata adottata, con successo, per la produzione di nanoparticelle uniformi da materiali non conduttori. Il caso che andremo ad analizzare è l'applicazione di questo processo su un wafer di silcio; nella fase sperimentale portata a termini da Barberio *et al.*[4] è stato osservato che, irradiando il campione con un impulso inferiore ad 1 ps, si formano superfici policristalline nanostrutturate su aree di decine di mm^2, contraddistinte da una variazione delle caratteristiche molto limitata.

Le immagini ottenute con tecniche AFM[4] e SEM[5] relative alla sperimentazione hanno confermato la formazione di sequenze ordinate di nanostrutture, composte da particelle semisferiche. Altre immagini dello stesso esperimento mostrano come queste particelle si siano formate in modo ordinato per tutta l'area irradiata, andando a formare una struttura nota come "nanotappeto" (vedere Figura 1.1 b e c).

[4]Atomic Force Microscopy
[5]Scanning Electron Microscopy

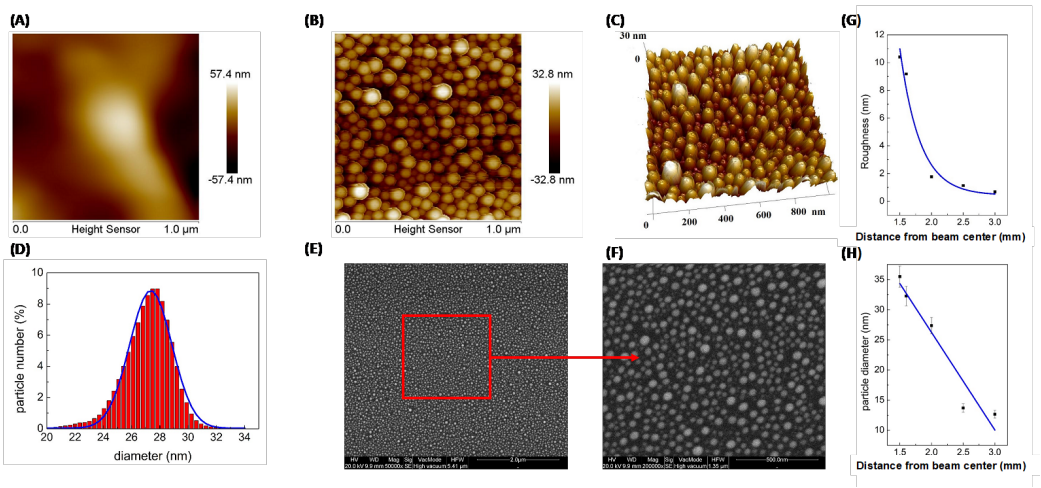

Figura 1.1: Immagini AFM e SEM del campione irradiato. Dall'articolo [4]: (A-C) 2D and 3D AFM images of the Silicon wafer before (A) and after (B-C) proton irradiation. The Silicon wafer was placed at a distance of 2.4 cm from the proton source; (D) Particle size distribution evaluated from AFM images taken at a radial distance of 2 mm from the center beam and fit with Gaussian analysis; (E-F) SEM images of the Silicon wafer surface taken at a radial distance of 1 mm from the beam center. Figure (E) shows an area of 6×6 ≠m2; Figure (F) an area of 1×1 ≠ m2; (G) Surface roughness as function of the radial distance from the beam center; (H) Particle diameter distribution as function of the radial distance from the beam center

Capitolo 2

Il modello matematico

Il modello matematico che caratterizza la metodologia della dinamica molecolare, indipendentemente dalle numerosissime implementazioni pratiche, è basato sul seguente sistema di equazioni differenziali di Newton,

$$\begin{aligned} m_i \frac{\mathrm{d}\vec{v}_i}{\mathrm{d}t} &= \sum_j F_2(\vec{r}_i, \vec{r}_j) + \sum_k F_3(\vec{r}_i, \vec{r}_j, \vec{r}_k) + \cdots \\ \frac{\mathrm{d}\vec{r}_i}{\mathrm{d}t} &= \vec{v}_i \end{aligned}$$

dove m_i è la massa dell'atomo i, $\vec{r}_i$ e $\vec{v}_i$ sono i suoi vettori posizione e velocità, F_2 è una funzione di forza che descrive l'interazione di coppia tra gli atomi, mentre F_3 descrive quella a tre corpi. E' inoltre possibile includere termini di forze che descrivano quelle interazioni presenti tra un numero di corpi maggiore. Nei limiti dell'accuratezza queste equazioni descrivono pienamente l'evoluzione temporale del sistema. Contrariamente ai metodi ab-initio quindi, la dinamica molecolare classica permette la descrizione del comportamento dinamico di un sistema atomico in modo empirico, senza dover risolvere l'equazione di Schroedinger

$$H\Psi = E\Psi$$

dove Ψ rappresenta la funzione d'onda totale del sistema ed E la sua energia. Questa formula dall'apparenza semplice, in realtà risulta estrememamente complessa da un punto di vista computazionale, a causa dell'operatore Hamiltoniano H; in particolare, la sua valutazione analitica non è possibile per sistemi con più di due particelle a causa di un termine di interazione tra di esse. Numerose tecniche numeriche e approssimazioni sono state sviluppate per risolvere questa limitazione ma risultano già proibitive per sistemi con poche decine di atomi. Un alternativa molto più economica ma meno accurata consiste nella tecnica DFT[1], la quale impiega un funzionale per valutare l'energia del sistema a partire dalla sua densità elettronica. Il problema si riduce quindi nel risolvere un sistema di equazioni del tipo

$$\left[-\frac{1}{2}\nabla^2 + V_{eff}(r)\right]\phi_i(r) = \epsilon_i\phi_i(r)$$

Nonostante i costi computazionali molto più contenuti, le dimensioni dei sistemi investigati e i corrispondenti tempi di simulazione risultano comunque di diversi ordini di grandezza inferiori a quelli ottenibili mediante la dinamica molecolare classica.

[1]Density Functional Theory

2.1 Algoritmo di integrazione Il Velocity Verlet

Esistono diversi algoritmi per implementare l'integrazione delle equazioni del moto di Newton. In questa sezione descriveremo il Velocity Verlet, uno degli integratori più comunemente impiegati, nonché la scelta di default applicata da LAMMPS. Tale integratore fornisce una buona stabilità numerica, oltre ad altre proprietà come la reversibilità temporale e la conservazione della forma simplettica nello spazio delle fasi, senza costo computazionale addizionale rispetto al semplice metodo di Eulero.

Per una equazione differenziale del secondo ordine del tipo $\ddot{\vec{x}}(t) = \vec{A}(\vec{x}(t))$ con condizioni iniziali $\vec{x}(t_0) = \vec{x}_0$ e $\dot{\vec{x}}(t_0) = \vec{v}_0$, una soluzione numerica approssimata $\vec{x}_n \approx \vec{x}(t_n)$ ai tempi $t_n = t_0 + n\Delta t$ con un passo di dimensione $\Delta t > 0$ può essere ottenuta con il metodo seguente:

- imponiamo $\vec{x}_1 = \vec{x}_0 + \vec{v}_0\Delta t + \frac{1}{2}A(\vec{x}_0)\Delta t^2$,
- per $n = 1, 2, \ldots$ iteriamo

$$\vec{x}_{n+1} = 2\vec{x}_n - \vec{x}_{n-1} + A(\vec{x}_n)\Delta t^2$$

Ricordiamo quindi che l'equazione del moto di Newton per un sistema fisico conservativo è

$$M\ddot{\vec{x}}(t) = F(\vec{x}(t)) = -\nabla V(\vec{x}(t)),$$

oppure singolarmente

$$m_k\ddot{\vec{x}}_k(t) = F_k(\vec{x}(t)) = -\nabla_{\vec{x}_k} V(\vec{x}(t)),$$

dove

t è il tempo,

$\vec{x}(t) = (\vec{x}_1(t), \ldots, \vec{x}_N(t))$ è l'insieme del vettore posizione di N oggetti,

V è la funzione del potenziale scalare,

F è il gradiente negativo del potenziale, che costituisce l'insieme delle forze agenti sulle particelle,

M è la matrice delle masse, tipicamente diagonale a blocchi com massa m_k per ogni particella.

Questa equazione può essere usata per descrivere l'evoluzione di diversi sistemi fisici, in funzione del potenziale V; infatti, i sistemi studiati possono variare da molecole interagenti alle orbite dei pianeti. Dopo una breve trasformazione per portare la massa a destra dell'uguaglianza e trascurando la struttura di particelle multiple, l'equazione può essere semplificata come

$$\ddot{\vec{x}}(t) = A(\vec{x}(t))$$

con una funzione vettoriale A adatta che rappresenta l'accellerazione dipendente dalla posizione. Tipicamente sono anche date una posizione

$\vec{x}(0) = \vec{x}_0$ ed una velocità iniziale $\vec{v}(0) = \dot{\vec{x}}(0) = \vec{v}_0$. Una volta definito il problema, è necessario discretizzarlo e risolverlo numericamente. Viene quindi definito un intervallo temporale $\Delta t > 0$ con cui viene campionata la sequenza $t_n = n\Delta t$; l'obiettivo è quello di costruire la sequenza di punti $\vec{x}_n$ che approssima i punti $\vec{x}(t_n)$ sulla traiettoria della soluzione esatta.

Mentre il metodo di Eulero usa una differenza in avanti come approssimazione della prima derivata nelle equazioni differenziali di prim'ordine, l'integrazione di Verlet può essere vista come l'applicazione della differenza centrata per approssimare la derivata seconda:

$$\frac{\Delta^2 \vec{x}_n}{\Delta t^2} = \frac{\frac{\vec{x}_{n+1}-\vec{x}_n}{\Delta t} - \frac{\vec{x}_n-\vec{x}_{n-1}}{\Delta t}}{\Delta t} = \frac{\vec{x}_{n+1} - 2\vec{x}_n + \vec{x}_{n-1}}{\Delta t^2} = \vec{a}_n = \vec{A}(\vec{x}_n).$$

L'integrazione di Verlet nella forma implementata nel metodo Störmer, utilizza questa equazione per ottenere il vettore posizione successivo a partire dai due precedenti, senza usare la velocità:

$$\vec{x}_{n+1} = 2\vec{x}_n - \vec{x}_{n-1} + \vec{a}_n \Delta t^2, \qquad \vec{a}_n = \vec{A}(\vec{x}_n)$$

La procedura di discretizzazione genera un errore che deve essere valutato. Fortunatamente, la simmetria temporale riduce il livello di errori locali introdotti dalla discretizzazione; infatti, i termini dispari vengono rimossi, nel caso specifico i termini in Δt di terzo grado. L'errore locale viene quantificato inserendo i valori esatti $\vec{x}(t_{n-1}), \vec{x}(t_n), \vec{x}(t_{n+1})$ nell'iterazione e valutando

le espansioni di Taylor al tempo $t = t_n$ del vettore posizione $\vec{x}(t \pm \Delta t)$ nelle diverse direzioni temporali:

$$\vec{x}(t + \Delta t) = \vec{x}(t) + \vec{v}(t)\Delta t + \frac{\vec{a}(t)\Delta t^2}{2} + \frac{\vec{b}(t)\Delta t^3}{6} + \mathcal{O}(\Delta t^4)$$
$$\vec{x}(t - \Delta t) = \vec{x}(t) - \vec{v}(t)\Delta t + \frac{\vec{a}(t)\Delta t^2}{2} - \frac{\vec{b}(t)\Delta t^3}{6} + \mathcal{O}(\Delta t^4),$$

dove $\vec{x}$ è la posizione, $\vec{v} = \dot{\vec{x}}$ la velocità, $\vec{a} = \ddot{\vec{x}}$ l'accellerazione e $\vec{b}$ lo strappo (derivata terza della posizione rispetto al tempo). Sommando queste due espressioni otteniamo

$$\vec{x}(t + \Delta t) = 2\vec{x}(t) - \vec{x}(t - \Delta t) + \vec{a}(t)\Delta t^2 + \mathcal{O}(\Delta t^4).$$

Possiamo vedere che i termini di primo e terzo ordine provenienti dall'espansione di Taylor si cancellano, rendendo così l'integratore di Verlet più accurato della sola espansione di Taylor. Bisogna comunque fare attenzione al fatto che l'accellerazione viene ottenuta dalla soluzione esatta, mentre nell'iterazione essa viene valutata nel punto centrale. Nella valutazione dell'errore globale, cioè la distanza tra la soluzione esatta e la sequenza approssimata, questi due termini non si cancellano, andando a influenzare l'errore globale.

E' importante notare che per la prima iterazione quando $n = 1$ e il tempo $t = t_1 = \Delta t$, per valutare $\vec{x}_2$, è necessare conoscere la posizione $\vec{x}_1$ al tempo $t = t_1$. A prima vista questo potrebbe creare problemi, poiché le condizioni

iniziali sono note solo al tempo iniziale $t_0 = 0$. Fortunatamente però, da queste ultime è nota l'accellerazione $\vec{a}_0 = \vec{A}(\vec{x}_0)$; Questo ci permette quindi di ottenere un'adatta approssimazione della posizione per il primo intervallo usando il polinomio di Taylor di secondo grado:

$$\vec{x}_1 = \vec{x}_0 + \vec{v}_0\Delta t + \frac{1}{2}\vec{a}_0\Delta t^2 \approx \vec{x}(\Delta t) + \mathcal{O}(\Delta t^3).$$

L'errore sul primo intervallo è quindi dell'ordine di $\mathcal{O}(\Delta t^3)$. Questo non è considerato un problema perché in una simulazione con un gran numero di intervalli, l'errore sul primo è solo una frazione trascurabile dell'errore totale, che al tempo t_n è del'ordine di $\mathcal{O}(e^{Lt_n}\Delta t^2)$, sia per la distanza dei vettori posizione $\vec{x}_n$ da $\vec{x}(t_n)$ che per la distanza delle differenze finite tra $\frac{\vec{x}_{n+1}-\vec{x}_n}{\Delta t}$ e $\frac{\vec{x}(t_{n+1})-\vec{x}(t_n)}{\Delta t}$. Inoltre, per ottenere questo errore globale del secondo ordine, l'errore iniziale deve essere almeno del terzo.

Per quanto riguarda le velocità, esse non sono date esplicitamente nell' equazione di base di Störmer, e spesso è necessario calcolarle per ottenere certe quantità fisiche come l'energia cinetica. Questo può creare particolari difficoltà nelle simulazioni di dinamica molecolare, poiché l'energia cinetica e le temperature istantanee al tempo t non possono essere calcolate per un sistema, fino a che le posizioni siano note al tempo $t+\Delta t$. Un primo metodo per superare questa limitazione, consiste nello stimare la velocità utilizzando i termini della posizione con il teorema del valor medio:

$$\vec{v}(t) = \frac{\vec{x}(t+\Delta t) - \vec{x}(t-\Delta t)}{2\Delta t} + \mathcal{O}(\Delta t^2).$$

E' da sottolineare che la velocità così ottenuta caratterizza l'intervallo precedente rispetto alla posizione, dato che si tratta della velocità al tempo t e non $t+\Delta t$ e questo significa che $\vec{v}_n = \frac{\vec{x}_{n+1}-\vec{x}_{n-1}}{2\Delta t}$ è un'approssimazione di secondo ordine di $\vec{v}(t_n)$. Per lo stesso motivo, se dimezziamo il time step, il termine $\vec{v}_{n+1/2} = \frac{\vec{x}_{n+1}-\vec{x}_n}{\Delta t}$ risulta essere un'approssimazione del secondo ordine di $\vec{v}(t_{n+1/2})$, con $t_{n+1/2} = t_n + \frac{1}{2}\Delta t$. E' anche possibile ridurre l'intervallo in modo da approssimare la velocità al tempo $t+\Delta t$, perdendo però in accuratezza:

$$\vec{v}(t+\Delta t) = \frac{\vec{x}(t+\Delta t) - \vec{x}(t)}{\Delta t} + \mathcal{O}(\Delta t).$$

Un secondo metodo consiste nell'usare un approccio simile al leap frog, ma a differenza di quest'ultimo, consente di calcolare velocità e posizioni nello stesso intervallo temporale. A tal fine, le velocità vengono incorporate in modo esplicito, risolvendo il problema del primo step nell'algoritmo base di Verlet:

$$\vec{x}(t+\Delta t) = \vec{x}(t) + \vec{v}(t)\Delta t + \frac{1}{2}\vec{a}(t)\Delta t^2,$$

$$\vec{v}(t+\Delta t) = \vec{v}(t) + \frac{\vec{a}(t)+\vec{a}(t+\Delta t)}{2}\Delta t.$$

Tale algoritmo viene chiamato Velocity Verlet. Si può dimostrare che l'errore è dello stesso ordine dell'algoritmo base di Verlet. Notare che questo algoritmo non richiede necessariamente più memoria, poiché non è necessario conservare le velocità ad ogni singolo intervallo durante la simulazione. Lo schema di implementazione standard è il seguente:

1. Calcolare $\vec{v}(t+\frac{1}{2}\Delta t) = \vec{v}(t) + \frac{1}{2}\vec{a}(t)\Delta t$.
2. Calcolare $\vec{x}(t+\Delta t) = \vec{x}(t) + \vec{v}(t+\frac{1}{2}\Delta t)\Delta t$.
3. Utilizzando le posizioni così ottenute, derivare $\vec{a}(t+\Delta t)$ a partire dal potenziale di interazione.
4. Valutare $\vec{v}(t+\Delta t) = \vec{v}(t+\frac{1}{2}\Delta t) + \frac{1}{2}\vec{a}(t+\Delta t)\Delta t$.

E' anche possibile trascurare le velocità a metà intervallo:

1. Calcolare $\vec{x}(t+\Delta t) = \vec{x}(t) + \vec{v}(t)\Delta t + \frac{1}{2}\vec{a}(t)\Delta t^2$.
2. Utilizzando le posizioni così ottenute, derivare $\vec{a}(t+\Delta t)$ dal potenziale di interazione.
3. Valutare $\vec{v}(t+\Delta t) = \vec{v}(t) + \frac{1}{2}(\vec{a}(t)+\vec{a}(t+\Delta t))\Delta t$.

La limitazione principale di questo algoritmo consiste nell'avere un'accellerazione unicamente dipendente dalle posizioni ma indipendente dalla velocità. Infine è interessante notare che i risultati a lungo termini dell'algoritmo

Velocity Verlet, e in modo analogo del leap frog, sono di un ordine più accurati del metodo semi-implicito di Eulero. Gli algoritmi sono pressoché identici a meno di uno spostamento di mezzo intervallo nelle velocità. Questo si può facilmente dimostrare ruotando il precedente ciclo a partire dal terzo intervallo e quindi notare che il termine di accellerazione nel primo step può essere eliminato unendo i passi 2 e 4 del primo schema di implementazione. La sola diffrenza è che la velocità di mezzo nel Velocity Verlet coincide con la velocità finale valutata nel metodo di Eulero semi-implicito. L'errore globale di tutti i metodi di Eulero è di primo ordine, laddove l'errore globale di questo metodo è, analogamente al metodo di punto medio, di ordine 2. Inoltre, se l'accellerazione è effettivamente il risultato delle forze in un sistema meccanico conservativo o Hamiltoniano, l'energia dell'approssimazione oscilla essenzialmente intorno al valore costante del sistema esatto risolto, con un errore globale risultante ancora una volta di primo ordine per Eulero semi-esplicito e ordine due per Verlet-leap frog. Lo stesso si può dire per tutte le quantità conservative del sistema, come il momento lineare o angolare, che sono sempre conservate in un integratore simplettico.

Come descritto in precedenza, l'errore locale sulla posizione dell'integratore di Verlet corrisponde a $\mathcal{O}(\Delta t^4)$ mentre quello sulla velocità è $\mathcal{O}(\Delta t^2)$. L'errore globale sulla posizione è invece $\mathcal{O}(\Delta t^2)$, così come quello sulla velocità globale. Questi possono essere derivati a partire dalla seguenti espressioni:

$$errore(x(t_0 + \Delta t)) = \mathcal{O}(\Delta t^4)$$

e anche

$$x(t_0 + 2\Delta t) = 2x(t_0 + \Delta t) - x(t_0) + \Delta t^2 x''(t_0 + \Delta t) + \mathcal{O}(\Delta t^4).$$

Da qui risulta che

$$errore(x(t_0 + 2\Delta t)) = 2 \cdot errore(x(t_0 + \Delta t)) + \mathcal{O}(\Delta t^4) = 3\mathcal{O}(\Delta t^4).$$

Allo stesso modo avremo

$$errore(x(t_0 + 3\Delta t)) = 6\mathcal{O}(\Delta t^4),$$
$$errore(x(t_0 + 4\Delta t)) = 10\mathcal{O}(\Delta t^4),$$
$$errore(x(t_0 + 5\Delta t)) = 15\mathcal{O}(\Delta t^4),$$

che può essere generalizzato come

$$errore(x(t_0 + n\Delta t)) = \frac{n(n+1)}{2}\mathcal{O}(\Delta t^4).$$

Se consideriamo ora l'errore globale nelle posizioni tra $x(t)$ e $x(t + T)$, dove $T = n\Delta t$, è chiaro che

$$errore(x(t_0 + T)) = (\frac{T^2}{2\Delta t^2} + \frac{T}{2\Delta t}\mathcal{O}(\Delta t^4),$$

di conseguenza, l'errore globale (cumulativo) su un intervallo costante di tempo è dato da

$$errore(x(t_0 + T)) = \mathcal{O}(\Delta t^2).$$

Poiché la velocità è determinata in modo non-cumulativo dalle posizioni con l'integratore di Verlet, l'errore globale sulle velocità sarà anch'esso $\mathcal{O}(\Delta t^2)$. Nelle simulazioni di dinamica molecolare, l'errore globale è generalmente molto più importante dell'errore locale; per questo motivo l'integratore di Verlet è considerato un integratore di secondo ordine.

Per finire consideriamo l'effetto sull'algoritmo di possibili vincoli in sistemi con molte particelle. Ad esempio, possiamo avere potenziali che fissano gli atomi ad una specifica distanza o con forze attrattive. Questi possono essere "modellati" come molle che collegano le particelle. Utilizzando molle infinitamente rigide, il modello può poi essere risolto con un algoritmo di Verlet. In una dimensione, la relazione tra le posizioni libere $\tilde{x}_i^{(t)}$ e le posizioni forzate $x_i^{(t)}$ dei punti i al tempo t, può essere valutata con l'algoritmo

$$\begin{aligned} d_1 &= x_2^{(t)} - x_1^{(t)}, \\ d_2 &= \|d_1\|, \\ d_3 &= \frac{d_2 - r}{d_2}, \\ x_1^{(t+\Delta t)} &= \tilde{x}_1^{(t+\Delta t)} + \frac{1}{2} d_1 d_3, \end{aligned}$$

$$x_2^{(t+\Delta t)} = \tilde{x}_2^{(t+\Delta t)} - \frac{1}{2} d_1 d_3,$$

L'integrazione di Verlet è utile perché collega direttamente la forza alla posizione, piuttosto che risolvere il problema usando le velocità. Si deve evidenziare che la presenza di diverse forze agenti su una singola particella possono generare problemi. Un modo per prevenire questo tipo di situazioni è di effettuare un ciclo per ogni punto di una simulazione, in modo che il rilassamento del vincolo al passo precedente venga impiegato per accellerare la diffusione dell'informazione.

2.2 L'algoritmo rRESPA

Oltre all'integratore Velocity Verlet, LAMMPS include la possibilità di impiegare l'integratore rRESPA (reversible reference system propagator algorithms), sviluppato da Tuckerman *et al.*[16] Questo algoritmo è particolarmente adatto proprio per descrivere sistemi che contengono diverse scale temporali (oscillatori rigidi in fluidi morbidi, masse con diversi ordini di grandezza) oppure in presenza contemporanea di forze a corto e lungo raggio. Il punto di partenza di tali metodi consiste nell'espansione di Trotter applicata al classico operatore di Liouville.

L'operatore di Liouville L per un sistema di f gradi di libertà viene definito in cordinate cartesiane come:

$$iL = \{\cdots, H\} = \sum_{j=1}^{f} \left[\dot{x}_j \frac{\partial}{\partial x_j} + F_j \frac{\partial}{p_j} \right],$$

dove $\Gamma = \{x_j, p_j\}$ sono le posizioni e i momenti coniugati del sistema, F_j è la forza agente sul j-esimo grado di libertà e $\{\cdots, \cdots\}$ rappresenta la parentesi di Poisson del sistema. L è un operatore Hermitiano lineare sullo spazio delle funzioni quadrato integrabili di Γ. Il propagatore classico è quindi

$$U(t) = e^{iLt}$$

mentre lo stato del sistema al tempo t è dato da

$$\Gamma(t) = U(t)\Gamma(0),$$

e $U(t)$ è un operatore unitario, *i.e.* $U(-t) = U^{-1}(t)$. L'operatore di Liouville L può essre diviso in due parti, di modo che

$$iL = iL_1 + iL_2.$$

Per questa decomposizione, il teorema di Trotter[17] ci permette di scrivere

$$\begin{aligned} e^{i(L_1+L_2)t} &= [e^{i(L_1+L_2)t/P}]^P \\ &= [e^{iL_1(\Delta t/2)}e^{iL_2\Delta t}e^{iL_1(\Delta t/2)}]^P \\ &\quad +\mathcal{O}(t^3/P^2), \end{aligned}$$

dove $\Delta t = t/P$. Da questo risultatio, possiamo definire il propagatore discreto come

$$\begin{aligned} G(\Delta t) &= U_1\left(\frac{\Delta t}{2}\right)U_2(\Delta t)U_1\left(\frac{\Delta t}{2}\right) \\ &= e^{iL_1(\Delta t/2)}e^{iL_2\Delta t}e^{iL_1(\Delta t/2)}. \end{aligned} \tag{2.1}$$

Poiché i tre fattori in $G(\Delta t)$ sono separatamente unitari, è facile dimostrare che $G(t)$ è anch'esso unitario, ovvero $G^{-1}(t) = G^{\dagger}(t) = G(-t)$. Questo significa che ogni integratore basato sulla fattorizzazione di Trotter sarà reversibile. La soluzione formale per qualunque decomposizione dell'operatore di Liouville può essere generata come segue: definiamo

$$\Gamma_1[\Delta t, \Gamma(0)] = U_1(\Delta t)\Gamma(0)$$

e

$$\Gamma_2[\Delta t, \Gamma(0)] = U_2(\Delta t)\Gamma(0)$$

come lo stato al tempo Δt quando il sistema è propagato di $U_1(\Delta t)$ o $U_2(\Delta t)$, a partire dallo stato $\Gamma(0)$. Applicando poi gli operatori nell'equazione 2.1 a $\Gamma(0)$, otteniamo

$$\begin{aligned}
\Gamma(\Delta t) &= U_1\left(\frac{\Delta t}{2}\right) U_2(\Delta t) U_1\left(\frac{\Delta t}{2}\right)\Gamma(0) \\
&= U_1\left(\frac{\Delta t}{2}\right) U_2(\Delta t)\Gamma_1\left[\frac{\Delta t}{2}, \Gamma(0)\right], \\
\Gamma(\Delta t) &= U_1\left(\frac{\Delta t}{2}\right)\Gamma_2\left\{\Delta t, \Gamma_1\left[\frac{\Delta t}{2}, \Gamma(0)\right]\right\}, \\
\Gamma(\Delta t) &= \Gamma_1\left(\frac{\Delta t}{2}, \Gamma_2\left\{\Delta t, \Gamma_1\left[\frac{\Delta t}{2}, \Gamma(0)\right]\right\}\right). \qquad (2.2)
\end{aligned}$$

La procedura da seguire per generare questa soluzione su un calcolatore è quindi:

1. Assunta la condizione iniziale $\Gamma(0)$, si genera il moto mediante il propagatore $e^{iL_1\Delta t/2}$. Questo genererà a sua volta lo stato $\Gamma_1[\Delta t/2; \Gamma(0)]$.

2. Assumiamo lo stato appena generato come condizione iniziale ed otteniamo il moto usando il propagatore $e^{iL_2\Delta t}$. Otteniamo in questo modo lo stato $\Gamma_2\{\Delta t; \Gamma_1[\Delta t/2; \Gamma(0)]\}$.

3. Assumendo lo stato ottenuto al passo precedente come stato iniziale, generiamo infine il moto applicando $e^{iL_1\Delta t/2}$. La soluzione corrisponde allo stato $\Gamma(\Delta t)$ specificato nell'equazione 2.2

Per determinare la soluzione per una separazione più complicata, si procede in maniera analoga. Si deve notare che l'espansione di Trotter eseguita per ordini più grandi produrrà integratori anch'essi di ordine maggiore. Per esempio, per ottenere un integratore con una precisione di ordine $\mathcal{O}(\Delta t^4)$, bisogna partire con una espansione di Trotter del tipo

$$\begin{aligned} e^{i(L_1+L_2)\Delta t} \approx\ & e^{iL_1\Delta t/2}e^{iL_2\Delta t/2} - e^{-iC(\Delta t^3/24)} \\ & \times e^{iL_2(\Delta t/2)}e^{iL_1(\Delta t/2)}, \end{aligned}$$

dove

$$-iC = [(iL_1 + 2iL_2), [iL_1, iL_2]].$$

Con $[\cdots, \cdots]$ viene rappresentata la parentesi di Lie o commutatore. Lo svantaggio di usare un ordine così grande per l'integratore consiste nel fatto che il commutatore conterrà derivate delle forze rispetto alle posizioni che potrebbero risultare molto difficili da valutare. Vediamo ora esempi di semplici integratori reversibili. Consideriamo il propagatore generato dalla suddivisione,

$$iL_2 = \dot{x}\frac{\partial}{\partial x}; \qquad iL_1 = F(x)\frac{\partial}{\partial p}$$

Questo ci permette di ottenere il propagatore

$$G(\Delta t) = e^{(\Delta t/2)F(x)\partial/\partial p} e^{\Delta t\dot{x}\partial/\partial x} e^{(\Delta t/2)F(x)\partial/\partial p}.$$

Apllicando questa fattorizzazione a $\{x(0), p(0)\}$ e usando la proprietà esplicita di ogni operatore nella forma $e^{c\partial/\partial q}$ tale che

$$e^{c\partial/\partial q} f(q) = f(q + c),$$

dove c è indipendente da q, si può infine derivare l'integratore (espresso in termini di posizioni e velocità),

$$\begin{aligned} x(\Delta t) &= x(0) + \Delta t\dot{x}(0) + \frac{(\Delta t^2)}{2}\frac{F[x(0)]}{m}, \\ x(\Delta t) &= \dot{x}(0) + \frac{\Delta t}{2m}\{F[x(0)] + F[x(\Delta t)]\}. \end{aligned}$$

Questo risultato è identico a quello dell'algoritmo Velocity Verlet. Poiché $G(-t) = G^{-1}(t)$, segue che l'integratore è esattamente reversibile nel tempo. Questo garantisce stabilità per l'integratore e inoltre ne consente l'utilizzo nell'algoritmo ibrido Monte Carlo. Un altro modo di scrivere l'integratore è

$$\dot{x}\left(\frac{\Delta t}{2}\right) = \dot{x}(0) + \frac{\Delta t}{2m}F[x(0)],$$
$$x(\Delta t) = x(0) + \Delta t\dot{x}\left(\frac{\Delta t}{2}\right),$$
$$\dot{x}(\Delta t) = \dot{x}\left(\frac{\Delta t}{2}\right) + \frac{\Delta t}{2m}F[x(\Delta t)].$$

Partendo dalla condizione iniziale $\{x(0), p(0)\}$ possiamo valutare la velocità a metà intervallo, seguita dalla posizione ad intervallo pieno ed infine la velocità all'intervallo corrispondente. Questa procedura assomiglia a quella del leap frog, ma il processo viene iniziato in modo differente, le velocità non vengono calcolate a intervallo pieno e solo in alcuni casi nell'ultimo intervallo.

Capitolo 3

Il codice LAMMPS

In questa sezione analizziamo il codice LAMMPS, soffermandoci in particolare sull'esecuzione di un singolo passo di dinamica, effettuata mediante l'integrazione Velocity Verlet. Il programma è scritto quasi interamente in C++ ed è caratterizzato da una molteplicità di classi a cui corrispondono i diversi comandi utilizzati nello script di input. Nonostante la grande quantità di files sorgenti, LAMMPS è però caratterizzato da una gerarchia di classi di semplice comprensione, come riportato in figura 3.1.

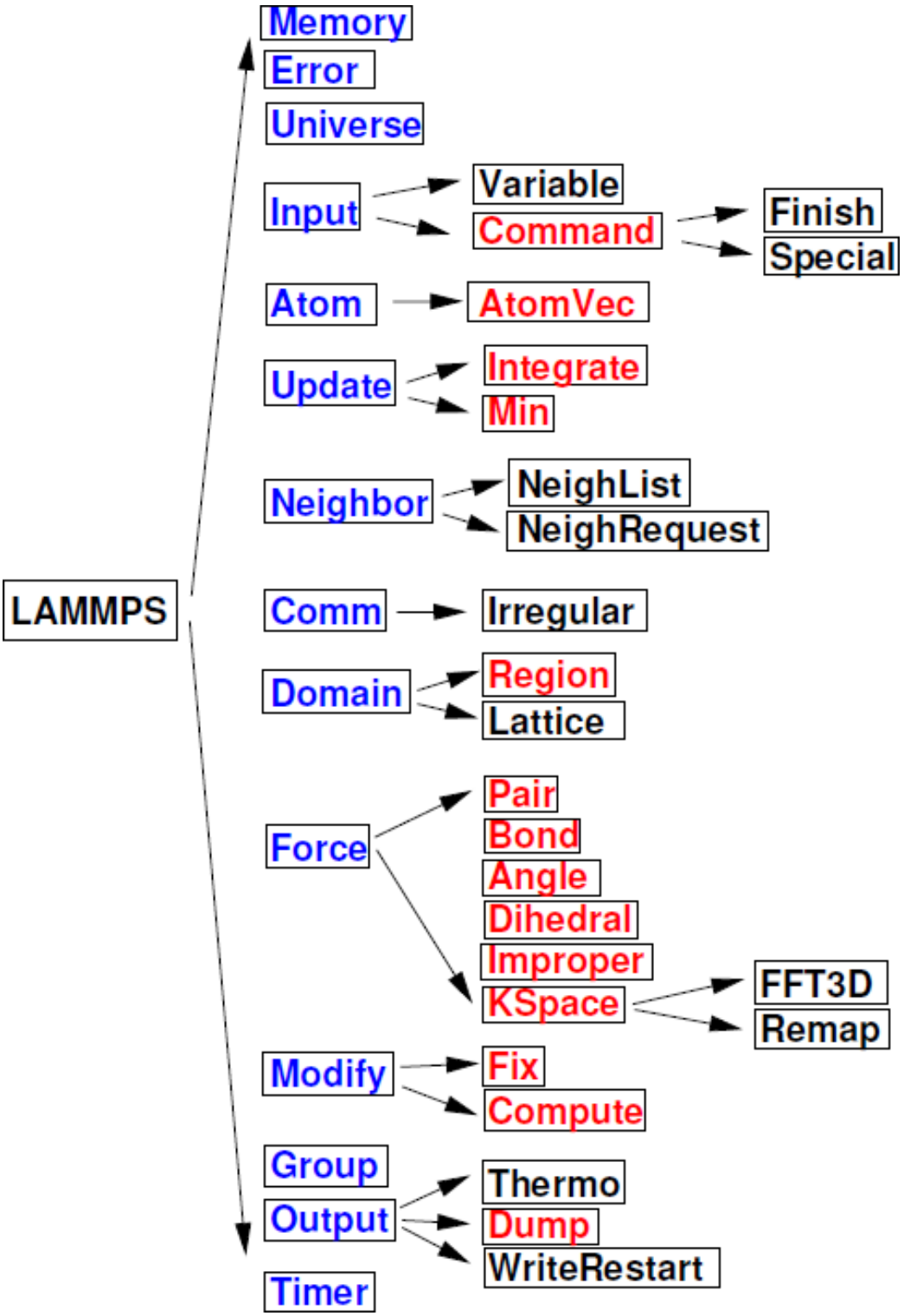

Figura 3.1: Gerarchia delle classi di LAMMPS.

Come si vede da tale rappresentazione, esiste una classe di LAMMPS omonima che viene invocata al principio di qualsiasi esecuzione di questo codice. Questa chiamata avviene ad opera dell'eseguibile principale (main.cpp), il quale può generare una o più istanze della classe **LAMMPS**, a cui verrà poi passato lo script di input. All'interno di questa è poi presente una dozzina di altre classi di alto livello, visibili ovunque nel codice ed evidenziate in blu nella figura 3.1; tale visibilità è dovuta ad alcune caratteristiche della classe **Pointers**, la quale viene ereditata da tutte le altre classi. Scendendo lungo la gerarchia, troviamo poi una serie di classi genitrici virituali, evidenziate in rosso, che definiscono i possibili stili associati ai singoli comandi; ognuna di queste classi sarà infatti genitrice di un certo numero di classi figlie, le quali implementeranno l'interfaccia prescelta nella genitrice. A titolo di esempio è interessante notare che lo stile "fix" possiede intorno a cento classi figlie; queste rappresentano i possibili constraint che possono essere specificati dal comando "fix" nello script di input, *e.g.* fix nve, fix shake, fix ave/time, le cui classi corrispondenti saranno quindi **FixNVE**, **FixShake** e **FixAveTime**. La sola eccezione a questa struttura, è rappresentata dai files della classe figlia per lo stile "command". Queste implementano comandi specifici nello script di input, che possono essere invocati prima, dopo e durante le singole esecuzioni. Come esempio ricordiamo i comandi create_box, minimize, run e velocity che codificano le classi **CreateBox**,**Minimize**,**Run** e **Velocity**.

Vediamo ora maggiori dettagli per le più importanti classi riportate in figura 3.1:

- **Memory**: gestisce l'allocazione di tutti i vettory e griglie di grandi dimensioni.

- **Error**: stampa tutti i messaggi d'errore e avvertimento.

- **Universe**: inizializza le partizioni in modo tale che possano essere effettuate più simulazioni allo stesso tempo, ognuna su un sottogruppo di processori allocati per uno stesso run; ciò avviene principalmente attraverso il compando mpirun.

- **Input**: legge lo script di input, immagazzina le variabili e invoca comandi isolati che sono classi figlie della classe **Command**.

- **Command**: classe genitrice per un certo script di input che esegue una operazione singola prima, dopo o durante le simulazioni. Le istanze di questa classe vengono generate, invocate e, una volta terminato il loro compito, distrutte a partire dalla classe **Input**

- **Finish**: viene instanziata per stampare le statitistiche su schermo al termine di una simulazione, quando vengono eseguiti comandi come run e minimize.

- **Special**: percorre la topologia di legame di un sistema molecolare per trovare i primi, secondi e terzi vicini di ogni atomo; viene invocata da diversi comandi, come read_data, read_restart e replicate.

- **Atom**: immagazzina tutti i dati relativi agli atomi. Più precisamente, essi sono allocati e conserfvati dalla classe **AtomVec** e la classe **Atom** conserva semplicemente un puntatore a quest'ultima; la classe **AtomVec** è una classe genitrice per gli stili che caratterizzano gli atomi, definita dal comando atom_style.

- **Update**: contiene un integratore e un minimizzatore; la classe **Integrate** è una genitrice degli integratori temporali Verlet e rRespa, a seconda della definizione contenuta nel comando run_style, mentre la classe **Min** è genitrice per i vari minimizzatori energetici.

- **Neighbor**: costruisce e immagazzina le liste dei vicini. La classe **NeighList** contiene una singola lista per tutti gli atomi mentre **NeighRequest** viene richiamata dagli stili pair, fix e compute, quando questi necessitano di una particolare lista di vicini.

- **Comm**: effettua comunicazioni tra i processori, tipicamente limitata alle informazioni degli atomi fantasmi. Questo implica scambi di messaggi MPI con i 6 processori adiacenti nella griglia logica costruita sul box di simulazione. A volte viene utilizzata la classe **Irregular**, quando atomi migrano verso altri processori.

- **Domain**: conserva la geometria del box di simulazione così come quella delle regioni geometriche e di ogni reticolo definito dall'utente. Que-

st'ultimo può essere costruito mediante i comandi region e lattice nello script di input.

- **Force**: valuta tutte le forze presenti tra gli atomi. La classe genitrice **Pair** si occupa di forze di non-legame o di coppia, incluse interazioni a molti corpi, mentre **Bond**,**Angle**, **Dihedral** e **Improper** rappresentano invece gli stili per quelle interazioni di legame caratteristiche di una topologia molecolare statica. Infine la classe genitrice **KSpace** consente di calcolare interazioni Coloumbiane a lungo raggio; una delle sue classi figlie, **PPPM** impiega le classi **FFT3D** e **Remap** per comunicare le informazioni ai processori vicini.

- **Modify**: immagazzina le liste delle classi **Fix** e **Compute**.

- **Group**: manipola e valuta diversi attributi di gruppi di atomi precedentemente assegnati tramite il comando group.

- **Output**: genera 3 tipi di output da una simulazione di LAMMPS. Avremo quindi informazioni termodinamiche stampate a schermo e su un log file, instantanee contenute in file di dump e files di restart; Le tre classi corrispondenti saranno rispettivamente **Thermo**, **Dump** e **WriteRestart**.

- **Timer**: conserva le informazioni relative ai tempi MPI, inviandole a schermo alla fine del run.

Al fine di comprendere al meglio il funzionamento di LAMMPS, analizziamo ora nel dettaglio lo pseudocodice che descrive lo svolgimento di un singolo timestep in una simulazione dinamica. Il primo step consiste nell'attivazione della classe **Integrate**, derivata della classe **Update**. **Integrate** è una classe genitrice, corrispondente al comando di script run_style; qui descriveremo il timestep caratterizzato dalla classe figlia **Verlet**, che implementa l'algoritmo di integrazione Velocity Verlet, descritto nella sezione precedente. Gran parte del lavoro viene svolto dal metodo **Verlet::run()**, ma è utile evidenziare altri metodi della classe. Nel dettaglio avremo i seguenti metodi:

- **init()**: viene chiamato all'inizio di ogni esecuzione, allo scopo di fissare alcuni valori interni, basati sui parametri dell'utente in altre parti del codice.

- **setup()**: effettua il calcolo preventivo delle forze necessarie per l'integratore il metodo Velocity Verlet; inoltre il metodo si occupa inoltre di impostare alcuni metodi fix e inizializzare alcuni contatori.

- **setup_minimal()**: variante del metodo precedente, che permette una inizializzazione più rapida; viene impiegato quando una seconda simulazione sia avviata al termine della precedente, a patto che le informazioni di base siano ancora valide.

- **force_clear()**: inizializza le forze a zero prima di ogni passo, in modo da permettere il calcolo cumulativo delle stesse.

Vediamo ora la struttura in pseudo codice, del metodo più importante alla base di uno step di dinamica, il **Verlet::run()**:

```
loop over N timesteps:
```

Questra istruzione ci indica che ciò che segue rappresenta il ciclo di istruzioni necessario ad eseguire un singolo step.

```
    ev_set()
```

Il metodo **ev_set()** (classe genitrice **Integrate**) assegna due parametri (**eflag** e **vflag**) per il cacolo dell'energia e del viriale, i quali determineranno se queste quantità dovranno essere calcolati nello step in esame. Questi parametri vengono poi passati ai diversi metodi che valutano le interazioni tra le particelle; ciò permette di limitare i successivi calcoli, in caso che queste quantità non siano necessarie per altre valutazioni.

```
    fix->initial_integrate()
    fix->post_integrate()
```

Alcuni fix vengono richiamati tramite la classe **Modify**, che conserva tutti gli oggetti di tipo **fix** e il loro ordine all'interno del timestep; questi permettono a LAMMPS di adattare tutte le operazioni necessarie in funzione dei parametri definiti dall'utente. Tutti i fix definiti con il metodo **initial_integrate()** vengono quindi richiamati all'inizio di ogni passo. Il metodo seguente (**post_integrate()**) viene invece utilizzato solo da alcuni fix, come **FixWallReflect**, impiegato nel descrivere il comportamento di particelle negli urti con il box di simulazione.

```
nflag = neighbor->decide()
if nflag:
        fix->pre_exchange()
        domain->pbc()
        domain->reset_box()
        comm->setup()
        neighbor->setup_bins()
        comm->exchange()
        comm->borders()
        fix->pre_neighbor()
        neighbor->build()
else
        comm->forward_comm()
```

Il metodo **decide()** della classe **Neighbor** determina se le liste dei vicini devono essere ricostruite nello step corrente; in caso questo non sia necessario, le cordinate degli atomi fantasmi vengono acquisite da ogni processore tramite il metodo **forward_comm()** della classe **Comm**. Se invece le liste dei vicini devono essere costruite, verranno eseguite diverse operazioni contenute nel codice condizionale. Tra queste, il metodo **pre_exchange()** di ogni fix definito viene richiamato per primo, generalmente per determinare l'inserimento o l'eliminazione di particelle dal sistema. Le condizioni periodiche al contorno vengono quindi applicate dalla classe **Domain**, tramite il

metodo **pbc()** per riposizionare all'interno del box le particelle che ne siano uscite. E' importante notare che questo procedimento non viene effettuato ad ogni passo, ma solamente quando le liste dei vicini vengono ricostruite; per questo motivo le coordinate atomiche possono trovarsi leggermente fuori dal box di simulazione, una volta effettuato il dump. Questo processo avviene in modo tale che ogni processore ottenga cordinate atomiche consistenti sia per i suoi atomi che per quelli fantasma. In caso sia necessario, il contorno del box viene poi reinizializzato attraverso il metodo **reset_box()** della classe **Domain**; infatti è possibile che i bordi del box di simulazione si siano ristretti attorno alle cordinate dell'atomo corrente. Infine, le informazioni riguardanti gli atomi fantasma e le liste dei vicini vengono aggiornate tramite i metodi **setup()** e **setup_bins()**, appartenenti rispettivamente alle classi **Comm** e **Neighbor**; questo aggiornamento è necessiaro in caso si voglia cambiare la dimensione o la forma del box di simulazione. Il codice è ora pronto per valutare le nuove posizioni di atomi che abbiamo cambiato il loro dominio con il processore associato. Tale operazione viene eseguita dal metodo **exchange()** della classe **Comm**, mentre Il metodo **borders()** identificherà gli atomi fantasma che circondano ciascun sottodominio e comunicherà le informazioni correspondenti. Viene quindi eseguito un ciclo su tutti gli atomi assegnati ad un dato processore per compilare le liste da inviare a quelli adiacenti; al seguente passo, le liste vengono quindi utilizzate dal metodo **Comm::forward_comm()**. Successivamente, fix con un metodo **pre_neighbor()** vengono richiamati per ricostruire alcune struttu-

re dati dipendenti dagli atomi attualmente presenti. Al termine di questa fase, LAMMPS è quindi pronto a ricostruire le liste dei vicini con il metodo **build()** della classe **Neighbor**, generalmente impilando tutti gli atomi assegnati e quelli fantasma.

```
force_clear()
fix->pre_force()

pair->compute()
bond->compute()
angle->compute()
dihedral->compute()
improper->compute()
kspace->compute()
comm->reverse_comm()

fix->post_force()
fix->final_integrate()
fix->end_of_step()

if any output on this step: output->write()
```

In questa ultima parte di codice, vengono quindi valutate tutte le forze di interazione tra le particelle, una volta azzerato il vettore della forza agente sui

singoli atomi attraverso il metodo **force_clear()**. Se il parametro newton è impostato su "on" dal comando omonimo, verranno calcolate sia le forze agenti sugli atomi assegnati che su quelli fantasma. Le forze di coppia vengono calcolate per prime con un prodotto scalare delle cordinate atomiche e delle forze, mediante il metodo **Pair::compute()**, permettendo una valutazione più semplice del viriale globale, nel caso che questo venga richiesto; includendo sia gli atomi assegnati che quelli fantasma, viene poi incluso in maniare corretta l'effetto delle condizioni periodiche al bordo. Successivamente vengono quindi valutate le interazioni della topologia molecolare (legami, angoli, diedri, impropri) e infine le interazioni Coulombiane a lungo raggio, invocate dalla classe **KSpace**. Se il parametro newton è attivo, le informazioni riguardanti le forze agenti sugli atomi fantasma verranno comunicate e sommate ai corrispondenti atomi assegnati; questa operazione verrà effettuata dal metodo **reverse_comm()** della classe **Comm**. Infine, eventuali constraint vengono applicatio dai fix che hanno il metodo **post_force()**. Si procede quindi con la seconda metà dell'integrazione Velocity Verlet, caratterizzata dall'aggiornamento delle velocità dopo metà passo, mediante alcuni fix come nve, nvt e npt; alla fine del timestep, fix che possiedono il metodo **end_of_step()** verranno invocati. Questi fix generalmente effettuano un calcolo di diagnostica come i fix ave/time e ave/spatial. Al termine vengono generati tutti gli output richiesti, con il metodo **write()** della classe **Output**. 3 sono i possibili output: un output termodinamico inviato a schermo e a un file di log, una istantanea dei dati degli atomi inviati a un file di dump,

e infine un file di restart.

In caso si voglia effettuare una minimizzazione energetica, il ciclo impiegato sarà molto simile: dapprima vengono calcolate le forze, le liste dei vicini vengono costruite quando necessarie, gli atomi migrano verso i nuovi processori e le cordinate atomiche e infine le forze vengono comunicate ai processori adiacenti. La differenza principale consiste in quelle operazioni invocate dalla classe **Fix** e in quali tempi; infatti, solo una parte di fix di LAMMPS sono utili durante la minimizzazione di energia. I metodi rilevanti della classe **Fix** sono **min_pre_exchange()**, **min_pre_force()** e **min_post_force()** e ognuno di questi verrà invocato nel punto appropriato del ciclo.

Capitolo 4

Decomposizione spaziale e scelta della griglia

Le simulazioni di dinamica molecolare si prestano in modo naturale al calcolo parallelo. In particolare due caratteristiche dei sistemi in esame sono state sfruttate per la costruzione di algoritmi di parallelizzazione. La prima consiste nel fatto che le forze sono limitate nella distanza, *i.e.* ogni atomo interagisce solo con gli atomi più vicini geometricamente. Solidi e liquidi sono spesso modellati in questo modo, a causa dello screening elettronico, o per evitare il costo computazionale di includere forze coloumbiane a lungo raggio. Per dinamiche a corta distanza, il costo computazionale per timestep è proporzionale al numero di atomi N. La seconda proprietà consiste nel fatto che gli atomi possono essere soggetti a notevoli spostamenti durante una singola simulazione. Questo fenomeno può essere dovuto alla diffusione in un solido o in un liquido, oppure a cambi di conformazione in una molecola biologica. Da un punto di vista computazionale, il concetto importante è che i vicini di ogni atomo cambiano al progredire della simulazione. Tenere nota dei vicini di ogni atomo in modo continuo e allo stesso tempo mantenere una

efficienza computazionale $\mathcal{O}(N)$ può essere particolarmente complicato.

Oltre a questi due fattori, l'idea alla base degli algoritmi sviluppati è quella di essere applicabile anche a sistemi piccoli, che meno si prestano alla parallelizzazione diretta. Questo perché, la strategia generale è quella di scegliere il sistema con il minor numero di atomi, ma che sia rappresentativo del campione reale. Di conseguenza, l'idea è di privilegiare simulazioni lunghe con modelli piccoli piuttosto che il contrario. D'altro canto, per sistemi molto grandi, un secondo obiettivo è quello di sviluppare algoritmi di parallelizzazione che siano scalabaili per macchine sempre più grandi e veloci. Gli algoritmi descritti scalano in modo ottimale per N e P, dove P rappresenta il numero di processori. In questo modo, queste procedure risulteranno utili anche quando aumenterà la potenza di calcolo.

Come accennato precedentemente, le simulazioni di dinamica molecolare consistono essenzialmente nell'integrare il seguente set di equazioni differenziali di Newton

$$\begin{aligned} m_i \frac{\mathrm{d}\vec{v}_i}{\mathrm{d}t} &= \sum_j F_2(\vec{r}_i, \vec{r}_j) \sum_k F_3(\vec{r}_i, \vec{r}_j, \vec{r}_k) + \cdots \\ \frac{\mathrm{d}\vec{r}_i}{\mathrm{d}t} &= \vec{v}_i \end{aligned} \tag{4.1}$$

Ricordiamo che F_2 è una funzione di forza che descrive l'interazione di coppia tra gli atomi, F_3 descrive l'interazione a tre corpi, ed eventualmente

si possono aggiungere interazioni a molti corpi. I termini delle forze sono derivate di formulazioni di energia dove l'energia dell'atomo i è tipicamente scritta come una funzione della sua posizione e di quella degli altri atomi. In pratica, solo uno o pochi termini dell'equazione vengono mantenuti e F_2 e F_3 sono costruiti in modo tale da includere gli effetti quantistici e a molti corpi. I termini di forza nell'equazione 4.1 sono tipicamente funzioni non lineari della distanza r_{ij} tra coppie di atomi e possono essere a corto o a lungo raggio. Per le forze a lungo raggio, come le interazioni Coloumbiane in un solido ionico o in un sistema biologico, ogni atomo interagisce con tutti gli altri. La valutazione diretta di queste forze scala come N^2 e risulta troppo costosa per un gran numero di atomi N. Alcuni metodi di approssimazione permettono di risolvere questo problema. Tra questi ricordiamo: gli algoritmi particella-griglia che scalano come $f(M)N$, dove M è il numero dei punti di griglia; metodi gerarchici che scalano come $N\log(N)$; oppure metodi a multipolo rapido che scalano come N.

Implementazioni parallele di questi algoritmi, hanno migliorato il loro ambito di applicazione, tuttavia, a causa del loro costo, modelli con forze a lungo raggio non sono comunemente usati in simulazioni di dinamica classica; per contrasto, modelli caratterizzati da forze a corto raggio vengono impiegati in modo approfondito. Questi vengono scelti sia perché lo screening elettronico effettivamente limita il raggio di influenza delle forze interatomiche, o semplicemente perché le forze a lungo raggio vengono troncate per diminuire il carico computazionale. In ogni caso, le somme nell'equazione

4.1 vengono limitate agli atomi in una qualche regione piccola che circonda l'atomo i. Questo viene tipicamente implementato usando una distanza di cutoff r_c, fuori dalla quale tutte le interazioni vengono trascurate. Il lavoro di valutare le forze ora scala linearmente con N. Malgrado questa approssimazione, la stragrande maggioranza del tempo computazionale viene speso comunque nella valutazione dei termini di forza. L'integrazione nel tempo tipicamente richiede solo 2-3% del tempo totale. Per valutare le somme in modo efficiente, è richiesta la conoscenza di quali atomi si trovano all'interno della distanza r_c ad ogni intervallo. La chiave è quindi di minimizzare il numero di vicini che devono essere controllati per possibili interazioni dato che operazioni compiute sui vicini alla distanza $r > r_c$ sono operazioni perse.

Due tecniche sono state impiegate per risolvere questo problema su macchine vettoriali e seriali. Poiché gli algoritmi paralleli incorporano simili idee ne portiamo una breve descrizione. La prima idea consiste nel creare una lista di vicini per ogni atomo. Tipicamente, quando la lista viene formata, vengono salvati tutti gli atomi vicini caratterizzati da una distanza di cutoff estesa $r_s = r_c + \delta$. La lista viene usata per un piccolo numero di intervalli per calcolare tutte le interazioni di forza. Dopodiché viene ricostruita prima che ogni atomo possa essersi mosso da una distanza $r > r_s$ ad una $r < r_c$. Il parametro δ viene sempre scelto piccolo rispetto a r_c, tuttavia il valore ottimale dipende da diversi parametri (*i.e.* temperatura, diffusione, densità). Il vantaggio di utilizzare la lista di vicini è che una volta che questa è stata costruita, esaminarla per possibili interazioni è molto più rapido che control-

lare tutti gli atomi del sistema. La seconda tecnica comunemente impiegata è conosciuta come il metodo a celle collegate. Ad ogni timestep, tutti gli atomi vengono inseriti in celle 3-D di lungezza $d = r_c$ o leggermente più grande. Questo riduce l'impegno di trovare i vicini di un dato atomo a cercarli in 27 regioni, quella contenente l'atomo stesso e le 26 che lo circondano.

Gli algoritmi impiegati utilizzano normalmente una combinazione di questi due metodi. Un risparmio aggiuntivo si può ottenere grazie alla terza legge di Newton, ovvero valutando solo la forza per ogni coppia di atomi, piuttosto che una volta per ogni atomo nella coppia. Nel metodo misto, questo si ottiene cercando solo metà dei contenitori che circondano ogni atomo per formare la lista dei vicini. Questo ha l'effetto di conservare l'atomo j nella lista dell'atomo i, ma non viceversa, dimezzando in questo modo il numero di valutazioni della forza che deve essere fatto.

A parte queste tecniche, algoritmi sono stati sviluppati per sfruttare il parallelismo naturale insito nella dinamica molecolare. Questo è dovuto al fatto che la valutazione delle forze e l'aggiornamento delle velocità e delle posizioni possono essere effettuate contemporaneamente per tutti gli atomi. L'obiettivo è quindi quello di dividere il calcolo delle forze nell'equazione 4.1 in modo bilanciato su tutti i processori. Un primo metodo consiste nell'assegnare il calcolo di un gruppo predeterminato di forze ad un singolo processore; questa scelta rimane fissa per la durata della simulazione. Il modo più semplice per ottenere questo consiste nel distribuire un sottogruppo di atomi ad ogni processore. Questo metodo è definito *decomposizione per*

atomi del carico di lavoro, dato che il processore calcola le forze sui suoi atomi, indipindentemente da dove siano localizzati nella cella di simulazione. In alternativa, è possibile assegnare un sottogruppo di forze da calcolare ad un singlo processore. Questo procedimento viene definito *decomposizione per forze*. Entrambe queste decomposizioni sono analoghe allo sviluppo di griglie Lagrangiane, impiegato in simulazioni di fluidi dove le celle della griglia si muovono con il fluido. Un terzo metodo, chiamato *decomposizione spaziale* del carico di lavoro, assegna invece ad ogni processore una porzione del dominio di simulazione fisico. Ogni processore valuta solamente le for<e agenti sugli atomi del proprio sottodominio. Quest'ultimo metodo presenta notevoli vantaggi rispetto ai precedenti, ed è quello implementato in LAMMPS. Nella sezione che segue viene riportata una descrizione dettagliata.

4.1 Algoritmo di decomposizione spaziale

Con questo algoritmo, il dominio fisico di simulazione è suddiviso in piccole celle 3-D, una per ogni processore. Ognuno di questi calcolerà le forze e aggiornerà le posizioni e le velocità di tutti gli atomi all'interno della sua cella, per ogni intervallo. Gli atomi sono riassegnati ai nuovi processori quando questi si muovono nel dominio fisico. Per poter calcolare le forze agenti sugli atomi, un processore deve solo conoscere le posizioni degli atomi nelle celle adiacenti. La comunicazione richiesta in questo algoritmo è quindi di natura locale a differenza degli altri algoritmi di decomposizione. La dimensione

e la forma della cella assegnata ad ogni processore dipenderà da N, P e dall'aspetto dell dominio fisico, che assumiamo essere un parallelepipedo 3-D rettangolare. La scelta del numero di processori in ogni dimensione è tale da rendere ogni cella assegnata il più simile possibile ad un cubo; la ragione dietro questa strategia è di minimizzare le comunicazioni dato che, per valori di N grandi, il costo di queste ultime risulta essere proporzionale all'area superficiale delle celle. Bisogna notare che, a differenza del metodo delle celle collegate descritto in precedenza, le lunghezze delle celle possono ora variare rispetto ai cutoff delle forze r_c e r_s.

Ogni processore mantiene due strutture dati, una per gli atomi N/P nella sua cella e una per gli atomi in quelle adiacenti. Nella prima struttura dati, ogni processore salva informazioni come posizioni, velocità e lista dei vicini. Questi dati sono mantenuti in una lista che consente inserimenti e cancellazioni in funzione del movimento degli atomi da una cella all'altra. Nella seconda struttura, vengono invece salvate solo le posizioni degli atomi. Comunicazioni tra i processori ad ogni intervallo permettono di mantenere queste informazioni aggiornate.

Il primo passo delle comunicazioni consiste nello scambiare informazione con i processori adiacenti lungo la direzione est-ovest. Un processore riempie una porzione di messaggio con le posizioni degli atomi che gli sono associati e che si trovano ad una distanza di cutoff r_s dalla cella del processore adiacente. Se $d < r_s$ dove d è la lunghezza della cella nella direzione est-ovest, questi consisteranno in tutti gli atomi del processore adiacente; altrimenti saranno

quelli più vicini al primo processore. Dopodiché, ogni processore manda i suoi messaggi al processore localizzato ad ovest e riceve il messaggio da quello a est. Ogni processore salva le informazioni ricevute nella sua seconda struttura dati. Ora la procedura viene invertita, con ogni processore che invia i dati ad est e li riceve da ovest. Se $d > r_s$, tutte le posizioni richieste nella direzione est/ovest sono state salvate da ogni processore. Se invece $d < r_s$, i passi est-ovest sono ripetutti con un invio di un numero maggiore di atomi dai singoli processori. Questo può essere ripetuto fino a quando ogni processore conosca tutte le posizioni degli atomi entro la distanza r_s dalla sua cella. La stessa procedura viene poi ripetuta nella direzione nord-sud. La sola differenza è che i messaggi inviati al processore adiacente ora contengono non soltanto gli atomi associati col processore, ma anche ogni posizione di atomi nella sua seconda struttura, necessari al processore adiacente. Per $d = r_s$, questo ha l'effetto di inviare un numero di posizioni atomiche contenute nell'equivalente di 3 celle in un solo messaggio. Finalmente nell'ultimo passo, il processo viene ripetuto nella direzione su-giù; come risultato, le posizioni atomiche di un intero piano di celle vengono inviate in ogni singolo messaggio.

Ci sono diversi vantaggi per questo schema, ognuno dei quali riduce il costo complessivo della comunicazione. Primo, per $d \geq r_s$, le posizioni deli atomi richiesti dalle 26 celle all'intorno sono ottenute con solo 6 scambi di informazioni. Secondo, quando $d < r_s$,*i.e.* l'informazione è richiesta da celle più distanti, il messaggio avverrà solo con pochi scambi extra, i quali saranno comunque con i 6 processori più vicini. Questa caratteristica rende questo

algoritmo particolarmente efficiente anche quando un gran numero di processori sono utilizzati su problemi relativamente piccoli. Un terzo vantaggio consiste nel fatto che la quantità di dati comunicati viene minimizzata. Ogni processore riceve solo le posizioni degli atomi che si trovano nella distanza r_s dalla sua cella. Quarto, tutte le posizioni degli atomi ricevute possono essere piazzate in modo contiguo direttamente nella seconda struttura dati del processore. Nessun tempo viene consumato nel riarrangiamento dei dati, eccetto quello necessario a creare i frammenti di messaggi che devono essere inviati. Per finire, la creazione del messaggio può avvenire in tempi molto brevi. Un'analisi delle due strutture dati viene compiuta solamente dopo alcuni intervalli, quando le liste dei vicini vengono create, per decidere quali posizioni di atomi inviare. La procedura di analisi crea la lista degli atomi che compongono ogni messagio. Durante gli atri intervalli può essere impiegata la lista invece di analizzare l'intero set di atomi, per indicizzare gli atomi referenziati e comporre il messaggio in modo rapido. Questo risulta equivalente all'operazione di raccoglimento che avviene su una macchina vettoriale. A titolo di esempio riportiamo nella Figura 4.1 la decomposizione spaziale della cella di simulazione impiegando una griglia 4x2. In rosso viene evidenziata una delle 8 sottocelle che si sono formate, la quale sarà associata ad un singolo processo.

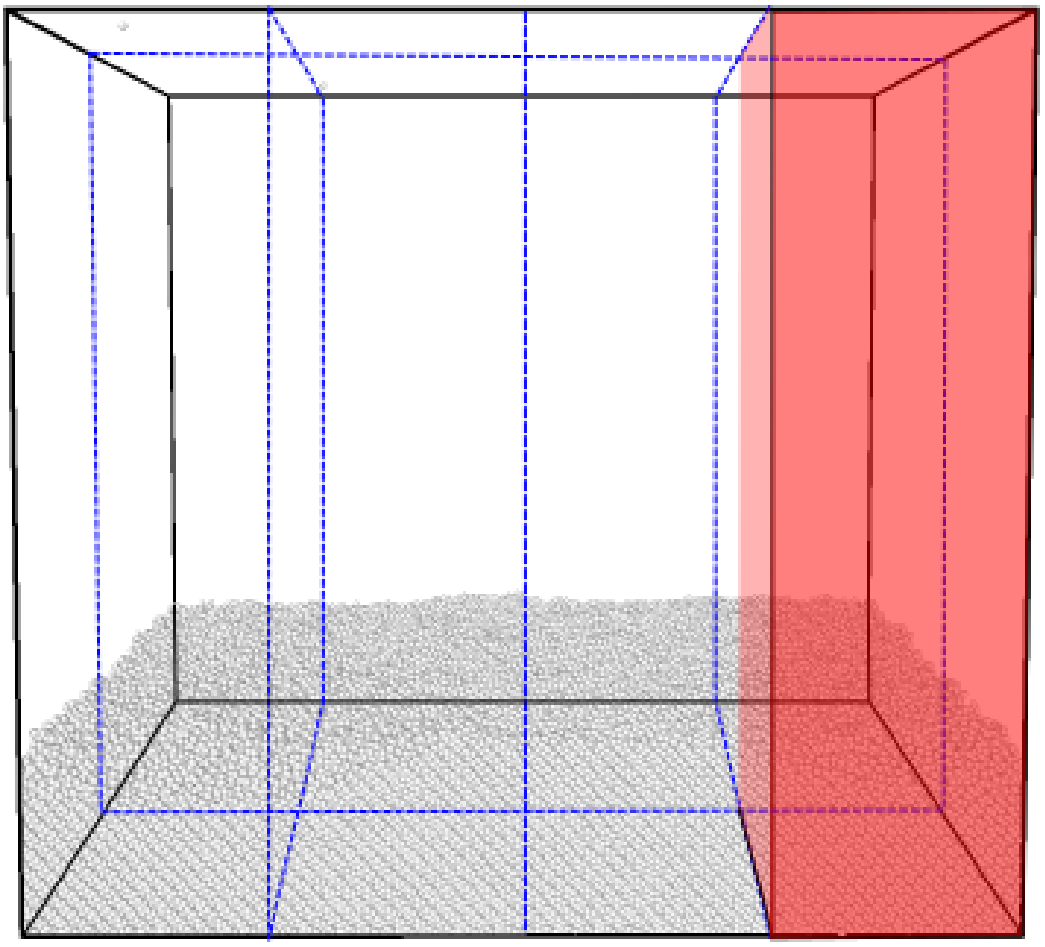

Figura 4.1: Esempio di griglia 4x2.

Capitolo 5

Dettagli Computazionali

Tutte le principali simulazioni sono state effettuate sul cluster CRESCO(Centro computazionale di RicErca sui Sistemi COmplessi) localizzato a Portici, presso l'ENEA (Ente Nazionale Energia e Ambiente). Due sistemi sono stati impiegati per le simulazioni: il cluster CRESCO4 di Portici, costituito da 304 nodi dove ogni nodo ha:

- 2 socket da 8 core con processore Intel E5-2670 2.6 GHz 64 GB/RAM (4 GB per core, 8 istruzioni per ciclo di clock),
- un disco 500GB SATA II,
- Una interfaccia IB QDR 40 Gb/s (32 Gb/s effettivi per i dati),
- Due interfacce GbE,
- Supporto BMC/IPMI 1.8 e software per la gestione remota della console.

il secondo cluster invece, chiamato CRESCO5, è un sistema di calcolo costituito da 40 nodi. Ogni nodo è caratterizzato da:

- 2 socket da 8 core con processore Intel(R) Xeon(R) CPU E5-2630 v3 con frequenza di clock pari a 2.40GHz e 64 GB/RAM (4 GB per core, 16 istruzioni per ciclo di clock),
- una interfaccia IB QDR 40 Gb/s (32 Gb/s effettivi per i dati),
- due interfacce GbE,
- supporto BMC/IPMI 1.8 e software per la gestione remota della console.

Su questi sistemi sono state eseguite dinamiche molecolari sul bulk di silicio a 773 K, impiegando l'ensemble termodinamico NPT. Il campione utilizzato contiene 8000 atomi posizionati secondo la struttura del diamante. Ne risulta una cella di simulazione cubica a cui sono state imposte condizioni periodiche al bordo. Per quanto riguarda il trattamento delle forze interatomiche, è stata applicata la versione modificata del potenziale tersoff. Ciò è stato necessario perché, implementando il potenziale standard, si ha una diminuizione di densità con il passaggio di stato da solido a liquido, contrariamente al comportamento reale. Un timestep di 10^{-3} ps è stato considerato sufficiente per una corretta simulazione del sistema. La durata totale della simulazione è stata invece di 1 μs. Al termine di questa procedura, il lato della cella è risultato essere di 54.56 Å e tale valore è stato assunto come riferimento per le simulazioni successive.

Si è quindi proceduto alla generazione della superficie, tagliando il bulk lungo il piano 001. La nuova cella contiene il medesimo numero di atomi e uno spazio vuoto di ∼200 Å lungo la direzione ortogonale alla superficie, per evitare interazioni tra un'immagine periodica e l'altra (250 Å totali lungo l'asse Z). A differenza del caso precedente, è stato implementato l'ensemble canonico (NVT).

Il sistema così definito è stato utilizzato per diverse simulazioni a differenti temperature. Inoltre, durante la singola simulazione, diverse zone di temperatura sono state considerate per la cella. Infatti, una zona *calda* (da 40 a 80 Å) è stata scaldata in maniera costante alla temperatura prefissata, mentre una seconda zona *fredda* è stata fissata a 773 K (da 0 a 15 Å), indipendentemente dalla simulazione. Infine, una zona intermedia tra queste due è stata lasciata libera di raggiungere l'equlibrio. La cella di simulazione è riportata in Figura 5.1. La zona *calda* tra 40 e 80 Å è stata suddivisa in tre porzioni per una migliore rappresentazione: azzurro (40-50 Å), magenta (50-65 Å) e verde (65-80 Å). Gli atomi nella zona di raffreddamento (80-250 Å) sono invece rappresentati in viola.

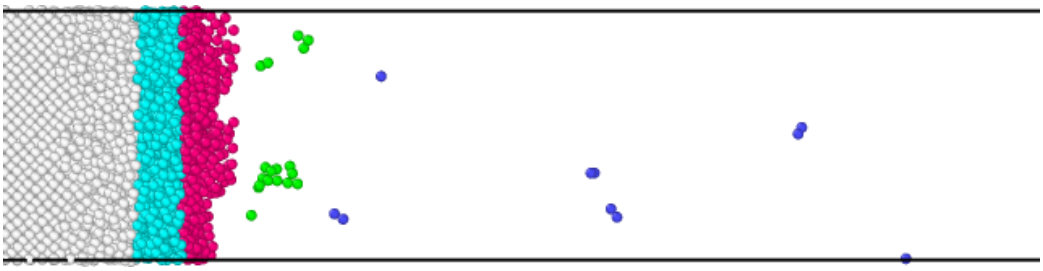

Figura 5.1: la cella di simulazione, con l'asse z diretto verso destra.

In conseguenza del riscaldamento, gruppi di atomi hanno lasciato la superficie, in quantità dipendenti dalla temperatura. Per determinare il raffreddamento di questi, da 80 Å in poi, la cella è stata divisa in 8 zone a temperatura decrescente (fino a 300 K in prossimità del bordo superiore). Per rendere il raffreddamento meno discontinuo, la temperatura è stata imposta solo a metà degli atomi presenti nella zona. Le temperature scelte per le diverse zone (con i loro bordi in Å) sono riportate in tabella 5.1

	1	2	3	4	5	6	7	8
Temp/Bordo	80-100	100-120	120-140	140-160	160-180	180-200	200-220	220-250
1573	1350	1200	1050	900	750	600	450	300
2073	1800	1600	1400	1200	1000	800	600	300
2573	2400	2100	1800	1500	1200	900	600	300
3073	2850	2500	2150	1700	1350	1000	650	300
3573	3100	2700	2300	1900	1500	1100	700	300
4073	3800	3300	2800	2300	1800	1300	800	300
4573	4000	3500	3000	2500	2000	1500	1000	300
4773	4000	3500	3000	2500	2000	1500	1000	300

Tabella 5.1: Definizione delle 8 fasce di raffreddamento nel vuoto per le varie temperature.

5.1 Efficienza della Parallelizzazione

Come accennato in precedenza, LAMMPS è particolarmente efficiente quando impiegato su sistemi contenenti un gran numero di processori connessi in parallelo. A dimostrazione di ciò, abbiamo effettuato diverse simulazioni dello stesso sistema, applicando di volta in volta una griglia di decomposizione differente ed incrementando il numero di processi impiegati. In Figura 5.2 abbiamo quindi riportato i tempi in secondi delle simulazioni in funzione del

numero di processi. Tale numero può essere differente dal numero di cores impiegati. Tuttavia, ciò comporta una complessità maggiore, conseguentemente è tipico associare un singolo core ad un singolo processo. Nel nostro caso abbiamo a disposizione nodi composti da 16 cores; di conseguenza, il sistema viene diviso applicando griglie composte da un numero di celle pari ad un multiplo di 16.

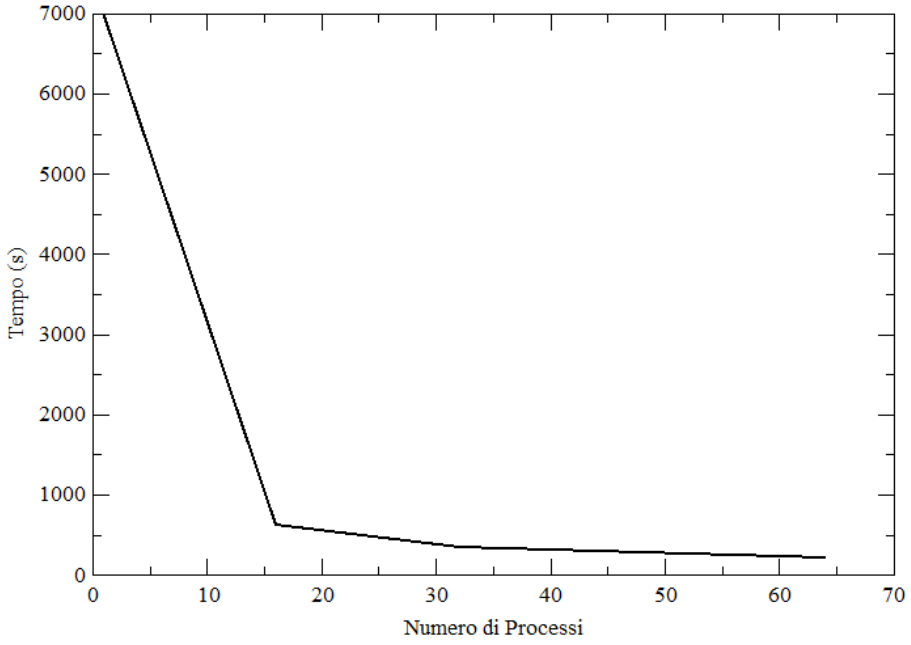

Figura 5.2: Tempo di simulazione in funzione del numero di processi/cores

Per valutare l'efficienza computazionale abbiamo anche riportato in Tabella 5.2 lo speedup, definito come $\frac{tempo(ser)}{tempo(par)}$, insieme al tempo (in secondi)

impiegato.

Numero processi	Speed Up	Tempo (s)
1 (Seriale)	1	6960
16	11.21	621
32	20.41	341
64	33.14	210

Tabella 5.2: Speed up della simulazione per diverso numero di processi.

Si può notare come, all'aumentare del numero di cores, hanno sempre più peso le comunicazioni tra i diversi nodi e tra i processori all'interno dello stesso nodo. Infatti, ad un incremento di 16, 32 e 64 cores rispetto al caso seriale, corrisponde uno speedup rispettivamente di 11, 20 e 33, con una forte deviazione rispetto al caso ideale, in cui il codice è perfettamente parallelizzato e le comunicazioni sono estremamente ridotte.

5.2 Incremento della taglia del sistema

Per avere un'idea del comportamento del codice LAMMPS nel caso di sistemi di grandi dimensioni abbiamo incrementato notevolmente la taglia del sistema, descritto nella sezione precedente, ed effettuato una serie di simulazioni impiegando diverse griglie con un numero incrementale di nodi. Il campione in esame risulta quindi costituito da una cella di lato pari a 545.63 Å lungo le direzioni x e y, e di 1200.00 Å lungo l'asse z, dove è stato aggiunto uno spazio vuoto sufficiente ad evitare interazioni tra le immagini periodiche. Il sistema costruito in questo modo contiene 8×10^6 atomi. Per le diverse simulazioni

abbiamo impiegando il cluster Marconi del consorzio interuniversitario CINECA, particolarmente adatto a calcoli paralleli di grandi dimensioni, avendo le seguenti caratteristiche:

- 2 socket da 18 cores Intel Xeon E5-2697 v4 (Broadwell) con un frequenza di 2.30 GHz
- 128 GB di memoria per nodo, 3.5 GB per core
- Peak Performance: circa 1 PFlop/s

L'utilizzo di tale cluster, oltre a permettere lo studio di sistemi di grandi dimensioni, ci dà anche l'occasione di osservare il comportamento del codice LAMMPS quando eseguito su diverse architetture. Da notare che, oltre ad impiegare nodi da 36 cores, abbiamo utilizzato un gran numero di nodi fino ad un totale di 72, per un numero di processori complessivo pari a 2592 processori.

In Tabella 5.3 vengono quindi riportati i tempi delle diverse simulazioni e lo speedup relativo al caso con il minor numero di nodi. Ricordiamo che tale quantità dovrebbe essere valutata in funzione del tempo impiegato per un calcolo in seriale; tale simulazione richiederebbe però tempi eccessivamente lunghi nel caso in esame.

Dai numeri della tabella si può osservare che l'incremento da 18 a 36 nodi comporta circa un dimezzamento dei tempi di calcolo, significativo di un'ottima efficienza di parallelizzazione. Come per il caso descritto nella sezione

Numero nodi	Speed Up	Tempo (s)
9	1	67855
18	1.92	35408
36	3.30	20541
72	5.84	11617

Tabella 5.3: Speed up della simulazione per diverso numero di nodi.

precedente, aumentando il numero di nodi si può notare l'aumento progressivo dei costi delle comunicazioni tra i nodi e all'interno del singolo nodo. Infatti, quadruplicando il numero di nodi otteniamo una velocità di poco superiore a 3 volte rispetto al caso base, mentre ad un incremento del numero di nodi pari ad 8, corrisponde uno speedup inferiore a 6. Questa deviazione rispetto alla linearità è ancora più evidente nel grafico rappresentato in Figura 5.3, dove sono riportati i tempi di simulazione in funzione del numero di nodi.

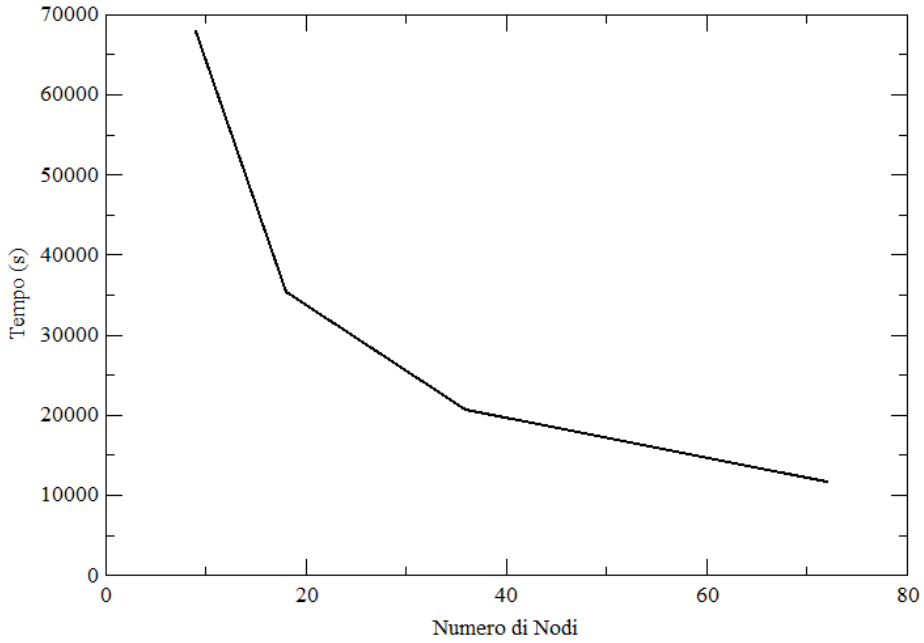

Figura 5.3: Tempo di simulazione in funzione del numero di nodi.

Capitolo 6

Simulazioni e risultati

6.1 Riproduzione dell'esperimento

L'esperimento descritto nelle sezioni precedenti consiste in un sistema chiuso caratterizzato da una lamina di silicio con uno spessore nell'ordine dei micrometri, verso cui vengono diretti dei protoni ad alta energia, generati da un impulso laser applicato su un materiale opportuno. Al fine di approfondire il meccanismo alla base della formazione di nanoparticelle, abbiamo eseguito una serie di simulazioni replicando le condizioni sperimentali. Per ottenere risultati compatibili con le osservazioni sperimentali, è stato impiegato un campione più grande di quello descritto in precedenza, ottenuto replicando la cella lungo il piano x-y per 5 volte. Il campione risultante consiste quindi di 200000 atomi distribuiti in una cella di dimensioni $\approx 278\times\ 278\times\ 250$ Å. Seguendo la procedura descritta da Barberio *et al.*, la temperatura del campione è stata dapprima incrementata in modo costante, a partire da 773 K, in un tempo molto rapido (10 ps). Questo riscaldamento corrisponde all'impulso del laser sul campione reale. Al termine di questa fase, la temperatura del sistema ha raggiunto i 1723 K, poco al di sopra della temperatura

di fusione. il sistema è stato quindi mantenuto a tale temperatura per un tempo pari a 570 ps, al termine del quale è stato eseguito un raffreddamento rapido (80 ps) a 1473 K, al di sotto della temperatura di fusione. Tali tempi sono stati ottenuti scalando quelli impiegati durante l'esperimento, in modo dal replicare quest'ultimo nella maniera più fedele possibile. In Figura 6.1 sono state riportate delle istantanee del sistema a tempi diversi. Gli atomi colorati sono connessi tra di loro in cordinazione tetraedrica tipica del silicio allo stato solido. L'istantanea a) rappresenta il campione al tempo iniziale, mentre quella b) riporta il campione al termine del riscaldamento, dopo 10 ps. Dopo tale intervallo, l'ordine superficiale è parzialmente perturbato, come si vede dalla presenza di alcune zone incolori. L'istantanea c) rappresenta invece il sistema in piena fase liquida (t = 230 ps), caratterizzato dalla presenza di aree superficiali ordinate molto limitate; infatti, gran parte degli atomi possiede una grande mobilità dovuto alla fase liquida del materiale. E' interessante notare che, una volta superato tale punto, il sistema tende a riformare la struttura cristallina in alcune parti, mentre le zone liquide si concentrano in bolle ben definite, fissate in fase di raffreddamento, come si può osservare dall'istantanea d) salvata alla fine della simulazione (660 ps). Dato l'aspetto ben localizzato e le dimensioni compatibili con le osservazioni sperimentali, l'ipotesi più probabile è che in tali bolle si possano venire a formare le nanoparticelle per tempi di simulazione molto lunghi.

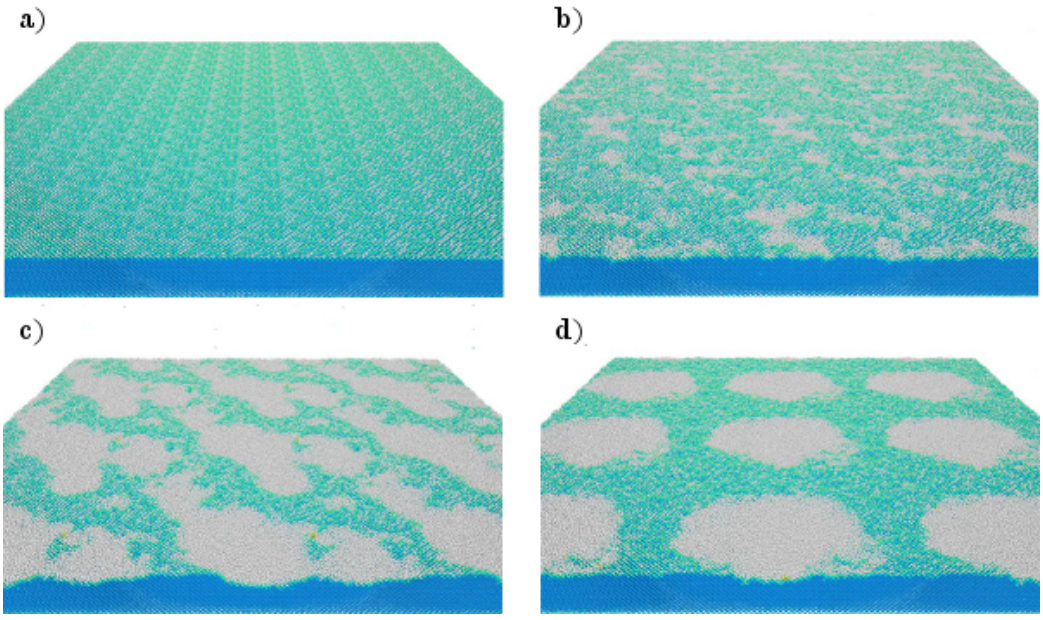

Figura 6.1: Instantanee della simulazione di riscaldamento salvate a differenti tempi: a) 0ps; b) 10 ps; c) 230 ps; d) 660 ps.

6.2 Analisi dei risultati

Dopo aver replicato l'esperimento, abbiamo eseguito uno studio su un campione del sistema più limitato. A tal fine, diverse simulazioni sono state eseguite impiegando 8 temperature differenti, in modo da osservare il comportamento del sistema lungo tutta la transizione di fase solido-liquido. La zona *calda* del sistema è stata scaldata in modo costante, fino a raggiungere la temperatura stabilita. In un secondo tempo, l'intero sistema viene lasciato equilibrare per i successivi 100 ps (Figura 6.2). Il riscaldamento è stato mantenuto costante per un migliore confronto, di conseguenza tempi diversi sono stati necessari per raggiungere la temperatura di equilibrio.Avendo impiegato

un riscaldamento costante, tempi diversi sono stati necessari per raggiungere la temperatura di equilibrio. I risultati ottenuti sono stati analizzati nelle sezioni successive.

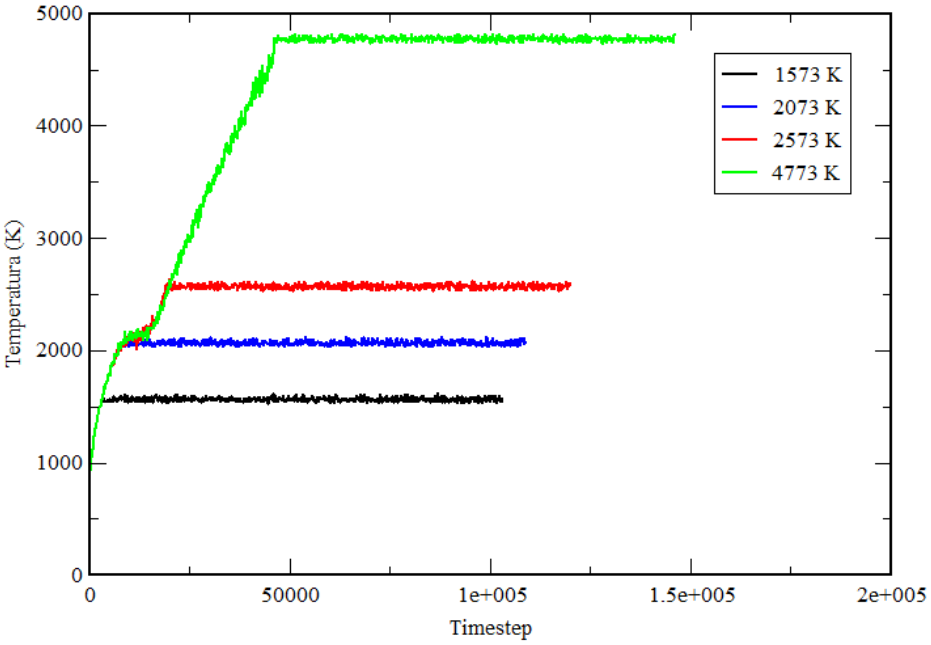

Figura 6.2: Temperatura delle simulazioni come funzione del Timestep.

6.3 Atomi uscenti

In seguito al riscaldamento gruppi di atomi hanno lasciato la superficie. Il numero degli atomi uscenti sono stati riportati nella tabella 6.1 e graficati in figura 6.3. In rosso sono riportati i valori rilevati alla *soglia*, mentre in

nero quelli al termine della simulazione.Due tempi sono stati considerati per tale analisi: la *soglia* al termine del riscaldamento e la fine della simulazione (*soglia* + 100 ps). Dal grafico corrispondente si nota un deciso andamento esponenziale del fenomeno. Come valore di riferimento abbiamo considerato come uscenti gli atomi ad una distanza di almeno 65 Å dal bordo inferiore della cella.

Temperatura	*soglia*(ps)	Atomi(*soglia*)	Atomi(*soglia*+100ps)
3073	25	0	4
3573	31	0	6
4073	39	1	47
4573	45	3	151
4773	46	14	244

Tabella 6.1: Atomi uscenti per le varie temperature.

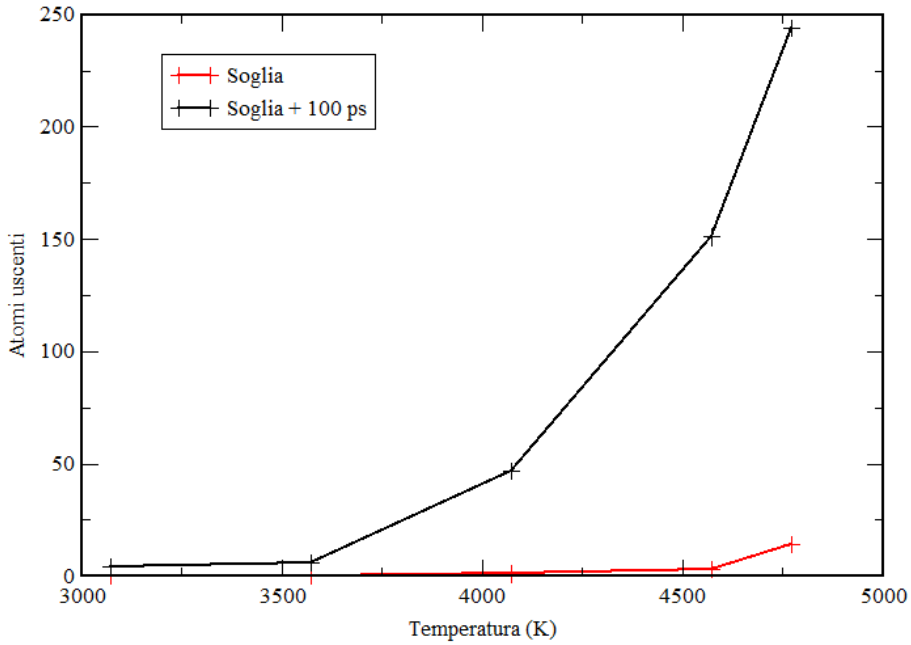

Figura 6.3: Atomi uscenti in funzione dalla temperatura.

6.4 Energia cinetica media

È' stata calcolata l'energia cinetica media alle diverse temperature per gli atomi presenti in due zone differenti: 50-65 Å e 65-80 Å. I risultati sono riportati in figura 6.4. IIn nero sono riportate quelle relative agli atomi presenti tra 50 e 65 Å mentre in rosso quelle per gli atomi tra 65 e 80 Å. Per le temperature tra 1573 K e 2573 K, l'energia cinetica nella seconda zona è nulla perché nessun atomo lascia la superficie, come descritto nella sezione

precedente. Se escludiamo tali punti, entrambi i set mostrano andamento pressoché lineare.

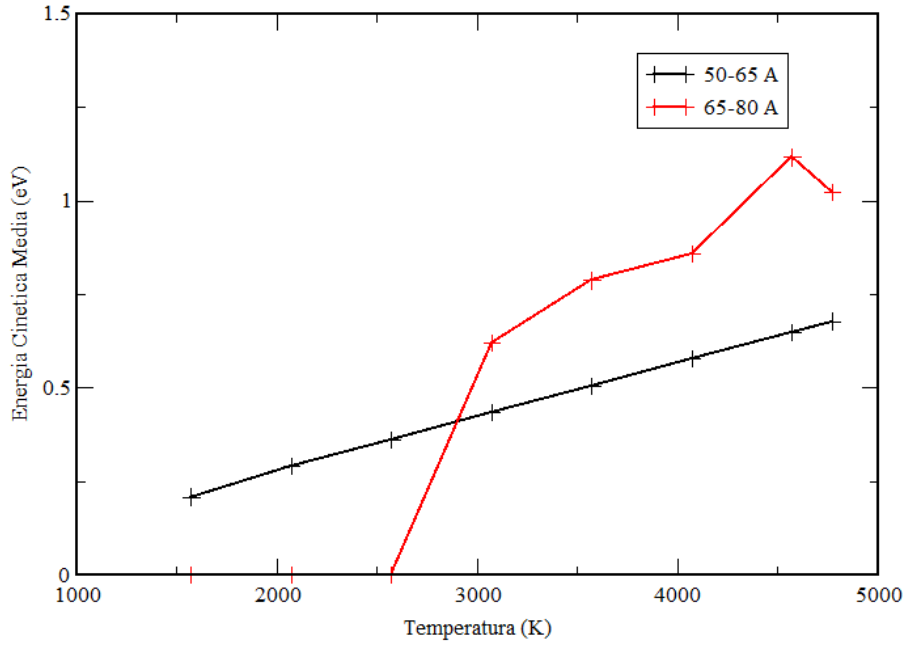

Figura 6.4: Energia cinetica media alle varie temperature.

6.5 Mobilità

Come ultima analisi, abbiamo studiato la mobilità del sistema alle diverse temperature. Abbiamo preso in considerazione solo gli atomi più superficiali (Z compresa tra 50 e 65 Å). Per questi ultimi, è stato misurato lo spostamento

medio rispetto alla configurazione di riferimento, al termine del riscaldamento (figura 6.5). In rosso e nero è rappresentato lo spostamento medio sull'asse x e y degli atomi, mentre in verde è rappresentato quello nella direzione z. In blu è riportato lo spostamento medio totale. Come si può vedere dal grafico, intorno ai 1700 K abbiamo un'aumento progressivo della mobilità lungo l'asse z(in verde), in conseguenza della transizione di stato solido-liquido. Spostamenti lungo gli assi x e y oscillano invece in modo incrementale intorno alla posizione di equilibrio.

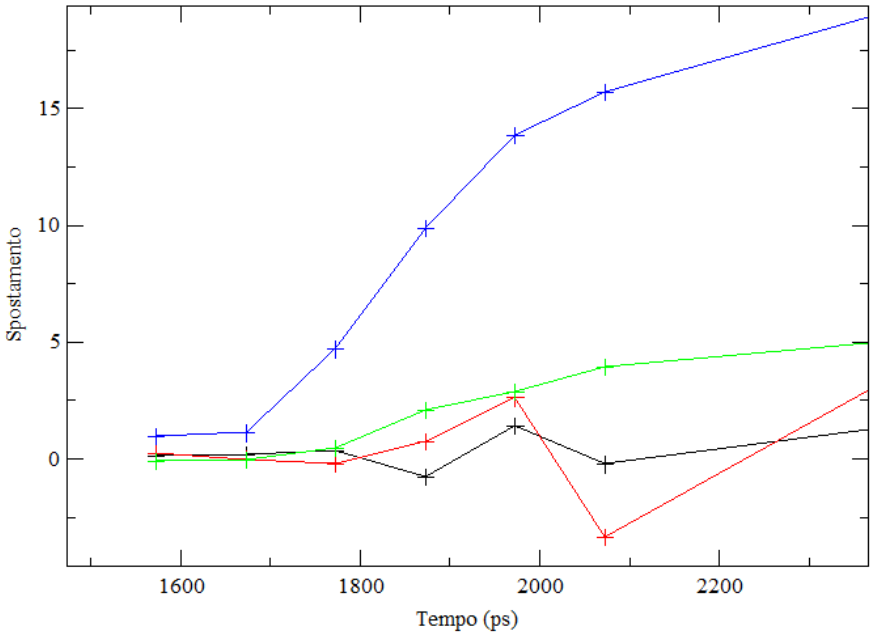

Figura 6.5: Mobilità del sistema alle diverse temperature.

Conclusioni

In questa tesi abbiamo studiato il codice LAMMPS, come implementazione della metodologia della dinamica molecolare. Questo metodo si basa sulla risoluzione di un sistema di equazioni differenziali di Newton, e permette di modellare sistemi atomici di grandi dimensioni e simularli per un tempo molto lungo. Questo rappresenta un grande vantaggio rispetto ai metodi ab-initio, basati sulla risoluzione dell'equazione di Schroedinger e la valutazione della funzione d'onda totale del sistema. Infatti, a parità di risorse computazionali, questi ultimi permettono uno studio più realistico e dettagliato di sistemi microscopici, ma si limitano a campioni molto più piccoli e a simulazioni di breve durata. Numerosi sono i codici che implementano la dinamica molecolare, ma tra questi LAMMPS è caratterizzato da una notevole versatilità grazie alla sua natura open-source, che ne permette la facile estensione con nuove caratteristice, e alla sua efficienza computazionale, che lo rende particolarmente adatto alla parallelizzazione su un grande numero di processori.

Come sistema in esame abbiamo considerato la formazione di nanoparticelle su di una superficie di silicio a formare una struttura conosciuta come nanocarpet. Questa composizione è ottenibile tramite l'impiego di un laser caratterizzato da un impulso nell'ordine dei ps. L'utilizzo di questa tecnica per la formazione di nanoparticelle, in particolare su superfici semiconduttri-

ci, rappresenta una novità rispetto all'utilizzo classico che se ne è fatto finora. Questo processo è stato compiuto sperimentalmente dal gruppo di Barberio *et al.*[4] e lo studio riportato in questa tesi permette un'analisi più dettagliata di quelli che sono i meccanismi alla base del processo sperimentale. Seguendo la metodologia impiegata in quest'ultimo, è stato costruito un modello della superficie di silicio, sottoposto poi ad un selettivo riscaldamento, a simulare l'impulso laser. La simulazione ha rispecchiato una buona fedeltà con quelli che sono i riscontri sperimantali. In particolare, abbiamo notato la formazione di bolle in fase liquida sulla superficie del sistema, ben localizzate e di dimensioni comparabili con le nanoparticelle rilevate dall'esperimento. Inoltre, impiegando un campione più limitato del sistema in simulazioni a diverse temperature lungo la transizione solido-liquido, sono stati osservati alcuni trend molto interessanti. Possiamo notare ad esempio un andamento esponenziale per quanto riguarda gli atomi uscenti dalla superficie in seguito al riscaldamento progressivo, mentre l'energia cinetica degli atomi superficiali presenta un andamento lineare. Se consideriamo poi la mobilità degli atomi, ovvero lo spostamento medio rispetto alla posizione di equilibrio, viene rilevato un progressivo aumento di questa quantità rispetto alla direzione perpendicolare alla superficie, conseguenza della transizione di stato solido-liquido; gli spostamenti lungo il piano oscillano invece in modo incrementale intorno alla posizione di equilibrio.

Bibliografia

[1] Lian-Yi Chen, Jia-Quan Xu, Hongseok Choi, Hiromi Konishi, Song Jin, and Xiao-Chun Li. Rapid control of phase growth by nanoparticles. *Nature Communications*, 5, 2014.

[2] Xiaogang Xue, R. Lee Penn, Edson Roberto Leite, Feng Huang, and Zhang Lin. Crystal growth by oriented attachment: kinetic models and control factors. *CrystEngComm*, 16:1419–1429, 2014.

[3] `lammps.sandia.gov`.

[4] M. Barberio, S. Vallières, M. Scisciò, and P. Antici. Nanostructure generation using laser-accelerated protons. *in press*.

[5] Alessandra Imbrogno, Rajesh Pandiyan, Marianna Barberio, Anastasia Macario, Assunta Bonanno, and My Ali El khakani. Pulsed-laser-ablation based nanodecoration of multi-wall-carbon nanotubes by co–ni nanoparticles for dye-sensitized solar cell counter electrode applications. *Materials for Renewable and Sustainable Energy*, 6(2):11, 2017.

[6] M. Barberio, D.R. Grosso, A. Imbrogno, and F. Xu. Preparation and photovoltaic properties of layered tio2/carbon nanotube/tio2 photoanodes for dye-sensitized solar cells. *Superlattices and Microstructures*, 91:158 – 164, 2016.

[7] Marianna Barberio, Pasquale Barone, Alessandra Imbrogno, and Fang Xu. Co2 adsorption on silver nanoparticle/carbon nanotube nanocomposites: A study of adsorption characteristics. *physica status solidi (b)*, 252(9):1955–1959, 2015.

[8] Hua Tong, Shuxin Ouyang, Yingpu Bi, Naoto Umezawa, Mitsutake Oshikiri, and Jinhua Ye. Nano-photocatalytic materials: Possibilities and challenges. *Advanced Materials*, 24(2), 2012.

[9] Feng-Jia Fan, Liang Wu, and Shu-Hong Yu. Energetic i-iii-vi2 and i2-ii-iv-vi4 nanocrystals: synthesis photovoltaic and thermoelectric applications. *Energy Environ. Sci.*, 7:190–208, 2014.

[10] J Baumgartner, A Dey, PH Bomans, C Le Coadou, P Fratzl, NA Sommerdijk, and D Faivre. Nucleation and growth of magnetite from solution. *Nat Mater.*, 12:310–314, 2013.

[11] E.J. Teo, Breesem M B H, E P Tavernier, A A Bettiol, and F Watt. boh. *Appl Phys Lett*, 84:3202, 2004.

[12] R. Dolbec, E. Irissou, M. Chaker, D. Guay, F. Rosei, and M. A. El Khakani. Growth dynamics of pulsed laser deposited pt nanoparticles on highly oriented pyrolitic graphite substrates. *Phys. Rev. B*, 70:201406, Nov 2004.

[13] Pratik Chaturvedi, Keng H. Hsu, Anil Kumar, Kin Hung Fung, James C. Mabon, and Nicholas X. Fang. Imaging of plasmonic modes of silver

nanoparticles using high-resolution cathodoluminescence spectroscopy. *ACS Nano*, 3(10):2965–2974, 2009. PMID: 19739603.

[14] R.A.D. Mackenzie. *Nanotechnology*, 1:163, 1990.

[15] A. Pyatenko, K. Shimokawa, M. Yamaguchi, O. Nishimura, and M. Suzuki. Synthesis of silver nanoparticles by laser ablation in pure water. *Applied Physics A*, 79(4):803–806, Sep 2004.

[16] M. Tuckerman, B. J. Berne, and G. J. Martyna. Reversible multiple time scale molecular dynamics. *The Journal of Chemical Physics*, 97(3):1990–2001, 1992.

[17] H.F. Trotter. On the product of semi-groups of operators. *Proc.Amer.Math.Soc.*, 10:545–551, 1959.