SAPIENZA
UNIVERSITÀ DI ROMA

Simulazione di Modelli SBML con LAMMPS

Facoltà di Ingegneria dell'informazione, Informatica e Statistica

Corso di Laurea in Informatica

Candidato

Leonardo Colosi
Matricola 1799057

Relatore

Prof. Enrico Tronci

Anno Accademico 2021/2022

Tesi non ancora discussa

Simulazione di Modelli SBML con LAMMPS
Tesi di Laurea. Sapienza – Università di Roma

Questa tesi è stata composta con LaTeX e la classe Sapthesis.

Email dell'autore: colosi.1799057@studenti.uniroma1.it

Sommario

Lo scopo del progetto è quello di creare un simulatore *Agent Based* per sistemi biologici che funzioni prendendo in input file SBML e restituendo in output codice eseguibile LAMMPS. Tale codice definirà una simulazione, basata sui dati estrapolati dal modello, che una volta eseguita mostrerà l'evoluzione del sistema con il passare del tempo. Tramite questo approccio è possibile interpretare modelli biochimici anche molto complessi, ricchi di specie diverse e di reazioni tra elementi appartenenti a queste specie. La modellazione Agent Based offre una visione intuitiva del sistema all'interno del quale ogni specie è considerata come un agente autonomo che rimane indipendente dagli altri agenti che lo circondano fino al momento in cui non vi entra in contatto diretto. Sono proprio i contatti tra gli agenti il punto centrale della simulazione in quanto, rappresentando concettualmente le reazioni tra le specie, sono quelli che permettono al sistema stesso di mutare. L'indipendenza dei singoli agenti, specialmente quella relativa al movimento all'interno dell'ambiente di simulazione, permette di introdurre un livello di casualità controllata e riproducibile che aggiunge realismo alle simulazioni e le rende rilevanti per l'osservazione del progredire di un sistema del quale si erano semplicemente definite alcune regole e condizioni iniziali. Questo meccanismo consente infatti di verificare o confutare ipotesi sullo stato finale del sistema, formulate solo alla luce della definizione di uno stato iniziale ma non di tutti i possibili stati intermedi.

Indice

1 Introduzione **1**
1.1 Contesto . . . 2
1.2 Motivazioni . . . 3
1.3 Contributi . . . 3
1.4 Progetti Correlati . . . 3
1.5 Struttura del Progetto . . . 4

2 Prerequisiti **5**

3 Metodi **6**

4 Implementazione **8**

5 Risultati Sperimentali **17**
5.1 Obiettivi . . . 18
5.2 Configurazione . . . 20
5.3 Casi di Studio . . . 24
5.4 Correttezza . . . 26
5.5 Stima Computazionale . . . 29
5.6 Valutazione Tecnica . . . 30

6 Conclusioni **32**

Bibliografia **33**

Capitolo 1

Introduzione

In questo primo capitolo della relazione sarà dato un quadro generale del contesto sotto il quale si è svolta la progettazione del sistema software per fornire poi le motivazioni che hanno portato alla sua realizzazione. Si parlerà dei contributi apportati dal sistema nell'ambito della possibili applicazione, così come si farà cenno ai lavori esterni correlati da cui si sono tratti spunti.
Nel capitolo seguente (2), saranno delineati i prerequisiti necessari ad una comprensione completa del lavoro svolto, fornendo allo stesso tempo il materiale di riferimento necessario ad ottenere i suddetti requisiti.
Si passerà poi all'illustrazione, nel capitolo 3, dei metodi formali adottati nella fase di progettazione. La scelta di questi metodi sarà giustificata delineando e mettendo a confronto i possibili approcci alla modellazione biologica ad oggi esistenti.
In seguito, nel capitolo 4, saranno affrontate direttamente le scelte implementative attuate nella fase di realizzazione del progetto software. Anche queste scelte verranno giustificate ed avvalorate dalla discussione diretta di alcuni degli algoritmi impiegati sotto forma di pseudo codice.
In conclusione verranno mostrati i risultati sperimentali (5), spiegandone gli obiettivi e descrivendo sia la configurazione dell'ambiente software che tutti i casi di studio affrontati, al fine di rendere gli esperimenti effettuati completamente verificabili e riproducibili. Alla luce dei dati ottenuti sarà fornito un giudizio sulla correttezza e sull'efficienza del sistema così come una valutazione sugli aspetti tecnici del progetto.

1.1 Contesto

L'importanza della modellazione matematica nello studio di sistemi biochimici può, a volte, risultare non molto evidente. Quando osserviamo un fenomeno biologico siamo messi a confronto con un insieme di processi complessi che non sempre possono essere spiegati a priori e il cui risultato non può essere previsto con l'intuizione. Anche quando i principi generali sono ben noti, la biochimica delle singole molecole coinvolte, o addirittura dell'intero sistema, è talvolta sconosciuta e può variare di molto tra specie differenti. Gli esperimenti conducono alla formulazione di ipotesi relative a processi individuali ma rimane spesso poco chiaro se queste ipotesi possano essere trasportate su larga scala, perché risulta difficile prevedere il comportamento globale di un sistema complesso solo dalla conoscenza delle sue componenti. In questo contesto ci possono venire in soccorso la modellazione matematica e le simulazioni virtuali che aiutano a comprendere la natura interna e le dinamiche di questi processi, permettendo di formulare predizioni sul loro sviluppo futuro e sugli effetti delle interazioni con l'ambiente circostante.

I modelli acquisiscono i loro riferimenti alla realtà attraverso la comparazione con i risultati sperimentali e dipendono dunque dalla qualità di questi ultimi ma non per questo risultano meno importanti. La modellazione combinata con la sperimentazione porta a maggiori vantaggi se comparata alla semplice sperimentazione diretta. In particolare la modellazione mette in risalto le lacune nella conoscenza o nella comprensione di un sistema, poichè durante la formulazione di un modello tutte le componenti o le interazioni non specificate devono essere determinate. Un modello adeguato può anche assistere la fase di sperimentazione, permettendo di testare scenari non accessibili dagli esperimenti classici, come per esempio quello in cui vengono introdotte perturbazioni mirate generalmente non applicabili a sistemi reali. Inoltre, una volta consolidato, il modello ottiene una certa indipendenza dall'iniziale oggetto di studio. Per questo motivo algoritmi risolutivi e programmi informatici possono essere usati sul modello indipendentemente dal sistema in esame.

Le simulazioni dei modelli biologici possono essere ripetute più volte sotto diverse condizioni e i prodotti essere rappresentati in termini matematici formali, consentendone così la generalizzazione. I dati generati da una simulazione risultano facilmente esprimibili in forma di grafici, tabelle o immagini 3D, favorendo così la visualizzazione del sistema studiato e facilitandone la comprensione. Per concludere, i modelli permettono di effettuare previsioni ben fondate e verificabili, costituiscono un archivio della conoscenza finora ottenuta, sia stabilita che ipotetica e allo stesso tempo forniscono ai ricercatori descrizioni qualitative di queste conoscenze, permettendo loro di effettuare simulazioni dei relativi processi biologici. Per una trattazione più completa sull'importanza della modellazione nel campo della ricerca biologica si consiglia di consultare il paper "*Systems Biology a Textbookt*" [4] in bibliografia.

1.2 Motivazioni

L'intento del programma è quello di creare, in primo luogo, un sistema di traduzione diretta a partire da un modello SBML in uno script LAMMPS. Il primo, *System Biology Markup Language*, rappresenta uno standard per la scrittura di modelli biochimici in formato *xml*. L'utilizzo di questo standard garantisce una certa portabilità del modello che, così definito, può essere letto ed interpretato da diversi software di simulazione. Tra questi non è però presente LAMMPS, software per simulazioni di dinamica molecolare, che ad oggi non offre nessuna possibilità di acquisizione immediata di file SBML.

In secondo luogo risulta interessante l'idea di sfruttare un simulatore potente come LAMMPS, ottimizzato per i compiti di Molecular Dynamics, utilizzandolo fuori dal suo contesto abituale. Gli strumenti offerti da LAMMPS, in particolare la sua capacità di simulare molto efficacemente aspetti di cinematica e termodinamica, si sono rivelati altrettanto utili anche quando utilizzati per definire simulazioni più astratte sui sistemi di larga scala. La possibilità, per esempio, di simulare agevolmente il moto browniano su un insieme di particelle è risultato uno strumento valido al servizio di simulazioni nelle quali le particelle sono gli agenti e il moto browniano ad esse applicato è ciò che ne determina la *'libertà'* di movimento.

1.3 Contributi

Poiché i file che seguono lo standard SBML sono molto diffusi ed essendo il formato *xml* particolarmente pratico da esaminare per qualsiasi linguaggio di programmazione, è stato sviluppato uno script Python, *create.py* (1.5), in grado di compiere questo passaggio di traduzione e ricoprire il ruolo di intermediario fra SBML e LAMMPS. La versatilià di Python ha reso agevole sia l'implementazione del processo di lettura del modello che quello di scrittura degli opportuni comandi nello script *in.lmp*, associando così direttamente ad ogni modello studiato il codice eseguibile che lo rappresenta. Questo processo, sopra descritto, ha portato alla costruzione di un vero e proprio simulatore di modelli SBML attraverso LAMMPS.

Vista l'immediatezza nelle fasi di traduzione e scrittura c'è da sottolineare come le parti del progetto che abbiano richiesto più lavoro siano state quelle di definizione del metodo di simulazione, illustrato nel capitolo 3, e di verifica delle attuali possibilità pratiche di adottare tale metodo su LAMMPS.

1.4 Progetti Correlati

Allo stato attuale non sono disponibili altri progetti software che mettono in relazione diretta i modelli SBML con il simulatore LAMMPS che però è massicciamente impiegato nello studio delle interazioni molecolari e non solo. Non è stato dunque difficile trovare esempi di applicazione in contesti sostanzialmente diversi da questo ma che si sono rivelati allo stesso modo utili nello sviluppo di determinate strategie implementative. In particolare si è fatto riferimento al paper "*Using the SRSim Software for Spatial and Rule-Based Modeling of Combinatorially Complex Biochemical Reaction Systems*" ([2]) utilizzato per determinare come strutturare genericamente una simulazione, definirne l'ambiente, inizializzarne gli elementi e impostarne le forze.

1.5 Struttura del Progetto

Di seguito è delineata la struttura del progetto[1] e l'organizzazione in cartelle e sottocartelle del codice sorgente e dei file di output. Alcuni di questi file saranno trattati in maniera più approfondita nel capitolo 4 Implementazione.

Struttura

- LATEX : contiene i template, le immagini, i grafici, i frammenti di codice inclusi nella relazione del progetto scritta in latex ed anche il file *main.tex* per generare il pdf in latex;
- MODELS : contiene i modelli di BioModel e quelli di testing organizzati in sottocartelle relative ad ogni esperimento;
- PRJ : contiene le cartelle, le librerie e gli script necessari all'esecuzione del software:
  - LIB : contiene gli script che definiscono ed implementano le funzioni per la manipolazione dei modelli;
    - *create.py* : centrale per il funzionamento del sistema, definisce le funzioni che si occupano della lettura di un modello SBML e la successiva traduzione in codice eseguibile LAMMPS;
  - OUTPUT : contiene le cartelle con i file di output prodotti dal programma per ogni esperimento[2];
    - NOME.ESPERIMENTO/*dump.nome.esperimento.out* : contiene un riepilogo della simulazione del relativo esperimento;
    - NOME.ESPERIMENTO/*sbml.analysis* : contiene la tabella delle specie ed il riepilogo delle reazioni presenti nel modello del relativo esperimento;
  - SIMULATION : contiene le cartelle con gli script per l'esecuzione della simulazione di ogni esperimento[2];
    - NOME.ESPERIMENTO/*in.lmp* : script di comandi LAMMPS per l'esecuzione di una simulazione;
  - *main.py* : gestisce un'interfaccia tra l'utente e lo script *create.py* da cui importa le funzioni che permettono di:
    1. selezionare un modello SBML dal filesystem;
    2. generare il rispettivo script LAMMPS;
    3. eseguire lo script avviando la simulazione;
    4. organizzare in cartelle i dati di output;
  - *run.sh* : si occupa di eseguire il codice LAMMPS parallelamente su più processori con i corretti parametri in input;
- RESOURCES : contiene tutte le principali risorse, paper e documentazioni utilizzate (i file esterni contenuti di questa cartella sono riportati nella bibliografia).

[1] Per una miglior lettura le cartelle saranno indicate in stampato maiuscolo mentre i file in corsivo.

[2] le cartelle saranno denominate con il nome dell'esperimento (NOME.ESPERIMENTO) analogo al nome del relativo modello.

Capitolo 2

Prerequisiti

Le conoscenze di base, necessarie per una comprensione generale dell'aspetto tecnico del progetto, si limitano a nozioni di programmazione imperativa e conoscenza delle tecniche per la valutazione della complessità algoritmica e del costo computazionale. Gli algoritmi impiegati nello script *create.py* sono quelli relativi all'organizzazione dei dati in strutture quali matrici o dizionari, alla definizione di procedure classiche di accesso a queste strutture, a fine di integrare le informazioni del modello nel codice generato, e alla scrittura su un file esterno, *in.lmp*, di comandi nella sintassi LAMMPS. Relativamente a quest'ultimo, per poter comprendere al meglio i meccanismi di funzionamento dei singoli comandi e più in generale quelli che regolano lo svolgimento delle simulazioni, è necessario possedere una conoscenza non banale del simulatore che può essere acquisita grazie al materiale fornito dal sito ufficiale di LAMMPS [7]. In bibliografia, inoltre, sono riportati i link alle pagine della documentazione relative a comandi specifici che ricoprono un ruolo importante per il funzionamento del software. Il funzionamento di questi comandi, con gli opportuni riferimenti bibliografici, sarà approfondito nel capitolo 4. In alternativa, per avere un'idea generale degli aspetti chiave dello strumento software impiegato si può fare riferimento al paper introduttivo "*LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales*" [1] messo a disposizione dallo stesso sito.

L'aspetto più formale del progetto, quello che concerne la descrizione degli approcci adottati nell'interpretazione dei modelli in analisi e l'illustrazione dei metodi realizzativi applicati, richiede una conoscenza fondamentale delle tecniche di modellazione ed in generale di alcuni concetti di biochimica molecolare. Nonostante i principi base di queste tecniche siano illustrati nel capitolo 3, al fine di fornire una più chiara giustificazione delle scelte compiute nel corso della progettazione si può far riferimento al paper "*Systems Biology a Textbookt*" [4], citato nello stesso capitolo.
In caso non si possedessero queste conoscenze non risulterebbe tuttavia impossibile comprendere la logica, piuttosto intuitiva, posta alla base del funzionamento del sistema. L'elevato livello di astrazione dal realismo dei modelli impiegati contribuisce infatti a semplificare i meccanismi principali della costruzione e della valutazione delle simulazioni. Resta comunque di rilevante importanza comprendere a pieno la filosofia di modellazione *Agent Based*, centrale in tutte le fasi di sviluppo del sistema. Questo concetto è quello trattato in maniera più approfondita nel capitolo seguente.

Capitolo 3

Metodi

In questo capitolo verranno delineati al meglio i metodi formali impiegati nella fase di progettazione, descrivendone l'applicazione nel contesto del lavoro svolto e degli intenti perseguiti. Per fare ciò è necessario premettere che, trattandosi dello sviluppo di un simulatore per modelli biologici, molte delle scelte effettuate sono state influenzate dell'approccio di modellazione adottato. Nello specifico, in questo campo sono tre gli approcci principali per la modellazione di sistemi biochimici:

1. *Network-based models*;
2. *Rule-based models (Agent Based)*;
3. *Statistic models*;

I *Network-based models* descrivono i sistemi interessati, analizzandone le proprietà, gli stati e le dinamiche, ossia le componenti del sistema e le loro interazioni in rete. Contesti tipici in cui i Network-based models risultano centrali sono: i sistemi di equazioni differenziali ordinarie per la modellazione di reazioni biochimiche, i modelli booleani per le reti le regolatorie geniche o per le reti metaboliche e la descrizione stocastica di reazioni biochimiche e altri processi di cambiamento di stato.

I *Rule-based models* o *Agent based models* rappresentano un approccio differente per la modellazione dei fenomeni biologici. Secondo questo approccio ogni componente del sistema può modificare il suo stato in accordo con una serie di regole predefinite. La modellazione basata su di un sistema di regole esplicita tutti i possibili cambiamenti di stato delle singole componenti ma non tutti i possibili stati in cui esse si potranno trovare, per questo motivo risulta meno avida, da un punto di vista computazionale, rispetto a quella basata su sistemi di equazioni differenziali ordinarie. Più complessi sono i modelli *Agent Based* dove tutte le componenti, come ad esempio proteine o cellule, sono considerate come agenti autonomi obbligati a seguire le loro personali regole. In questo caso gli agenti potranno muoversi liberamente all'interno dello spazio di contenimento, in accordo con le meccaniche dell'ambiente che lo stesso rappresenta. Questo approccio può essere utilizzato per descrivere l'interazione tra diverse cellule del sistema immunitario così come quella tra un parassita con il tessuto ospite occupato.

Infine, alla luce della massiccia produzione di dati da tecnologie omiche, i modelli statistici risultano molto importanti nel campo della systems biology. Gli *Statistic models* stabiliscono relazioni tra i dati misurati e forniscono una guida per l'estrazione delle strutture nascoste dei sistemi biologici che hanno prodotto tali dati.

Come si evince dalla loro descrizione questi approcci vengono impiegati per far fronte a necessità di modellazione molto differenti tra loro ma questo non vuol dire che siano mutuamente esclusivi. Non di rado, infatti, si tenta di combinarli al fine di ottenere il meglio da ognuno. Per una trattazione più approfondita riguardo alle tecniche di modellazione matematica di sistemi biologici complessi consultare "*Systems Biology a Textbookt*" [4].

Nel progetto è stato adottato un approccio *Agent Based*, interpretando le specie presenti nel modello SBML come agenti e le reazioni, descritte dallo stesso modello, come le regole che questi agenti sono obbligati a seguire. L'attribuzione del ruolo di agenti indipendenti alle specie ha consentito di tralasciare la descrizione di alcuni aspetti, come le leggi dinamiche a cui sono classicamente sottoposte, e di trattare le stesse come delle generiche *'sfere'* piuttosto che come molecole o cellule. Questo ha comportato un certo livello di distacco dal realismo del sistema studiato, permettendo allo stesso tempo di concentrarsi sulla modellazione delle regole, o reazioni, imposte dal modello e fondamentali per la definizione dell'interazione tra gli agenti.

In sostanza quello che accade durante una simulazione è che un certo numero di specie, inserite all'interno di un unico ambiente in modo casuale, muovendosi disordinatamente imitando il moto browniano delle particelle entrano in contatto tra di loro scontrandosi le une con le altre. Questi scontri rappresentano il fulcro delle interazioni nel sistema. A seguito di uno di essi è infatti possibile, con un livello di probabilità dato, che si generi un legame tra gli agenti interessati. I termini per la formazione di un legame sono definiti dalle regole del modello in analisi. Se il file SBML stabilisce che può esistere una reazione tra una specie di tipo a ed una di tipo b allora sarà possibile formare un legame tra gli agenti di queste specie.

Per quanto riguarda la modellazione del processo di reazione vero e proprio non ci si è però limitati a considerare le interazioni ed i legami tra gli agenti, questo aspetto definisce infatti il principio di reazione ma non ne illustra il compimento. Prima di specificare il metodo con cui è stato affrontato il processo di reazione tra specie è doveroso premettere che le reazioni sono state qui concepite come irreversibili e con eguale velocità di esecuzione. La cinetica delle reazioni chimiche, riportata quasi sempre nei file SBML, consiste in uno di quegli aspetti tralasciati al fine di favorire il processo di traduzione del modello in codice LAMMPS.
Per stabilire il termine di una reazione si procede con l'eliminazione dei reagenti in essa coinvolti, ossia quegli agenti che hanno stretto fra loro legami, e all'introduzione nel sistema di una serie di nuovi agenti appartenenti a specie equivalenti a quelle dei prodotti della reazione. Possono anche esistere reazioni senza prodotti che descrivono il decadimento delle specie interessate e che si risolvono nella semplice eliminazione degli agenti coinvolti.

Capitolo 4

Implementazione

In questo capitolo sono illustrate alcune delle tecniche implementative impiegate nello sviluppo della componente software, in accordo con l'approccio descritto nel capitolo precedente. Adottare un metodo *Agent Based* per l'interpretazione di modelli biologici ha portato ad effettuare scelte ben precise, non solo rispetto alla decisione dei metodi progettuali da applicare ma anche rispetto al processo di generazione del codice sorgente delle diverse simulazioni LAMMPS. Questo processo è caratterizzato da due aspetti principali: un aspetto di lettura ed interpretazione ed un aspetto di traduzione e scrittura.

L'aspetto della lettura fa riferimento al momento dell'acquisizione da parte dello script *create.py* di un modello SBML di livello 3[1]. Questo viene analizzato grazie al supporto delle funzioni fornite dalla libreria Python *libSBML* [22]; dall'analisi si ricavano informazioni utili sull'ambiente di simulazione, sulle specie e sulle reazioni. Ognuno di questi aspetti è affidato ad un'apposita classe Python che si occupa di definire ed eseguire gli algoritmi necessari all'estrapolazione e organizzazione dei dati richiesti i quali saranno poi accessibili sotto forma di attributi pubblici della classe stessa. Le classi in questione sono:

- *ComparmentClass*;
- *SpeciesClass*;
- *ReactionClass*;

Un oggetto di ognuna di queste classi è istanziato nelle prime righe della funzione *make_lmp*, per essere poi utilizzato nella generazione del codice della simulazione.

La *ComparmentClass* raccoglie le specifiche dell'ambiente di simulazione e si occupa della ricerca dei compartimenti definiti nel modello, di cui memorizza la dimensione. Questo passaggio è fondamentale rispetto all'interpretazione del concetto di compartimento che si è deciso di adottare. Ogni specie è associata ad un solo compartimento; la dimensione di questo influenzerà la quantità di elementi di tale specie al tempo 0 espressa come *Concentrazione Iniziale*. La funzione che lega la grandezza del compartimento di appartenenza di una specie con il numero di agenti di quella specie, da creare prima dell'avvio della simulazione, è definita

[1] Anche file SBML L3V4 sono accettati

dalla formula espressa alla riga 10 dell'algoritmo 1. Lo stesso algoritmo si occupa inoltre di verificare che nel modello tutte le quantità iniziali siano espresse o in termini di concentrazione (condizione assai più comune) o di *Amount* (quantità diretta). In questo secondo caso infatti non ci sarebbe bisogno di calcolare il numero relativo di agenti che rimarrebbe indipendente dalle dimensioni del compartimento. Sebbene l'utilizzo di *Amount* sia meno comune è giusto considerare comunque questa possibilità ed escludere l'utilizzo contemporaneo dei due metodi che risultano infatti mutualmente esclusivi. Nel caso una specie venga definita in termini di *Concentrazione Iniziale* e un'altra in termini di *Amount* l'algoritmo si blocca e restituisce un errore.

Algorithm 1 Calcolo della Concentrazione Iniziale

1: **for** i in NUM_OF_SPECIES **do**
2: $specie = model.getSpecies(i)$
3: $value = specie.getInitialConcentration()$
4: $size = getCompartmentSizeOf(specie)]$
5:
6: **if** $value = NaN$ **then**
7: $scale \leftarrow false$
8: $break$
9: **else**
10: $Initial\_Numbers \leftarrow \lceil value \times 10^{-exp(value)} \times size \rceil$
11: **end if**
12: **end for**
13:
14: **if** $scale = false$ **then**
15:
16: **for** i in NUM_OF_SPECIES **do**
17: $specie = model.getSpecies(i)$
18: $value = specie.getInitialAmount()$
19:
20: **if** $value = NaN$ **then**
21: $return$ "$Error: \ no\ initial\ amount/concentration\ given$"
22: **else**
23: $Initial\_Numbers \leftarrow value$
24: **end if**
25: **end for**
26: **end if**
27:
28: $return\ Initial\_Numbers$

La *SpeciesClass* si occupa della ricerca delle specie definite nel modello e ad ogni specie identificata associa una serie di valori utili per la costruzione della simulazione in LAMMPS. Questi valori vengono organizzati in un dizionario ed è possibile accedervi direttamente utilizzando l'*id* della specie di interesse, una volta creata un'istanza dell'oggetto *SpeciesClass*. I dati contenuti nel dizionario delle specie sono:

1. Atom Id : un numero intero progressivo che rappresenta l'*id* dell'agente associato alla specie all'interno della simulazione;
2. Specie Compartment : identifica in modo univoco il compartimento a cui la specie appartiene;
3. Initial Number : è il risultato dell'algoritmo 1 illustrato in precedenza e rappresenta il numero di agenti della relativa specie da inserire all'inizio della simulazione;
4. Growth Rate : indica il tasso di crescita di una specie in termini di timestep. In questa sede è definito come un semplice numero casuale ma il valore può essere sostituito da uno sperimentalmente valido;
5. Deposit Counter : consiste in un semplice contatore utilizzato per distinguere i comandi LAMMPS per la nascita di un nuovo agente della relativa specie nel caso in cui questa sia coinvolta in più reazioni come prodotto;

L'Algoritmo 2 definisce la struttura del dizionario delle specie.

Algorithm 2 Creazione del Dizionario delle Specie

```
for i in NUM_OF_SPECIES do
    specie = model.getSpecies(i)
    Initial_Number = Initial_Atoms[i]
    Growth_Rate = random(100, 500)
    Deposit_Counter = 0

    Dictionary_of_Species[specie] = [

                Atom_Id,
                specie.getCompartment(),
                Initial_Number,
                Growth_Rate,
                Deposit_Counter
            ]

    Atom_Id += 1

end for

return Dictionary_of_Species
```

La *ReactionClass* si occupa della ricerca delle reazioni definite nel modello di cui interessa conoscere la lista dei reagenti e dei prodotti. Anche in questo caso i dati vengono organizzati sotto forma di dizionario dove come chiave si usa l'*id* della reazione e come valore vengono poste le due liste nominate. Un aspetto rilevante è che tra i compiti di questa classe ricade quello di calcolare tutte le possibili combinazioni dei reagenti per ogni singola reazione. Questo si mostra necessario dal momento in cui LAMMPS permette di definire legami esclusivamente tra coppie di agenti. Come anticipato nel capitolo precedente, i legami tra elementi di una specie vengono interpretati come interazioni tra agenti e nel caso in cui una di queste coinvolga più di due agenti risulta fondamentale poterli mettere tutti in relazione fra di loro. Questo si concretizza nella definizione di legami a due a due tra gli agenti coinvolti, per farlo è necessario stabilire tutte le possibili combinazioni di 2 elementi su un insieme di dimensione n. Poiché questa operazione va eseguita per tutte le razioni possiamo dedurre la complessità dell'algoritmo 3 Calcolo delle Combinazioni tra Reagenti, come $\mathcal{O}(m\frac{n!}{k!(n-k)!})$, dove m è il numero di reazioni totali, n è il numero massimo di reagenti individuati in una reazione e $k = 2$.

Algorithm 3 Calcolo delle Combinazioni tra Reagenti

```
for r_id in Reactions.keys() do
    for i in range(0, rri) do
        for j in range(i+1, rri) do
            combinations ← (reactions[r_id][1][i], reactions[r_id][1][j])
        end for
    end for
end for

return combinations
```

Il secondo compito affidato allo script *create.py* è, come già anticipato, quello di traduzione e scrittura. Per traduzione si intende quel processo di manipolazione dei dati *'puri'* estratti dal modello SBML al fine di trasformarli in comandi LAMMPS, scritti poi nel file sorgente *in.lmp* (1.5).
LAMMPS (*Large-scale Atomic/Molecular Massively Parallel Simulator*) è il software impiegato per la gestione delle simulazioni. Si tratta di un simulatore di dinamica molecolare programmabile che, una volta installato, consente di interpretare ed eseguire script personalizzati. Questi script devono contenere una sequenza ordinata di comandi, scelti tra quelli messi a disposizione da LAMMPS, attraverso cui è possibile definire le regole di svolgimento di una simulazione. Alla luce di questo, si precisa che all'interno di una simulazione si considerano agenti quelli che LAMMPS definisce con la parola chiave *atoms*. Tutti i comandi e le funzioni di LAMMPS sono implementati in C++ (per maggiori informazioni consultare il sito riportato in bibliografia [6]). Nel caso si volessero sperimentare le potenzialità di LAMMPS per valutarne un possibile utilizzo futuro, si consiglia di seguire la guida ufficiale [8] per l'installazione ed il tutorial Environment Setup Build LAMMPS [10] per il set-up.
Di questa operazione di traduzione si vogliono descrivere alcuni aspetti interessanti come il meccanismo di definizione dei legami tra le coppie di agenti stabilite dall'Algoritmo 3.

La sintassi di questo comando in particolare :

"fix ID group-ID bond/create Nevery itype jtype Rmin bondtype keyword values";

indica, per esempio, che si tratta di un vincolo (fix), che viene applicato ogni N timestep, che coinvolge atomi di tipo i e di tipo j, (il tipo dipende dall' Atom_Id (2) e che definisce un legame tra questi atomi in base a delle caratteristiche specifiche [14]. L'algoritmo 4 scrive, nel file *in.lmp*, una serie di questi comandi, in relazione al numero di coppie nell'array *combinations*, imponendo inoltre che per ogni agente sia possibile formare solo un legame di questo tipo.

Algorithm 4 Definizione di Legami tra Coppie di Agenti

```
1: for type in combinations do
2:     type1 ← Dictionary_of_Species[type[1]][1]
3:     type2 ← Dictionary_of_Species[type[2]][1]
4:
5:     write → fix bond_type1_type2 all bond/create 10 "+
6:           "type1 type2 1.0 1 prob 0.5 seed ")
7:
8: end for
```

Un altro aspetto cruciale nella fase di traduzione è la definizione di un meccanismo per valutare le reazioni effettivamente avvenute con il progredire della simulazione. Per fare ciò, si è deciso di adottare il metodo del conteggio del numero di legami formati da un agente con altri agenti coinvolti nella stessa reazione. Per esempio, se un agente di tipo a è coinvolto in una reazione con un agente di tipo b e durante la simulazione si rileva un nuovo legame tra i due, allora sarà necessario sostituire la coppia con un nuovo agente (o una serie di nuovi agenti) prodotto della reazione. Poiché in LAMMPS è possibile contare solamente il numero di legami relativi ad un agente (atomo) si è deciso di calcolare questo valore solo per gli agenti di un tipo x[2] tra quelli coinvolti nella reazione e verificare periodicamente se fosse uguale al numero di reagenti della stessa reazione meno uno. Questo, infatti, indicherebbe che l'agente in questione ha stretto un legame con tutti gli altri agenti meno se stesso, completando la reazione. Il controllo di questa condizione avviene ricalcolando ciclicamente il numero di legami e modificando il valore di una serie di variabili *bond_r_id* booleane, associate alle diverse reazioni (riga 23 algoritmo5): r_id infatti varia nel ciclo for indicando l'*id* della reazione che si sta analizzando. Queste variabili sono tutte di tipo *atoms*, in sostanza degli array che contengono tanti elementi quanti sono gli agenti correntemente presenti nella simulazione ed assumono valore *true* solo se l'agente è del tipo prescelto e se il numero di legami formati da quell'agente corrisponde al numero desiderato. Un'altra variabile è stata impiegata per determinare quali agenti eliminare una volta terminata la reazione; si tratta della variabile *toDelete* che stabilisce per ogni agente se questo è pronto ad essere eliminato. Il principio di questa variabile è analogo a quello di *bond_r_id* e l'unica differenza è che questa può assumere valore *true* per atomi di tutti i tipi, non

[2]Nello specifico per ogni reazione viene scelto il tipo di agente che compare per primo nella definizione

solo di un tipo x, a patto che questi abbiano raggiunto il loro numero massimo di legami (consultare [17] per maggiori informazioni sul funzionamento delle variabili in LAMMPS). L'applicazione di questo meccanismo di valutazione delle reazioni mette in risalto un punto debole del sistema, ossia la possibile ricorrenza in errore nel caso in cui un agente sia coinvolto in due reazioni contemporaneamente e venga scelto come rappresentante di entrambe: il numero di legami potrebbe essere valido se si sta considerando una reazione ma errato per un'altra. Queste informazioni vengono calcolate tramite l'utilizzo dei comandi *compute*. Tutte le variabili e i *compute* definiti a questo punto della simulazione necessitano di un continuo aggiornamento effettuato dal comando *run* 0. Da questo algoritmo vengono escluse quelle reazioni che coinvolgono meno di due reagenti in quanto non ci sarebbero interazioni da valutare.

Algorithm 5 Valutazione dei Legami nel Corso della Simulazione

```
# compute type for each atom
write → "compute atype all property/atom type"

# compute number of bonds for each atoms
write → "compute nbond all property/atom nbonds"

for r_id in Reactions.keys() do
    if num_of_rectants[r_id] ≤ 1 then
        pass

    else
        num_of_bonds = (num_of_rectants[r_id] − 1)

        for type in rectants[r_id] do
            type ← Dictionary_of_Species[type][1]
            toDelete ← "(c_atype = type && c_nbond ≥ num_of_bonds) ..."
        end for

        # count type of bond present in the
        # simulation at the current timestep
        f_type ← first atom_id in reaction r_id

        write → "variable bond_of_(r_id) atoms c_atype = f_type &&
                 c_nbond = num_of_bonds"

        write → "compute count_type all reduce sum v_bond_of_(r_id)"

        write → "variable new_atom_type equal c_count_type "

        New_Atoms ← v_new_atom_type

    end if
end for
write → toDelete
```

Nel caso in cui vi siano delle reazioni senza reagenti ma con uno o più prodotti, questo è da interpretare come un caso di *"nascita"* di una nuova specie. Una situazione di questo tipo necessita di essere gestita separatamente rispetto alle classiche reazioni tra agenti. Per questo motivo è stato utilizzato il comando:

"fix ID group-ID deposit M type N seed keyword values";

che definisce un vincolo, valido solo una volta durante la simulazione, che impone la creazione (o il deposito) di un nuovo atomo ogni N timestep fino a che il numero di agenti depositati non raggiunge M. Ponendo come M il numero totale di timestep della simulazione è possibile prolungare gli effetti del comando rendendolo di fatto sempre attivo. Inoltre ponendo $N = Growth\_Rate$ relativo all'agente in questione è possibile simulare una frequenza di *"nascite"* realistica. L'algoritmo 6 rappresenta l'applicazione di questo comando a tutte le specie interessate. Da questo algoritmo vengono escluse tutte quelle reazioni che non hanno prodotti e che rappresentano il decadimento di un agente, un processo il cui funzionamento sarà illustrato in seguito.

Algorithm 6 Prodotti Spontanei

```
for r_id in Reactions.keys() do
    if reaction(r_id) non ha prodotti) then
        pass

    else
        for type in products[r_id] do
            type ← Dictionary_of_Species[type][1]
            growth ← Dictionary_of_Species[type][4]

            if reaction(r_id) non ha reagenti then
                write → "fix deposit_type all deposit duration type"
                        "growth seed region box near 2.0"
            else
                pass

            end if
        end for
    end if
end for
```

Le reazioni di decadimento sono simulate eliminando periodicamente tutti gli agenti assegnati al gruppo dinamico *perishable*. Un gruppo dinamico in LAMMPS rappresenta una collezione di agenti il cui numero di elementi può variare nel corso della simulazione. I criteri con cui gli agenti vengono inseriti in un gruppo dinamico sono vari, per stabilirli si fa affidamento a variabili booleane di tipo *atoms*. Nel caso in cui un agente, per esempio, avesse valore *true* per la variabile *toDelete* verrebbe aggiunto al gruppo *garbage* grazie al comando:

"group garbage dynamic all every 10 var toDelate "

Questo meccanismo viene impiegato per eliminare gli agenti che hanno terminato una reazione. Similmente gli agenti coinvolti in reazioni senza prodotti e il cui livello di energia cinetica è calato sotto una certa soglia minima vengono assegnati al gruppo *perishable* e successivamente rimossi. Per un chiarimento sul funzionamento dei gruppi in LAMMPS fare riferimento a [18].

In conclusione sarà qui illustrato il meccanismo di creazione di nuovi agenti in base al risultato delle computazioni descritte precedentemente. Il comando *crete_atoms*, utilizzato anche all'inizio della simulazione per inserire nell'ambiente di riferimento gli agenti di partenza, viene in questo caso invocato ogni 100 timestep[3] e si occupa di posizionare nello spazio di simulazione l'adeguato numero di agenti prodotti dalle diverse reazioni. L'invocazione periodica di questo ed altri comandi avviene tramite un meccanismo di loop interno a LAMMPS, ottimizzato per lo scopo, la cui frequenza può essere modificata direttamente nel comando di *run* attraverso la keyword *every*. La durata totale della simulazione viene stabilita dall'utente al momento dell'avvio ed è rappresentata nel codice dalla variabile *steps* in termini di timestep. Le quantità di nuovi agenti da generare saranno ovviamente diverse ad ogni invocazione del comando *crete_atoms* poiché dipendono dalla specie a cui l'agente appartiene e variano in base al risultato delle computazioni dell'algoritmo 5. Per questo motivo saranno rappresentate da una serie di variabili, costantemente aggiornate, i cui nomi per il riferimento sono contenuti nell'array New_Atoms costruito anch'esso dall'algoritmo 5. Infine, per quanto riguarda le diverse posizioni nelle quali questi agenti vengono posti, è stato deciso di renderle casuali utilizzando *seed* per la generazione di numeri random sempre differenti (vedi la variabile *position*) al fine di evitare sovrapposizioni.
L'algoritmo 7 viene eseguito solo se nel modello vi sono reazioni che coinvolgono dei reagenti. In questo caso pero', a differenza del 5, anche se il modello non presenta reazioni complete, il codice deve essere eseguito poiché contiene il comando *run* necessario all'avvio della simulazione. Ulteriore scopo dell'algoritmo è anche quello di eliminare, periodicamente, tutti gli agenti assegnati ai gruppi dinamici di scarto ossia quelli che hanno completato una reazione o che hanno compiuto il loro ciclo di vita.

[3] Questo numero potrebbe variare per modelli ricchi di specie diverse dove risulta conveniente aggiornare più frequentemente lo stato del sistema

Algorithm 7 Avvio della Simulazione e Creazione di Nuovi Agenti

```
increment = 0

if len(combinations) > 0 then
    write → " run steps every 100"
    write → "   variable position equal ⌈(random(1, 1000000000, 10)/100000.0)⌉ "

    for r_id in Reactions.keys() do

        if reaction(r_id) non ha prodotti) then
            pass

        else
            for type in products[r_id] do
                type ← Dictionary_of_Species[type][1]
                growth ← Dictionary_of_Species[type][4]

                if reaction(r_id) non ha reagenti then
                    pass

                else
                    write → "create_atoms type random "
                            "New_Atoms[increment] position+increment box"

                    increment+ = 0

                end if
            end for
        end if
    end for

else
    write → " run steps"

end if
```

Capitolo 5

Risultati Sperimentali

Verrà qui riportata una breve descrizione introduttiva riguardo alle prove effettuate ed ai risultati ottenuti in questa fase, seguita da un'analisi approfondita dei punti rilevanti per una corretta interpretazione degli esiti sperimentali. Gli esperimenti compiuti si sono mostrati necessari e sufficienti per un esame completo del comportamento del sistema. La molteplicità di test attuati ha infatti permesso una copertura esaustiva di tutti i casi di studio illustrati nel paragrafo 5.3. I prodotti di ogni test sono stati poi analizzati ed interpretati in accordo con gli obbiettivi perseguiti, esposti in seguito nel paragrafo 5.1. Informazioni attinenti al processo di generazione dei dati ed ai modelli impiegati nelle prove sono invece riportate al paragrafo 5.2 relativo alla configurazione del sistema nello stadio sperimentale.
Una stima del costo computazionale complessivo considera tutti i passaggi richiesti dal software per la valutazione di un modello SBML, ossia l'interpretazione di questo sulla base della quale costruire una simulazione e l'esecuzione della simulazione stessa. Tale stima è riportata nel paragrafo 5.5 con i dovuti approfondimenti. Conviene infine ricordare che, non rientrando nelle competenze specifiche dello sviluppatore del sistema la costruzione e l'analisi dei modelli biochimici, l'interpretazione dei dati sperimentali è stata concepita in funzione di dimostrare la correttezza del sistema più che per trarre conclusioni rilevanti nel campo di applicazione dello stesso.

5.1 Obiettivi

Lo svolgimento di test mirati sul funzionamento del sistema e l'esecuzione di esperimenti, legati a modelli SBML predefiniti, sono stati eseguiti con il fine di conseguire i seguenti obiettivi:

- verificare la correttezza del codice nella fase di traduzione e simulazione di un modello SBML;
- ottimizzare il codice prodotto al fine di rendere le singole simulazioni significativamente più efficienti;
- ottenere una stima del costo computazionale medio atteso per l'esecuzione di un esperimento;
- effettuare previsioni sull'evoluzione di un sistema biologico osservando lo sviluppo della simulazione di un modello;

La verifica della correttezza, come anticipato nell'introduzione a questo capitolo, rappresenta l'obiettivo primario della fase sperimentale. Una definizione più accurata di ciò che si intende per correttezza sarà data in seguito nel paragrafo 5.4. Solo portando avanti numerosi esperimenti, eseguiti impiegando modelli significativamente diversi tra loro, è stato possibile possibile coprire tutti i casi di studio per ottenere riscontri validi sulla correttezza generale e portare a compimento questo obiettivo. In particolare i modelli utilizzati in questa occasione possono essere considerati come modelli scritti ad hoc con lo scopo di testare il sistema e provarne la validità. La struttura e l'organizzazione dei citati modelli verrà discussa nel paragrafo 5.2 sulla configurazione.

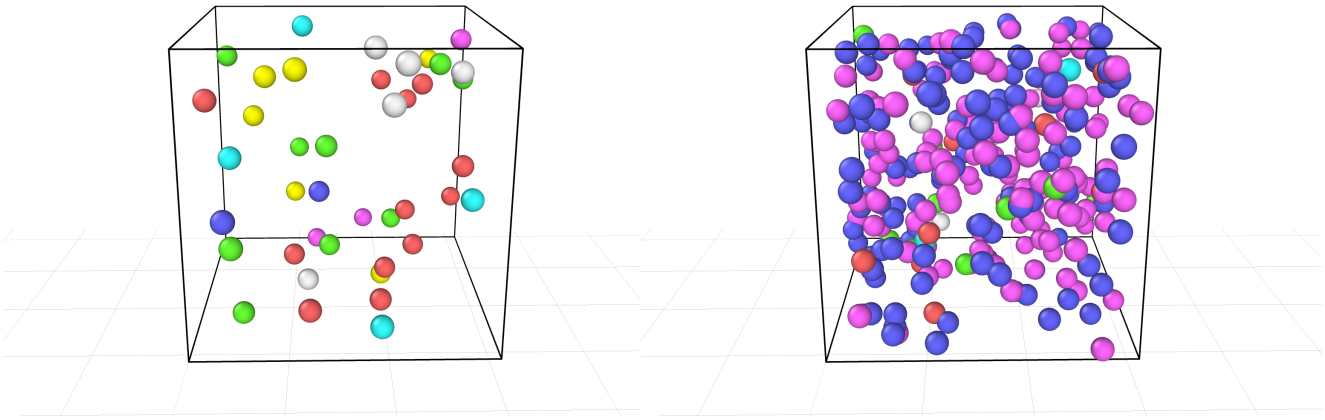
L'ottimizzazione del codice può essere considerata secondaria rispetto alla dimostrazione della correttezza, pertanto subordinata a questa. Ottimizzare un codice non verificato e dunque possibilmente incorretto risulterebbe un operazione inutile e controproducente, per tale motivo questo obiettivo è stato perseguito solo in seguito al compimento del precedente. Lo scopo finale dell'ottimizzazione è quello di manipolare il codice per renderlo più compatto, leggibile e funzionale, con l'eventuale vantaggio di velocizzare l'esecuzione delle singole simulazioni. Un tale risultato, che implica la possibilità di garantire buone prestazioni del sistema nell'atto di eseguire simulazioni lunghe e complesse, è stato ottenuto grazie all'impiego di test mirati. I suddetti test hanno comportato l'utilizzo di file SBML personalizzati, simili a quelli utilizzati in precedenza per la verifica della correttezza, la cui struttura sarà anche in questo caso esplicitata nel paragrafo 5.2 sulla configurazione.

La stima dell'efficienza del sistema risulta ancora una volta come un obiettivo subordinato, in questo caso successivo anche a quello dell'ottimizzazione. Parlando di efficienza ritorna il tema sulla separazione delle competenze. In questo caso infatti l'efficienza stimata fa riferimento principalmente all' aspetto tecnico più che applicativo del sistema, si parlerà infatti nel paragrafo 5.5 di una stima del costo computazionale. Una valutazione sull'efficienza applicativa e dunque sulla capacità del software di fornire vantaggi significativi nell'ambito della predizione di modelli biochimici è lasciata ad eventuali utenti esperti. Stabilita quindi la correttezza ed effettuata l'ottimizzazione del codice è stato necessario effettuare ulteriori test con l'unico scopo di raccogliere dati sulle prestazioni. Un'analisi dettagliata dei risultati sarà esposta nel sopracitato paragrafo 5.5.

L'ultimo obiettivo sarà qui discusso ma non verrà ripreso in nessuno dei successivi paragrafi per i motivi già citati. Esso infatti concerne principalmente l'aspetto applicativo e può essere pienamente conseguito solo da un utente finale del sistema. Tuttavia è rilevante sottolineare, nella fase sperimentale, quale sia l'obiettivo ultimo dell'impiego di un software di questo tipo. Una simulazione realistica e corretta rappresenta uno strumento molto prezioso per lo studio di sistemi complessi.
Osservare l'evoluzione di un sistema biologico durante il progredire di una simulazione che agisce in accordo con le regole definite nel modello SBML ma allo stesso tempo introduce elementi di casualità controllata e riproducibile, può risultare fondamentale per la verifica o la confutazione di predizioni relative al sistema stesso. Partendo da un modello descrittivo di uno specifico sistema biomolecolare e conoscendo la dinamica delle interazioni fra molecole che lo compongono, è possibile studiare le perturbazione che ne modificano lo stato con il progredire della simulazione. Un caso particolarmente rilevante è quello delle simulazioni in campo biomedico dove i sistemi di riferimento consistono in aree interne all'organismo e le specie studiate sono quelle che abitano tali aree.

Homeostatic mechanisms in dopamine synthesis and release

L'esempio qui riportato è quello di un modello matematico che descrive i processi di sintesi, rilascio, ricaptazione ed uso della *Dopamina*, per studiarne l'omeostasi nei terminali dei singoli neuroni. La dopamina è una catecolamina usata come neurotrasmettitore sia dal sistema nervoso centrale che da quello periferico. Disfunzioni in vari sistemi dopaminergici sono note per essere associate con diverse malattie come schizofrenia, sindrome di Parkinson e sindrome di Tourette. Queste conseguenze legate a disfunzioni nei processi di regolazione indicano l'importanza del mantenimento della funzionalità della dopamina che avviene attraverso meccanismi omeostatici attribuiti al delicato equilibrio tra sintesi, immagazzinamento, rilascio, metabolismo e ricaptazione. In questo caso la perturbazione del sistema è dovuta alla mancata inibizione del substrato della tirosina idrossilasi da parte della tirosina che porta ad uno squilibrio nei livelli di dopamina citosolica e vescicolare.



(a) step 0 **(b)** step 10000

Figura 5.1. Aumento non regolato dei livelli di acido omovanillico e dopamina nel citosol.

5.2 Configurazione

La configurazione del sistema è qui riportata nel dettaglio, fornendo un quadro completo dell'ambiente di sviluppo e dei software impiegati sia nel corso di questa fase che nella precedente fase implementativa come anticipato nel capitolo 4. Lo sviluppo e la sperimentazione del programma sono stati eseguiti su un sistema GNU/Linux Debian-based x86_64, all'interno del quale è stato creato un ambiente virtuale, tramite il modulo venv di Python.

Descrizione dell'ambiente virtuale

- Versione Python 3.9.7 con installati i pacchetti:
  - pip 20.3.4
  - pkg-resources 0.0.0
  - python-libsbml 5.19.2
  - setuptools 44.1.1
- Versione di LAMMPS [1] del 29 Oct 2020 con:
  - Compiler: GNU C++ 10.3.0 with OpenMP 4.5
  - C++ standard: C++11
  - MPI v3.1: Open MPI v4.1.1, package: Open MPI
  - ident: 4.1.1, repo rev: v4.1.1, Apr 24, 2021
- Versione Systems biology markup language (SBML) level 3
- Versione OVITO Basic 3.5.4

In secondo luogo sono riportati i passaggi perseguiti allo stadio attuale per una corretta e riproducibile sperimentazione sul sistema:

1. Raccolta di un numero elevato modelli SBML dal portale BioModels;
2. Analisi dei modelli raccolti ed estrapolazione delle informazioni strettamente necessarie al fine di generare una simulazione;
3. Creazione di modelli SBML di testing personalizzati;
4. Impiego dei modelli personalizzati per la verifica della correttezza e l'ottimizzazione;
5. Ricerca e correzione di eventuali errori logici o concettuali, basandosi sull'osservazione dei risultati prodotti dalle simulazioni generate;
6. Esecuzione di esperimenti con l'utilizzo dei modelli originali ottenuti dal portale.

[1] I pacchetti LAMMPS utilizzati corrispondono con quelli compresi nell'installazione standard

In riferimento ai file condivisi pubblicamente nel portale BioModels se ne assume la correttezza. Come è riportato nella home del sito i modelli disponibili sono basati sulla letteratura scientifica esistente[2].

Una lettura dei suddetti modelli verificati è stata eseguita attraverso lo script *create.py*, centrale nel funzionamento del sistema. I dettagli tecnici sul funzionamento di questo script sono già stati trattati nel capitolo 4 relativo all'implementazione. A questa fase di lettura è conseguito un processo di estrapolazione delle informazioni rilevanti per la creazione della simulazione adeguata, anch'esso affidato allo script. Il risultato di questo paesaggio può essere osservato nei file di output denominati *sbml.analysis*, presenti nelle cartelle relative ai singoli esperimenti eseguiti. L'organizzazione dei dati di output è resa esplicita nel capitolo 1 al paragrafo 1.5. Un file *sbml.analysis* contiene una tabella delle specie individuate nel modello ed una lista delle reazioni presenti. L'assunzione di correttezza sui modelli utilizzati non preclude la presenza di un passaggio di controllo sintattico, si presuppone però che, oltre la corretta modellazione del sistema di interesse, il file di input rispetti anche lo standard SBML di livello 3. Questo implica, ad esempio, che ogni specie, così come ogni reazione, sia contrassegnata da un identificatore univoco. Nel caso in cui una stessa specie sia collocabile in più compartimenti diversi si suppone che questa sia stata definita due volte con due *id* distinti e con gli opportuni riferimenti al relativo compartimento. Se così non fosse risulterebbe molto complicato stabilire in quali reazioni sia coinvolta una specie presente in più di un compartimento. Solo nelle reazioni che coinvolgono specie presenti nello stesso compartimento? O in tutte le reazioni che fanno riferimento all'*id* di quella specie?
Da qui la scelta di trattare le specie multi-compartimento come specie formalmente identiche ma distinte. Le diverse interpretazioni del concetto di compartimento sono illustrate sul sito *SBML.org* alla pagina riportata in bibliografia [20] sotto la voce *"Isn't SBML's definition of 'compartment' wrong? A compartment is an amount of substance!"*. Mentre la definizione standard di compartimento, adottata in questo contesto è quella fornita dalla documentazione SBML [3].

La creazione di modelli di testing, sulla base delle informazioni raccolte al passo 2, rappresenta un punto fondamentale della fase sperimentale. Questi sono infatti i modelli utilizzati per la verifica della correttezza e l'ottimizzazione del codice e sono stati scritti rispettando i seguenti criteri:

- i modelli di testing devono, complessivamente, coprire tutti i casi di studio possibili ma limitandosi a quelli verso cui il progetto è indirizzato;
- i modelli di testing devono limitarsi a definire un modello rilevante per lo scopo del test e che soddisfi i requisiti minimi affinché sia considerato corretto per lo standard SBML;
- i modelli di testing devono descrivere modelli semplici la cui evoluzione è in certa misura prevedibile, in modo da facilitare la valutazione di correttezza del sistema dopo l'esecuzione;

[2] per maggiori informazioni consultare il link presente nella bibliografia [24]

Di seguito è riportato l'esempio di un semplice modello personalizzato usato per testare il meccanismo di scontro tra due agenti di specie differenti e la successiva creazione di un nuovo agente di una terza specie. Questo meccanismo simula la reazione *r1* descritta tra la riga 33 e la riga 43, mentre le specie coinvolte sono definite alle righe 20, 22, 24.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <sbml xmlns="http://www.sbml.org/sbml/level3/version1/core" level="3"
       version="1">
3
4    <model substanceUnits="mole" timeUnits="second" extentUnits="mole">
5
6      <listOfUnitDefinitions>
7        <unitDefinition id="per_second">
8          <listOfUnits>
9            <unit kind="second" exponent="-1" scale="0"
10                 multiplier="1"/>
11         </listOfUnits>
12       </unitDefinition>
13     </listOfUnitDefinitions>
14
15     <listOfCompartments>
16       <compartment id="c1" spatialDimensions="3" size="1" units="
       litre" constant="true"/>
17     </listOfCompartments>
18
19     <listOfSpecies>
20       <species id="s1" compartment="c1" initialConcentration="1"
       substanceUnits="mole" boundaryCondition="false" constant="false"/>
21
22       <species id="s2" compartment="c1" initialConcentration="1"
       substanceUnits="mole" boundaryCondition="false" constant="false"/>
23
24       <species id="s3" compartment="c1" initialConcentration="1"
       substanceUnits="mole" boundaryCondition="false" constant="false"/>
25
26     </listOfSpecies>
27
28     <listOfParameters>
29       <parameter id="k" value="1267" units="per_second" constant="
       true"/>
30     </listOfParameters>
31
32     <listOfReactions>
33       <reaction id="r1" reversible="false" fast="false">
34
35         <listOfReactants>
36           <speciesReference species="s1" constant="true"/>
37           <speciesReference species="s2" constant="true"/>
38         </listOfReactants>
39
40         <listOfProducts>
41           <speciesReference species="s3" constant="true"/>
42         </listOfProducts>
43       </reaction>
44
45     </listOfReactions>
46
47   </model>
48 </sbml>
```

I punti 4 e 5 sono stati reiterati più volte, prima di procedere al passo conclusivo, fino a quando non si sono ottenuti risultati concordi con la definizione di correttezza del sistema e si è stabilito che l'ottimizzazione del software fosse la massima possibile. Ogni iterazione è stata caratterizzata dall'esecuzione multipla dello script *create.py* con in *"input"* i file di testing menzionati. I diversi file sono stati usati per simulare tutti i possibili scenari di applicazione del sistema e valutarne così il comportamento. Per ogni esecuzione non andata a buon fine è stata effettuata un'operazione di debugging per la ricerca e la correzione degli errori nel codice. Una volta superata la prova di correttezza si è passato alla fase di ottimizzazione, la quale ha avuto termine in seguito ad un giudizio soggettivo dello sviluppatore che non esclude la possibilità di ulteriori ottimizzazioni e rifinimenti futuri. Questo processo è avvenuto con il supporto di OVITO 5.2, un software di visualizzazione ed analisi di dati di simulazioni atomiche o particellari. Passando ad OVITO i file di output di una simulazione LAMMPS denominati *dump.nome.esperimento.out* è possibile generare un modello 3D delle interazioni tra gli agenti (particelle, atomi o molecole). Questi file *dump* contengono una sequenza di "istantanee" del sistema scattate ogni N passi della simulazione.

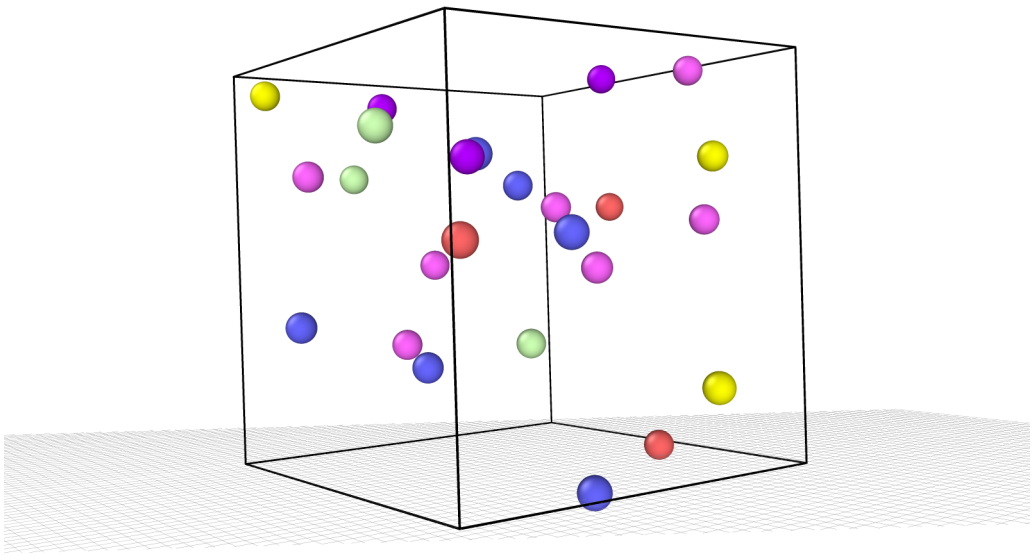


Figura 5.2. Rappresentazione di un modello 3D con OVITO.

Una volta completata la fase di testing è stato possibile procedere con il passo finale della fase sperimentale, ossia con la valutazione dell'efficienza del sistema. Per ottenere una stima realistica riguardo ai costi computazionali, sono stati generati ed attuati molteplici script di simulazioni sui modelli SBML originariamente ottenuti dal portale online. Queste simulazioni rappresentano dunque un esempio realistico di applicazione del software seppure, come già anticipato, la loro finalità è stata unicamente quella di fornire un quadro generale sul funzionamento del programma.

5.3 Casi di Studio

Questo paragrafo risponde alla domanda: quali modelli sono stati presi in considerazione nella fase sperimentale?
Saranno quindi esposte le principali caratteristiche dei modelli SBML impiegati e delineati i criteri per la selezione fra la vasta offerta disponibile su BioModels. Non tutti i tipi di modelli presenti nella pagina *Browse* del portale [24], infatti, sono stati presi in considerazione. In particolare è stata praticata una selezione accurata su file appartenenti alle seguenti sotto categorie:

- Modelli SBML L3 o L2 V4 [3];
- Modelli relativi al Homo Sapiens;
- Modelli basati su equazioni differenziali ordinarie;

I modelli effettivamente selezionati tra quelli soddisfacenti i requisiti richiesti, di cui un esempio è riportato in Fig.5.3, possono essere trovati nella cartella *models*, come indicato nel paragrafo 1.5. I file di testing, presenti anch'essi nella medesima cartella, offrono invece una chiara rappresentazione concettuale dei diversi casi di studio modellati dai file originali in quanto creati allo scopo di riprodurre, ad un livello più astratto, tutti i possibili scenari applicativi del sistema.

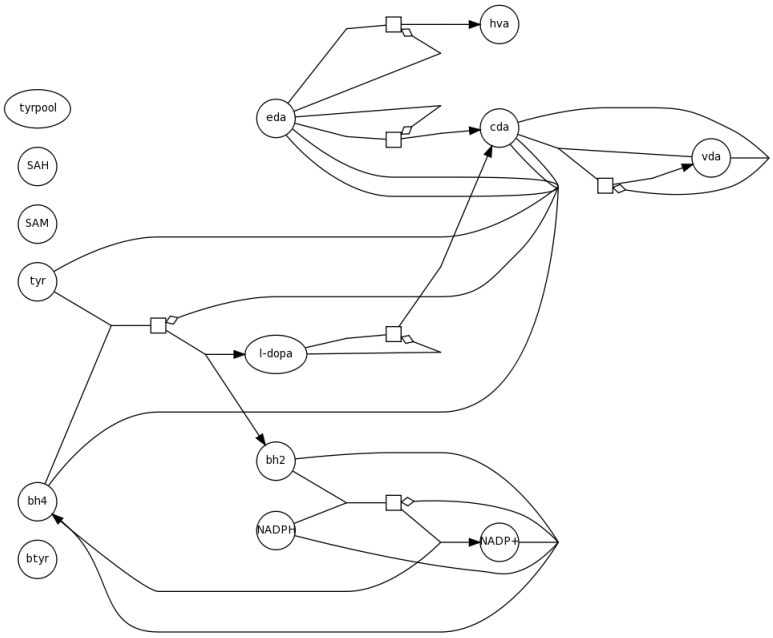


Figura 5.3. Mappa delle reazioni del modello: "Homeostatic mechanisms in dopamine synthesis and release".

[3] Nonostante nel paragrafo sulla configurazione 5.2 sia stato specificato l'utilizzo di SBML L3 a causa dell'attuale scarsità di modelli di questo tipo è stato necessario ricorrere a quelli di una versione precedente, comunque compatibile

Tutti i modelli relativi ai casi di studio considerati contengono la definizione di almeno una o di una combinazione di reazioni appartenenti ai tipi di seguito elencati. Nell'ambito del poggetto non è concepito nessun altro tipo di reazione, né tanto meno sono stati individuati file modellanti una reazione di un tipo non valutato. Pertanto è lecito considerare i casi di studio affrontati come esaustivi. Ogni tipo di reazione qui riportata rappresenta effettivamente un caso di studio affrontato ed è associata al suo corrispondente file di test:

- Crescita di una specie;
- Inibizione di una specie;
- Trasformazione di una specie;
- Reazione tra due specie con una terza specie come prodotto [4];
- Reazione tra più di due specie con una terza specie come prodotto;
- Crescita contemporanea di due o più specie;
- Inibizione contemporanea di due o più specie;
- Reazione tra due specie con più di una specie come prodotto;
- Reazione tra più di due specie con una più di una specie come prodotto;
- Reazione a catena: il prodotto di una reazione e' coinvolto a sua volta in una o più reazioni.

Vi è una differenziazione tra le reazioni che coinvolgono due specie rispetto a quelle che ne coinvolgono tre o più. Questo dipende da una scelta implementativa, illustrata nel capitolo 4 Implementazione, legata ad una restrizione imposta da LAMMPS riguardo la possibilità di creare legami esclusivamente tra coppie di atomi.

[4] Un esempio di questa reazione è illustrato in figura 5.5

5.4 Correttezza

Un sistema software di questo tipo si definisce corretto quando i risultati forniti sono quelli attesi a partire da ben definiti dati di ingresso. I dati utilizzati con lo scopo di verificare la correttezza del sistema, cioè i modelli di testing, sono assunti come ben definiti in quanto estrapolati da dati verificati perché approvati nel portale BioModels. Per quanto riguarda i risultati attesi entra in gioco ancora una volta la sfera di competenza: per effettuare previsioni su risultati di simulazioni di modelli biochimici complessi è necessaria una conoscenza profonda del relativo campo di studi. In questa sede si è preferito affidarsi a modelli semplici, dal comportamento intuitivo e prevedibile, da cui la giustificazione nell'utilizzo di modelli personalizzati. Inoltre, come già accennato nel paragrafo sulla configurazione 5.2, un ruolo chiave nella valutazione degli esperimenti di verifica lo ha giocato il software di rappresentazione 3D OVITO, grazie al quale è stato possibile ottenere una conferma visiva del risultato ottenuto.

La valutazione della correttezza del codice prodotto, quindi del funzionamento del sistema in tutti i suoi aspetti, è stata perseguita sulla base di quattro criteri fondamentali:

1. Correttezza sintattica del codice;
2. Correttezza nella fase di traduzione;
3. Correttezza nella fase di simulazione;
4. Affidabilità dei dati prodotti;

Il primo di questi criteri concerne un aspetto strettamente tecnico, ossia la verifica dell'assenza di errori sintattici nella definizione degli algoritmi e delle strutture dati. Una prova di correttezza in questa caso è data direttamente dall'interprete Python nella fase di esecuzione degli script e dal compilatore GNU C++ nella fase di esecuzione della simulazione.

Verificare la correttezza del sistema nella fase di traduzione di un modello SBML in uno script di simulazione LAMMPS consiste nell'accertare che il codice prodotto sia adeguato ai metodi precedentemente stabiliti. Questi metodi sono esposti e giustificati nel capitolo 3 Metodi, mentre la loro interpretazione pratica è descritta nel capitolo 4 Implementazione. Sono qui riportati in breve alcuni aspetti importanti dei metodi implementativi adottati, al fine di chiarire il processo decisionale sulla valutazione di correttezza. Un primo aspetto consiste, per esempio, nell'astrazione delle molecole coinvolte nella simulazione come "palline" libere di muoversi in uno spazio definito, di seguito chiamate agenti. Vi è poi la considerazione dei legami formatisi tra gli agenti nel corso della simulazione come espressione di reazioni chimiche tra specie in accordo con regole del modello SBML ed infine un'idealizzazione della struttura biochimica dello spazio di simulazione con conseguente interpretazione flessibile delle unità di misura.

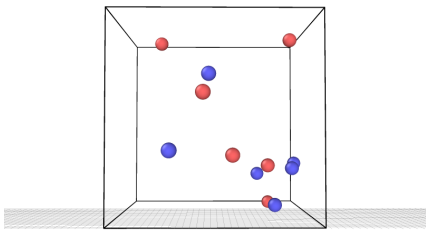
Un corretto svolgimento della simulazione è considerato tale quando questa va a buon fine (5.4), non presenta errori durante l'esecuzione e nel corso del suo svolgimento è possibile osservare l'evoluzione del sistema in accordo con le regole definite dal modello. In questo caso non si ricercano errori evidenziati dal compilatore, quanto errori logici che nel corso della simulazione possono produrre comportamenti imprevisti portando ad una degenerazione inaspettata del sistema.

```
Nlocal:        8.00000 ave           8 max           8 min
Histogram: 1 0 0 0 0 0 0 0 0 0
Nghost:        0.00000 ave           0 max           0 min
Histogram: 1 0 0 0 0 0 0 0 0 0
Neighs:        2.00000 ave           2 max           2 min
Histogram: 1 0 0 0 0 0 0 0 0 0

Total # of neighbors = 2
Ave neighs/atom = 0.25
Ave special neighs/atom = 0.0000000
Neighbor list builds = 1
Dangerous builds = 1
Created 0 atoms
create_atoms CPU = 0.000 seconds
Deleted 0 atoms, new total = 8

INFO
Starting Atoms: 12
Duration: 10000
ALL DONE
Total wall time: 0:00:13
```

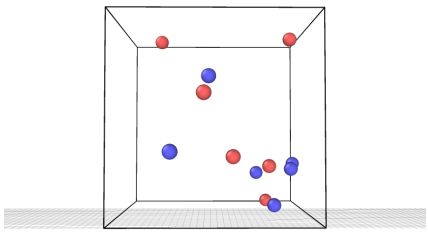
Figura 5.4. Schermata conclusiva di una simulazione LAMMPS andata a buon fine.

Una simulazione corretta, inoltre, produce dati di output validi che permettono analisi a posteriori su ipotesi formulate a priori. I dati di output prodotti dallo script LAMMPS durante una simulazione rappresentano una sequenza ordinata di fotografie del sistema, scattate ogni N passi, e consentono di osservare in un secondo momento l'andamento dell'esperimento in Fig.5.5. Inoltre la possibilità di definire dei *seed* per la generazione di valori casuali rende ogni simulazione riproducibile consentendo il confronto con simulazioni di modelli analoghi o dello stesso modello, con diversi parametri di input. Solo tramite l'attuazione di molteplici esperimenti, eseguiti impiegando modelli molto diversi tra loro è possibile coprire tutti i casi di studio e ottenere riscontri validi sulla correttezza del sistema.

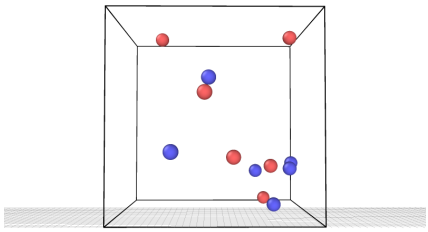
La dimostrazione della correttezza non è stata limitata all'applicazione di modelli di testing. Una volta accertato il corretto comportamento del software si è passati allo studio di file significativamente più complessi, modelli realistici di interazioni molecolari ben definite, i cui elementi di interesse rispecchiassero però quelli dei test utilizzati in precedenza. Per tale motivo questo passo ha avuto principalmente un ruolo di conferma del risultato sulla correttezza già praticamente garantito. Nondimeno una conferma del funzionamento del software nelle condizioni di reale possibile applicazione ha avuto un ruolo determinante nel verificare che nessun caso di studio fosse stato escluso. Inoltre, solo in questo modo è stato possibile accertarsi del fatto che le informazioni in surplus, presenti nei modelli SBML originali ma non in quelli personalizzati, non avessero influito, come previsto, nella generazione di una corretta simulazione. In sostanza, dunque, questo sistema software può essere definito corretto poiché nel momento in cui accetta come dati di ingresso modelli SBML affidabili si è dimostrato in grado di produrre simulazioni coerenti e riproducibili, la cui esecuzione è andata a buon fine e i cui risultati si sono mostrati in accordo con le previsioni.



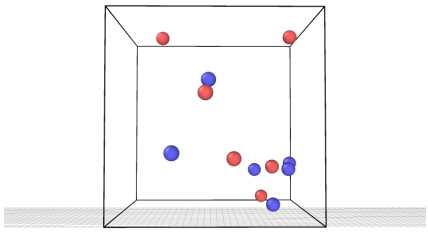
(a) step 250



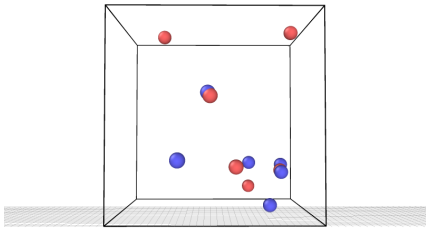
(b) step 260



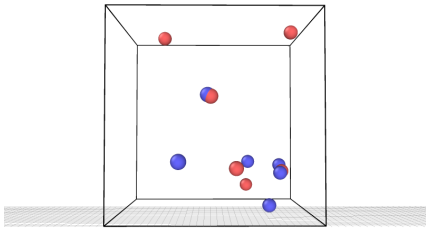
(c) step 270



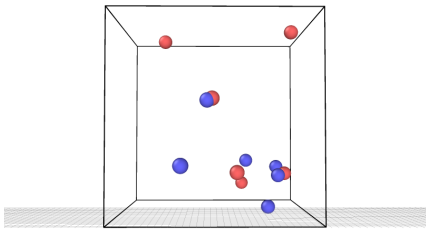
(d) step 280



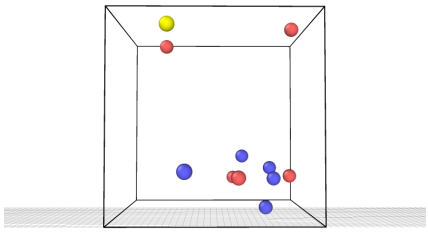
(e) step 290



(f) step 300



(g) step 310



(h) step 320

Figura 5.5. Scontro tra due agenti di tipo a (rosso) e b (blu) con successiva eliminazione dei due e creazione di un terzo agente di tipo c (giallo).

5.5 Stima Computazionale

Una valutazione completa del costo computazionale include un'analisi sia dell'efficienza degli algoritmi impiegati nella fase di generazione dello script LAMMPS, a partire dalla lettura del un documento SBML in ingresso, che della velocità di esecuzione dello script stesso nel momento della simulazione. Entrambi queste funzionalità sono fortemente influenzate da fattori esterni quali la dimensione, o la complessità, dei file di input SBML o la scelta della durata di simulazione da parte dell'utente.

Per quanto riguarda l'aspetto della lettura del modello, gran parte del costo computazionale è affidato alle funzioni della libreria Python libSBML ottimizzata per lo scopo. Il resto della computazione è relativa alla manipolazione dei dati acquisiti al fine di tradurre le informazioni del modello in comandi LAMMPS. Questa operazione ha comportato la costruzione di strutture dati come matrici bidimensionali e dizionari e l'impiego di metodi che su di essi agiscono e il cui tempo di esecuzione è strettamente legato alla dimensione. Un esempio pratico è dato dal dizionario delle reazioni che permette l'associazione di una serie di reagenti con i rispettivi prodotti nell' ambito di una reazione. Qualunque operazione su una tale struttura non può essere effettuata in un tempo inferiore a $\mathcal{O}(k(n+m))$, dove k rappresenta il numero delle reazioni mentre n ed m rispettivamente il numero di reagenti e di prodotti per ogni reazione. Un ragionamento simile è applicabile nei confronti del dizionario delle specie, dove però capita spesso che si voglia accedere ad un valore relativo ad una data specie e questa operazione non sia richiesta per tutte le specie presenti: per esempio non è detto che tutte le specie siano coinvolte in almeno una reazione. Accedere quindi all'*id* di una specie, nell'analisi di una reazione, equivale ad accedere ad un valore del dizionario associato ad una chiave; questa operazione in Python ha una complessità media di $\mathcal{O}(1)$ (vedi [23]). Ancora una volta la complessità dell'algoritmo risulta dunque determinata dal numero di specie coinvolte in una reazione. Una descrizione approfondita di questi algoritmi e delle scelte implementative che ne hanno portato alla scrittura è fornita nel capitolo 4.

Relativamente al secondo aspetto, nella valutazione della complessità di esecuzione di una singola simulazione entrano in gioco fattori distinti, interni ed esterni. Come anticipato, un ruolo rilevante è coperto dalla scelta dell'utente sulla lunghezza della simulazione in termini di *timestep* e di questo dato si può prevedere un valore medio al quale fare riferimento. Nella fase sperimentale, per esempio le simulazioni sono state eseguite con un valore fisso di 10000 *timestep*; questo si è rivelato sufficiente per compiere le necessarie verifiche di correttezza. A questo valore, in termini di passi di simulazione, ha corrisposto una durata media di $0:00:13$ secondi. Un ulteriore fattore rilevante nell'ottenimento di queste prestazioni e' stata la capacità di LAMMPS di eseguire simulazioni parallele su multiprocessori utilizzando un sistema di memoria distribuito. L'impiego del modulo OpenMPI [9], riportato nella descrizione della configurazione del sistema 5.2, ha permesso di sfruttare questa capacità, in particolare attraverso il comando `env OMP_NUM_THREADS=16 lmp -sf omp`, evocato dallo script bash *run.sh* responsabile dell'avvio di ogni simulazione. Infine il fattore con il costo computazionale forse più rilevante nell'ambito della simulazione è rappresentato dagli algoritmi di *Collision Detection* implementati nel codice sorgente di LAMMPS. Grazie all'applicazione di questi algoritmi ed all'impiego di un meccanismo di neighbor list [12], anch'esso implementato da LAMMPS, è stato possibile ottenere informazioni sulle collisioni avvenute nel corso della simulazione in tempo $\mathcal{O}(n)$. Questo è stato fondamentale considerando che ogni collisione può determinare con una certa probabilità l'avvenimento di una reazione.

5.6 Valutazione Tecnica

Come anticipato nell'introduzione al capitolo, non sarebbe possibile effettuare una rigorosa valutazione tecnica senza tenere conto delle competenze biochimiche necessarie per una corretta interpretazione dei dati sperimentali relativi a modelli biochimici complessi. Pertanto la valutazione tecnica in questa sede si limita ad esprimere un giudizio sulle prestazioni del sistema, principalmente in termini dell'efficienza osservata durante la fase sperimentale.
Nei paragrafi precedenti è stato evidenziato come entrambi gli obiettivi di correttezza ed efficienza siano stati portati a termine definendo un sistema che porta a compimento lo scopo previsto in un tempo accettabile. Un futuro raffinamento dei metodi implementativi che non stravolga l'idea alla base dello sviluppo di un simulatore *Agent Based*, dovuto magari ad un avanzamento del potere computazionale del simulatore stesso, non è da escludere e potrebbe comportare un significativo miglioramento nelle prestazioni del sistema. Le valutazioni tecniche fanno riferimento ad un software considerato ottimizzato e che comunque non ha deluso le aspettative. Come infatti preannunciato nel paragrafo 5.5, tutte le simulazioni di test sono terminate in un tempo inferiore al secondo. Anche per quanto riguarda simulazioni più complesse i tempi di esecuzione non sono mai aumentati significativamente, mostrando che l'unico fattore veramente rilevante a livello di impatto sulle performance è la durata della simulazione. Simulazioni più lunghe sono state eseguite ponendo il valore del massimo timestep a 100000 ed hanno impiegato anche un tempo di 0 : 01 : 32 secondi a terminare. Di seguito è riportata un immagine con il recap delle performance di LAMMPS nell'esecuzione di uno dei modelli di testing, per la precisione il modello *complete.xml* contenente la definizione di tutte le possibili reazioni ed un numero notevole di specie. Nell'immagine viene indicato l'impiego totale della CPU ed è riportata una tabella con i tempi di esecuzione di ogni task richiesto dal simulatore. Si può inoltre notare come l'esecuzione di un singolo task venga ripartita su 16 thread.

```
Setting up Verlet run ...
Unit style    : lj
Current step  : 99900
Time step     : 0.001
Per MPI rank memory allocation (min/avg/max) = 39.83 | 39.83 | 39.83 Mbytes
Step Temp PotEng c_countr1
99900    317.32341           0           0
100000    324.36177           0           0
Loop time of 0.0126833 on 16 procs for 100 steps with 8 atoms

Performance: 681212.283 tau/day, 7884.401 timesteps/s
411.5% CPU use with 1 MPI tasks x 16 OpenMP threads

MPI task timing breakdown:
Section |  min time  |  avg time  |  max time  |%varavg| %total
---------------------------------------------------------------
Pair    | 1.1456e-05 | 1.1456e-05 | 1.1456e-05 |   0.0 |  0.09
Bond    | 0.0043148  | 0.0043148  | 0.0043148  |   0.0 | 34.02
Neigh   | 2.2468e-05 | 2.2468e-05 | 2.2468e-05 |   0.0 |  0.18
Comm    | 2.0548e-05 | 2.0548e-05 | 2.0548e-05 |   0.0 |  0.16
Output  | 0.00024845 | 0.00024845 | 0.00024845 |   0.0 |  1.96
Modify  | 0.0080175  | 0.0080175  | 0.0080175  |   0.0 | 63.21
Other   |            | 4.81e-05   |            |       |  0.38
```

Figura 5.6. Informazioni visualizzate a schermo da LAMMPS durante la simulazione, si può notare come sia riportato l'utilizzo complessivo della CPU diviso tra i sotto processi e il tempo impiegato per ogni task.

Per quanto riguarda i tempi di esecuzione dello script *create.py* si sono rivelati talmente esigui, anche nel caso di modelli di input molto grandi, tali da non essere stati presi in considerazione. Il seguente grafico (Fig.5.7) rappresenta invece le prestazioni delle CPU e l'impiego di RAM nel corso dell'esecuzione dell'intero sistema partendo dalla fase di lettura del documento SBML, passando per la fase di traduzione in codice LAMMPS, per concludersi con l'esecuzione della simulazione generata. Si può notare come la percentuale d' impiego delle CPU nel tempo presenti un picco sopra il 40%, condiviso dalle curve di tutti i processori, nel momento dell'avvio della simulazione. Questo avviene perché LAMMPS è eseguito in modo da svolgere la simulazione in parallelo su più processori con memoria condivisa. Si osserva inoltre come l'impiego di RAM rimanga minimo e costante nel tempo.

Figura 5.7. I grafici mostrano l'andamento dell'impiego di RAM e CPU durante lo svolgimento di una simulazione complessa della durata di 10000 timestep.

Capitolo 6

Conclusioni

Le potenzialità dell'applicazione di questo software nel campo delle simulazioni *Agent Based* di *Molecular Dynamics* sono da tenere in conto. L'obiettivo di costruire un simulatore LAMMPS per modelli SBML è stato parzialmente raggiunto. Nonostante le simulazioni generate seguano fedelmente alcune delle regole prestabilite dal modello non tutte queste regole sono tenute in considerazione. La necessità, per esempio, di semplificare le reazioni biochimiche analizzate al fine di poter effettuare una traduzione più diretta fra il linguaggio del modello e quello della simulazione, ha reso determinati aspetti delle stesse impossibili da replicare. Un esempio di ciò è dato dai modificatori delle reazioni che, in SBML, ne stabiliscono la velocità di esecuzione. Questo aspetto è stato tralasciato nel momento della trasposizione a codice LAMMPS, perdendo così il controllo sul tempo di reazione. Scelte di questo tipo sono dovute soprattutto al fatto che LAMMPS è ottimizzato per simulare strutture molecolari (come polimeri o catene di amminoacidi) ma non un numero elevato di reazioni simultanee tra numerose specie diverse. Dunque è stato necessario applicare un certo livello di astrazione dal modello biologico iniziale, interpretare le molecole come "palline" o agenti (atomi in LAMMPS) ed ancora le reazioni come lo scontro e la formazione di legami tra questi agenti.

Per quanto riguarda la correttezza del sistema, facendo questa riferimento ad uno standard prefissato il quale tiene conto del livello di astrazione sopra citato, non si può dire che sia migliorabile nel senso che non si può rendere il sistema più corretto di quanto già non lo sia. Solo modificando il metodo implementativo si potrebbe aumentare lo standard di correttezza portando alla definizione di simulazioni ancora più realistiche e dunque ancora più corrette dal punto di vista biologico. Nonostante questo allontanamento dal realismo dei modelli va però riconosciuto il merito del sistema nel creare simulazioni intuitive la cui applicazione può fornire risultati reali e validi. Un esempio di questo è nel paragrafo 5.1 di questo capitolo. Non è inoltre da sottovalutare la capacità del sistema di accogliere in input ed interpretare modelli SBML disparati. Questo permette di non escludere la possibilità, in futuro, di estendere il campo di applicazione del sistema ad altri modelli rappresentabili in un contesto *Agent Based*, la cui struttura non differisca di molto da quella dei modelli biochimici.
Per questi motivi la valutazione tecnica sul sistema può essere considerata complessivamente positiva, sia rispetto alle possibili applicazioni pratiche che riguardo alle potenzialità dimostrate.

Bibliografia

[1] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, S. J. Plimpton, *LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales*, Comp Phys Comm, 271 (2022) 10817.

[2] Gerd Gruenert, Peter Dittrich, *Using the SRSim Software for Spatial and Rule-Based Modeling of Combinatorially Complex Biochemical Reaction Systems*, Conference Paper, August 2010

[3] Hucka, Bergmann, Chaouiya, Dräger, Hoops, Keating, König, Le Novère, Myers, Olivier, Sahle, Schaff, Sheriff, Smith, Waltemath, Wilkinson,Zhang, "*The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 2 Core*", Release 2, 29 March 2019, sbml.org

[4] Klipp, Edda Kowald, Axel Liebermeister, Wolfram Wierling, Christoph Wiley, "*Systems Biology a Textbookt*", Second Completely Revised and Enlarged Edition, Wiley-VCH 2016

[5] Grazziela P. Figueredo, Peer-Olaf Siebers, Markus R. Owen, Jenna Reps, Uwe Aickelin, "*Comparing Stochastic Differential Equations and Agent-Based Modelling and Simulation for Early-Stage Cancer*", 1 School of Computer Science, The University of Nottingham, Nottingham, United Kingdom, 2 Centre for Mathematical Medicine and Biology, School of Mathematical Sciences, The University of Nottingham, Nottingham, United Kingdom

[6] LAMMPS Molecular Dynamics Simulator, https://www.lammps.org/

[7] LAMMPS Documentation (17 Feb 2022 version), https://docs.lammps.org/Manual.html

[8] Build LAMMPS with CMake, https://docs.lammps.org/Build_cmake.html

[9] Building LAMMPS with the OPENMPI package, https://docs.lammps.org/Speed_omp.html

[10] [Environment Setup 14] Build LAMMPS molecular dynamics simulation code and link it to VTK toolkit, https://www.youtube.com/watch?v=Id3eVPDinDE

[11] Write your first LAMMPS Input script Molecular dynamics simulations, https://www.youtube.com/watch?v=gauP_ibET-A

[12] What does the Neighbor command do in LAMMPS, https://mattermodeling.stackexchange.com/questions/1646/what-does-the-neighbor-command-do-in-lammps

[13] Conditional Statment, https://matsci.org/t/conditional-statment/34266

[14] Fix deposit command, https://docs.lammps.org/fix_deposit.html

[15] Fix bond/create command, https://docs.lammps.org/fix_bond_create.html

[16] Compute property/atom command, https://docs.lammps.org/compute_property_atom.html

[17] Variable command, https://docs.lammps.org/variable.html

[18] Group command, https://docs.lammps.org/group.html

[19] SBML, https://synonym.caltech.edu/

[20] SBML Frequently Asked Questions, https://synonym.caltech.edu/documents/faq/

[21] libSBML documentation, https://synonym.caltech.edu/software/libsbml/libsbml-docs/

[22] Python-libsbml 5.19.3, https://pypi.org/project/python-libsbml/

[23] Time Complexity https://wiki.python.org/moin/TimeComplexity

[24] BioModels, https://www.ebi.ac.uk/biomodels/

[25] Best2009 - Homeostatic mechanisms in dopamine synthesis and release, https://www.ebi.ac.uk/biomodels/MODEL1502230000#Overview